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

Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)

Winter Semester 2015/2016

Munich, Germany

Editors

Georg Carle, Daniel Raumer, Lukas Schwaighofer

Publisher

Chair of Network Architectures and Services

Network Architectures and Services NET 2016-07-1

Network Architectures
and Services
NET 2016-07-1

FI & IITM
WS 15/16

Proceedings zu den Seminaren Future Internet (FI) und Innovative Internet Technologien und Mobilkommunikation (IITM)

Wintersemester 2015/2016

München, 5. 10. 2015 – 29. 02. 2016

Editoren: Georg Carle, Daniel Raumer, Lukas Schwaighofer

Organisiert durch den Lehrstuhl Netzarchitekturen und Netzdienste (I8),
Fakultät für Informatik, Technische Universität München

Technische Universität München

Proceedings of the Seminars
Future Internet (FI), and Innovative Internet Technologies and Mobile Communication Networks (IITM)
Winter Semester 2015/2016

Editors:

Georg Carle
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
85748 Garching b. München, Germany
E-mail: carle@net.in.tum.de
Internet: `http://www.net.in.tum.de/~carle/`

Daniel Raumer
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
E-mail: raumer@net.in.tum.de
Internet: `http://www.net.in.tum.de/~raumer/`

Lukas Schwaighofer
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
E-mail: schwaighofer@net.in.tum.de
Internet: `http://www.net.in.tum.de/~schwaighofer/`

Cataloging-in-Publication Data

Seminars FI & IITM WS 15/16
Proceedings zu den Seminaren „Future Internet“ (FI) und „Innovative Internettechnologien und Mobilkkommunikation“ (IITM)
München, Germany, 5. 10. 2015 – 29. 02. 2016
ISBN: 978-3-937201-51-1

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2016-07-1
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2016-07-1
Series Editor: Georg Carle, Technische Universität München, Germany

Vorwort

Vor Ihnen liegen die Proceedings des Seminares „Future Internet“ (FI) und des Seminares „Innovative Internettechnologien und Mobilkommunikation“ (IITM). Wir sind stolz, Ihnen Ausarbeitungen zu aktuellen Themen, die im Rahmen unserer Seminare im Wintersemester 2015/2016 an der Fakultät für Informatik der Technischen Universität München verfasst wurden, präsentieren zu dürfen. Den Teilnehmerinnen und Teilnehmern stand es wie in der Vergangenheit frei, das Paper und den Vortrag in englischer oder in deutscher Sprache zu verfassen. Dementsprechend finden sich sowohl englische als auch deutsche Paper in diesen Proceedings.

Unter allen Themen, die sich mit Aspekten der Computernetze von morgen befassen, verliehen wir in jedem der beiden Seminare einen Best Paper Award. Im IITM Seminar ging dieser an Herrn Julius Michaelis, der in seiner Ausarbeitung „Middlebox Models in Network Verification Research“ einen Überblick über Modelle zur Verifikation von Middleboxes gibt. Im FI wurde dieser Herrn Michael Eder verliehen für seine Ausabreitung „Hypervisor- vs. Container-based Virtualization“ in welcher er Vor- und Nachteile von hypervisorbasierter mit containerbasierter Virtualisierung verglich.

Einige der Vorträge wurden aufgezeichnet und sind auf unserem Medienportal unter `http://media.net.in.tum.de` abrufbar.

Im Seminar FI wurden Beiträge zu aktuellen Themen der Netzwerkforschung vorgestellt. Die folgenden Themen wurden abgedeckt:

- Hypervisor- und Container-based Virtualisierung
- Projekte für weltweiten Internetzugang
- Vergleich von Hardware- und Software-Traffic-Generatoren und ihrem Einsatz in der Praxis

Auf `http://media.net.in.tum.de/#%23Future%20Internet%23WS15` können die aufgezeichneten Vorträge zu diesem Seminar abgerufen werden.

Im Seminar IITM wurden Vorträge aus dem Themenbereich der Netzwerktechnologien inklusive Mobilkommunikationsnetze vorgestellt. Die folgenden Themen wurden abgedeckt:

- Das Interface für Routing Systeme
- Performancevergleiche von Data Plane Devices
- MoonGen Tutorial
- Middlebox Models und Netzwerkverifikation
- Topologieerkennung in kontrollierten Umgebungen
- PFQ: Ein Framework für hochperformante Netzwerk E/A

Auf `http://media.net.in.tum.de/#%23IITM%23WS15` können die aufgezeichneten Vorträge zu diesem Seminar abgerufen werden.

Wir hoffen, dass Sie den Beiträgen dieser Seminare wertvolle Anregungen entnehmen können. Falls Sie weiteres Interesse an unseren Arbeiten habe, so finden Sie weitere Informationen auf unserer Homepage `http://www.net.in.tum.de`.

München, Juli 2016

Georg Carle

Daniel Raumer

Lukas Schwaighofer

Preface

We are pleased to present you the interesting program of our seminars on “Future Internet” (FI) and “Innovative Internet Technologies and Mobil Communication” (IITM) which took place in the winter semester 2015/2016. In both seminar courses the authors were free to write their paper and give their talk in English or German.

We honored the best paper of both seminars with an award. This semester the award in the IITM seminar was given to Mr Julius Michaelis who presented an overview to existing models for network verification in his paper “Middlebox Models in Network Verification Research”. In the FI seminar the award was given to Mr Michael Eder for his paper “Hypervisor- vs. Container-based Virtualization” wherin he discussed advantages and disadvantages ob both virtualization techniques.

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

In the seminar FI we dealt with issues and innovations in network research. The following topics were covered:

- Hypervisor- vs. Container-based Virtualization
- Analyzing "Global Access to the Internet for All" Projects
- Comparison of Hardware and Software-based Traffic Generation and its Practical Application

Recordings can be accessed on `http://media.net.in.tum.de/#%23Future%20Internet%23WS15`.

In the seminar IITM we dealt with different topics in the area of network technologies, including mobile communication networks. The following topics were covered:

- The Interface to the Routing System
- How-To Compare Performance of Data Plane Devices
- MoonGen Tutorial
- Middlebox Models in Network Verification Research
- Topology Discovery in Controlled Environments
- Comparing PFQ: A High-Speed Packet IO Framework

Recordings can be accessed on `http://media.net.in.tum.de/#%23IITM%23WS15`.

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

Munich, July 2016

Seminarveranstalter

Lehrstuhlinhaber

Georg Carle, Technische Universität München, Germany (I8)

Seminarleitung

Daniel Raumer, Technische Universität München, Germany
Lukas Schwaighofer, Technische Universität München, Germany

Betreuer

Edwin Cordeiro (cordeiro@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Cornelius Diekmann (diekmann@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Paul Emmerich (emmericp@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Sebastian Gallenmüller (gallenmu@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Holger Kinkelin (kinkelin@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Daniel Raumer (raumer@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Florian Wohlfart (wohlfart@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Seminarhomepage

`http://www.net.in.tum.de/de/lehre/ws15/seminare/`

Inhaltsverzeichnis

Seminar Future Internet

Hypervisor- vs. Container-based Virtualization 1
Michael Eder (Betreuer: Holger Kinkelin)
Analyzing "Global Access to the Internet for All" Projects 9
Sascha Rushing (Betreuer: Edwin Cordeiro)
Vergleich von Hardware- und Software-Traffic-Generatoren und ihrem Einsatz in der Praxis 17
Tobias Weiher (Betreuer: Paul Emmerich, Daniel Raumer)

Seminar Innovative Internet Technologien und Mobilkommunikation

The Interface to the Routing System 25
Elias Hazboun (Betreuer: Edwin Cordeiro)
How-To Compare Performance of Data Plane Devices 33
Matthias Holdorf (Betreuer: Daniel Raumer, Florian Wohlfart)
MoonGen Tutorial 41
Jonas Jelten (Betreuer: Paul Emmerich, Daniel Raumer)
Middlebox Models in Network Verification Research 47
Julius Michaelis (Betreuer: Cornelius Diekmann)
Topology Discovery in controlled environments 55
Maximilian Pudelko (Betreuer: Florian Wohlfart, Sebastian Gallenmüller)
Comparing PFQ: A High-Speed Packet IO Framework 61
Dominik Schöffmann (Betreuer: Sebastian Gallenmüller)

Hypervisor- vs. Container-based Virtualization

Michael Eder
Betreuer: Holger Kinkelin
Seminar Future Internet WS2015/16
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: edermi@in.tum.de

ABSTRACT

For a long time, the term *virtualization* implied talking about hypervisor-based virtualization. However, in the past few years container-based virtualization got mature and especially Docker gained a lot of attention. Hypervisor-based virtualization provides strong isolation of a complete operating system whereas container-based virtualization strives to isolate processes from other processes at little resource costs. In this paper, hypervisor and container-based virtualization are differentiated and the mechanisms behind Docker and LXC are described. The way from a simple chroot over a container framework to a ready to use container management solution is shown and a look on the security of containers in general is taken. This paper gives an overview of the two different virtualization approaches and their advantages and disadvantages.

Keywords

virtualization, Docker, Linux containers, LXC, hypervisor, security

1. INTRODUCTION

The paper compares two different virtualization approaches, hypervisor and container-based virtualization. Container-based virtualization got popular when Docker [1], a free tool to create, manage and distribute containers gained a lot of attention by combining different technologies to a powerful virtualization software. By contrast, hypervisor-based virtualization is the alpha male of virtualization that is widely used and around for decades. Both technologies have advantages over each other and both come with tradeoffs that have to be taken into account before deciding which of the both technologies better fits the own needs. The paper introduces hypervisor- and container-based virtualization in Section 2, describes advantages and disadvantages in Section 3 and goes deeper into container-based virtualization and the technologies behind in Section 4. There is already a lot of literature about hypervisor-based virtualization whereas container-based virtualization started to get popular in the last few years and there are fewer papers about this topic around, so the main focus of this paper is container-based virtualization. Another focus of the paper is Docker which is introduced in Section 4.3, because it traversed a rapid development over the last two years and gained a lot of attention in the community. To round the paper out, general security risks of container-based virtualization and Docker and possible ways to deal with them are elaborated in Section 5. Section 6 gives a brief overview to related work on this topic.

2. DISTINCTION: HYPERVISOR VS. CONTAINER-BASED VIRTUALIZATION

When talking about virtualization, the technology most people refer to is hypervisor-based virtualization. The hypervisor is a software allowing the abstraction from the hardware. Every piece of hardware required for running software has to be emulated by the hypervisor. Because there is an emulation of the complete hardware of a computer, talking about *virtual machines* or *virtual computers* is usual. It is common to be able to access real hardware through an interface provided by the hypervisor, for example in order to access files on a physical device like a CD or a flash drive or to communicate with the network. Inside this virtual computer, an operating system and software can be installed and used like on any normal computer. The hardware running the hypervisor is called the *host* (and the operating system *host operating system*) whereas all emulated machines running inside them are referred to as *guests* and their operating systems are called *guest operating systems*. Nowadays, it is usual to get also utility software together with the hypervisor that allows convenient access to all of the hypervisor's functions. This improves the ease of operation and may bring additional functionalities that are not exactly part of the hypervisor's job, for example snapshot functionalities and graphical interfaces. It is possible to differentiate two types of hypervisors, type 1 and type 2 hypervisors. Type 1 hypervisors are running directly on hardware (hence often referred to as *bare metal hypervisors*) not requiring an operating system and having their own drivers whereas type 2 hypervisors require a host operating system whose capabilities are used in order to perform their operations. Well-known hypervisors or virtualization products are:

- KVM [2], a kernel module for the Linux kernel allowing access to virtualization capabilities of real hardware and the emulator usually used with KVM, qemu [3], which is emulating the rest of the hardware (type 2),
- Xen [4], a free hypervisor running directly on hardware (type 1),
- Hyper-V [5], a hypervisor from Microsoft integrated into various Windows versions (type 1),
- VMware Workstation [6], a proprietary virtualization software from VMware (type 2)
- Virtual Box [7], a free, platform independent virtualization solution from Oracle (type 2).

From now on, this paper assumes talking about type 2 hypervisors when talking about hypervisors.
Container-based virtualization does not emulate an entire computer. An operating system is providing certain features to the container software in order to isolate processes from other processes and containers. Because the Linux kernel provides a lot of capabilities to isolate processes, it is required by all solutions this paper is dealing with. Other operating systems may provide similar mechanisms, for example FreeBSD's jails [8] or Solaris Zones. Because there is no full emulation of hardware, software running in containers has to be compatible with the host system's kernel and CPU architecture. A very descriptive picture for container-based virtualization is that "containers are like firewalls between processes" [9]. A similar metaphor for hypervisor-based virtualization are processes running on different machines connected to and supervised by a central instance, the hypervisor.

3. USE CASES AND GOALS OF BOTH VIRTUALIZATION TECHNOLOGIES

Hypervisor and container-based virtualization technologies come with different tradeoffs, hence there are different goals each want to achieve. There are also different use cases for virtualization in general and both hypervisor and container-based virtualization have therefore special strengths and weaknesses relating to specific use cases. Because of abstracting from hardware, both types of virtualization are easy to migrate and allow a better resource utilization, leading to lower costs and saving energy.

3.1 Hypervisor-based virtualization

Hypervisor-based virtualization allows to fully emulate another computer, therefore it is possible to emulate other types of devices (for example a smartphone), other CPU architectures or other operating systems. This is useful for example when developing applications for mobile platforms — the developer can test his application on his development system without the need of physically having access to a target device. Another common use case is to have virtual machines with other guest operating systems than the host. Some users need special software that does not run on their preferred operating system, virtualization allows to run nearly every required environment and software in this environment independently from the host system. Because of the abstraction from the hardware, it is easier to create, deploy and maintain images of the system. In the case of hardware incidents, a virtual machine can be moved to another system with very little effort, in the best case even without the user noticing that there was a migration. Hypervisor-based virtualization takes advantage of modern CPU capabilities. This allows the virtual machine and its applications to directly access the CPU in an unprivileged mode [10], resulting in performance improvements without sacrificing the security of the host system.

Hypervisors may set up on the hardware directly (type 1) or on a host operating system (type 2). Figure 1 shows a scheme where the hypervisor is located in this hierarchy. Assuming a type 1 Hypervisor, all operating systems were guests in Figure 1 whereas a type 2 hypervisor was on the same level than other userspace applications, having the operating system (not shown in the figure) and the real hardware on the layers below it.

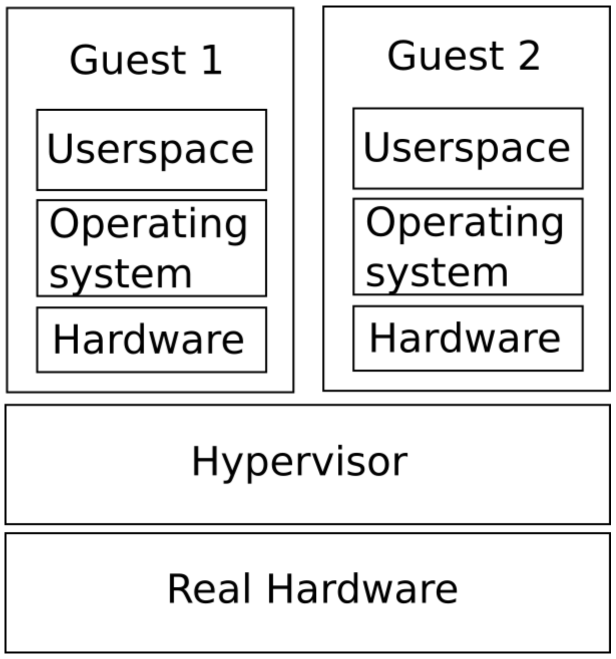

Figure 1: Scheme of hypervisor-based virtualization. Hardware available to guests is usually emulated.

3.2 Container-based virtualization

Container-based virtualization utilizes kernel features to create an isolated environment for processes. In contrast to hypervisor-based virtualization, containers do not get their own virtualized hardware but use the hardware of the host system. Therefore, software running in containers does directly communicate with the host kernel and has to be able to run on the operating system (see figure 2) and CPU architecture the host is running on. Not having to emulate hardware and boot a complete operating system enables containers to start in a few milliseconds and be more efficient than classical virtual machines. Container images are usually smaller than virtual machine images because container images do not need to contain a complete toolchain to run a operating system, i.e. device drivers, kernel or the init system. This is one of the reasons why container-based virtualization got more popular over the last few years. The small resource fingerprint allows better performance on a small and larger scale and there are still relatively strong security benefits.

Shipping containers is a really interesting approach in software developing: Developers do not have to set up their machines by hand, instead of that a build system image may be distributed containing all tools and configuration required for working on a project. If an update is required, only one image has to be regenerated and distributed. This approach eases the management of different projects on a developers machine because one can avoid dependency conflicts and separate different projects in an easy way.

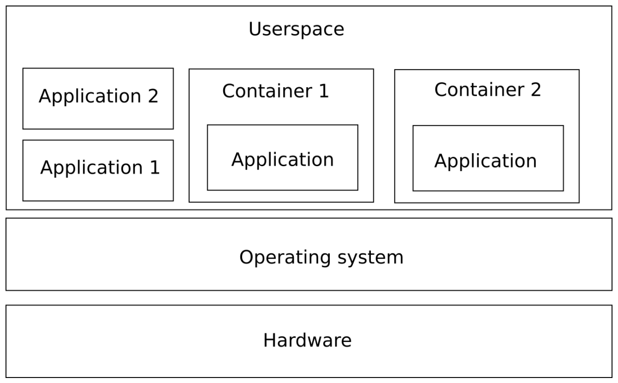

Figure 2: Scheme of container-based virtualization. Hardware and Kernel of the host are used, containers are running in userspace separated from each other.

4. FROM CHROOT OVER CONTAINERS TO DOCKER

Container-based virtualization uses a lot of capabilities the kernel provides in order to isolate processes or to achieve other goals that are helpful for this purpose, too. Most solutions build upon the Linux kernel and because the paper's focus lies on LXC and Docker, a closer look at the features provided by the Linux kernel in order to virtualize applications in containers will be taken. Because the kernel provides most of the capabilities required, container toolkits usually do not have to implement them again and the kernel code is already used by other software that is considered stable and already in production use. In case of security problems of the mechanisms provided by the kernel, fixes are getting distributed with kernel updates, meaning that the container software does not need to be patched and the user only has to keep its kernel up to date.

4.1 chroot

The **ch**ange **root** mechanism allows to change the root directory of a process and all of its subprocesses. Chroot is used in order to restrict filesystem access to a single folder which is treated as the root folder (/) by the target process and its subprocesses. On Linux, this is not considered a security feature because it is possible for a user to escape the chroot [11].

Apart from that, chroot may prevent erroneous software from accessing files outside of the chroot and — even if it is possible to escape the chroot — it makes it harder for attackers to get access to the complete filesystem. Chroot is often used for testing software in a somehow isolated environment and to install or repair the system. This means chroot does not provide any further process isolation apart from changing the root directory of a process to a different directory somewhere in the filesystem.

Compared to a normal execution of processes, putting them into a chroot is a rather weak guarantee that they are not able to access places of the filesystem they are not supposed to access.

4.2 Linux containers

Linux containers [12] (LXC) is a project that aims to build a toolkit to isolate processes from other processes. As said, chroot was not developed as a security feature and it is possible to escape the chroot — LXC tries to create environments (*containers*) that are supposed to be escape-proof for a process and its child processes and to protect the system even if an attacker manages to escape the container. Apart from that, it provides an interface to fine-grained limit resource consumption of containers. Containers fully virtualize network interfaces and make sure that kernel interfaces may only be accessed in secure ways. The following subsections are going to show the most important isolation mechanisms in greater detail. The name *Linux containers* (*LXC*) may be a little bit confusing: It is a toolkit to create containers on Linux, of course there are other possibilities to achieve the same goal with other utilities than LXC and to create isolated processes or groups of them (*containers*) under Linux. Because LXC is a popular toolkit, most of the statements of this paper referring to *Linux containers* apply to LXC, too, but we are going to use the term *LXC* in order to specifically talk about the popular implementation and *Linux containers* in order to talk about the general concept of process isolation as described in this paper on Linux. The capabilities introduced in the following are difficult to use meaning that proper configuration requires a lot of reading documentation and much time setting everything up. Deploying complex configuration of system-level tools is a hard task. Easily adopting the configuration to other environments and different needs without endangering security of the existing solution is nearly impossible without a toolkit providing access to those features. LXC is a toolkit trying to fulfill these requirements.

4.2.1 Linux kernel namespaces

By now, processes are only chrooted but still see other processes and are able to see information like user and group IDs, access the hostname and communicate with others. The goal is to create an environment for a process that allows him access to a special copy of this information that does not need to be the same that other processes see, but the easy approach of preventing the process to access this information may crash it or lead to wrong behaviour. The kernel provides a feature called namespaces (in fact, there are several different [13] namespaces) in order to realize these demands. Kernel namespaces is a feature allowing to isolate processes, groups of processes and even complete subsystems like the interprocess communication or the network subsystem of the kernel. It is a flexible mechanism to separate processes from each other by creating different namespaces for processes that need to be separated. The kernel allows passing global resources such as network devices or process/user/group IDs into namespaces and manages synchronization of those resources. It is possible to create namespaces containing processes that have a process ID (PID) that is already in use on the host system or other containers. This simplifies migration of a suspended container because the PIDs of the container are independent from the PIDs of the host system. User namespaces allow to "isolate security-related identifiers and attributes, in particular, user IDs and group IDs [. . .], the root directory, keys [. . .], and capabilities" [14]. Combining all these features makes it possible to isolate processes in a way that

- process IDs inside namespaces are isolated from process IDs of other namespaces and are unique per-namespace, but processes in different namespaces may have the same process IDs,
- global resources are accessible via an API provided by the kernel if desired,
- there is abstraction from users, groups and other security related information of the host system and the containers.

In fact, kernel namespaces are the foundation of process separation and therefore one of the key concepts for implementing container-based virtualization. They provide a lot of features required for building containers and are available since kernel 2.6.26 which means they are around for several years and are used in production already [15].

4.2.2 Control groups

Control groups (further referred to as cgroups) are a feature that is not mandatory in order to isolate processes from other processes. In the first place cgroups is a mechanism to track processes and process groups including forked processes. [16] In the first instance, they do not solve a problem that is related to process isolation or making the isolation stronger, but provided hooks allow other subsystems to extend those functionalities and to implement fine-grained resource control and limitation which is more flexible compared to other tools trying to achieve a similar goal. The ability to assign resources to processes and process groups and manage those assignments allows to plan and control the use of containers without unrestricted waste of physical resources by a simple container. The same way it is possible to guarantee that resources are not unavailable because other processes are claiming them for themselves. At first this might look like a feature primarily targeting at hosts serving multiple different users (e.g. shared web-hosting), but it is also a powerful mechanism to avoid Denial of Service (DoS) attacks. DoS attacks do not compromise the system, they try to generate a useless workload preventing a service to fulfill its task by overstressing it. A container behaving different than suspected may have an unknown bug, be attacked or even controlled by an attacker and consuming a lot of physical resources — having limited access to those resources from the very beginning avoids outages and may safe time, money and nerves of all those involved. As said, controlling resources is not a feature required for isolation but essential in order to compete with hypervisor-based virtualization. It is very common to restrict resource usage in hypervisor-based virtualization and being able to do the same with containers allows to better utilize the given resources.

4.2.3 Mandatory Access Control

On Linux (and Unix), everything is a file and every file has related information about which user and which group the file belongs to and what the owner, the owning group and everybody else on the system is allowed to do with a file. *Doing* in this context means: reading, writing or executing. This *Discretionary Access Control* (*DAC*) decides if access to a resource is granted solely on information about users and groups. Contrary, *Mandatory Access Control* (*MAC*) does not decide on those characteristics. Mandatory Access Control is a set of concepts defining management of access control which is gaining more and more attention over the past years. The need for such systems is generally to improve security and in this case MAC is used to harden the access control mechanisms of Linux/Unix. Instead, if a resource is requested, authorization rules (*policies*) are checked and if the requirements defined by the policy are met access is granted. There are different approaches on how to implement such a system, the three big ones are namely SELinux [17], AppArmor [18] and grsecurity's RBAC [19]. MAC policies are often used in order to restrict access to resources that are sensitive and not required to be accessed in a certain context. For example the *chsh*[1] userspace utility on Linux has the setuid bit set. This means that a regular user is allowed to run this binary and the binary can elevate it self to running with root privileges in order to do what it is required to do. If a bug was found in the binary it may be possible to execute malicious code with root privileges as a normal user. A MAC policy could be provided allowing the binary to elevate its rights in order to do its legitimate job but denying everything root can do but the library does not need to do. In this case, there is no use case for the chsh binary to do networking but it is allowed to because it can do anything root can do. A policy denying access to network related system components does not affect the binary but if it was exploited by an attacker, he wouldn't be able to use the binary in order to sniff traffic on active network interfaces or change the default gateway to his own box capturing all the packets because a MAC policy is denying the chsh tool to access the network subsystem. There are a lot of examples where bugs in software let attackers access sensitive information or do malicious activities that are not required by the attacked process, e.g. CVE–2007–3304[2] which is a bug in the Apache webserver allowing an attacker to send signals to other processes which gets mitigated already by enabling SELinux in enforcing mode [10]. Especially network services like web- or mail servers, but also connected systems like database servers are receiving untrusted data and are potential attack targets. Restricting their access to resources to their minimal requirements increases the security of the system and all connected systems. Applying these security improvements to containers adds another layer of protection to mitigate attacks against the host and other containers from inside of the container.

4.3 Docker

LXC is a toolkit to work with containers saving users from having to work with low level mechanisms. It creates an interface to access all the features the Linux kernel is providing while reducing the learning curve for users. Without LXC, a user needs to spend a lot of time to read the kernel documentation in order to understand and use the provided features to set up a container by hand. LXC allows to auto-

[1]chsh allows a user to change its login shell. This is a task every user is allowed to do on its own but requires modification of the file /etc/passwd which is accessible for root only. chsh allows safe access to the file and allows a user to change his own login shell, but not the login shells of other users

[2]Common Vulnerabilities and Exposures is a standard to assign public known vulnerabilities a unique number in order to make it easy and to talk about a certain vulnerability and to avoid misunderstandings by confusing different vulnerabilities

mate the management of containers and and enables users to build their own container solution that is customizable and fits even exotic needs. For those who want to simply run processes in an isolated environment without spending too much time figuring out how the toolkit works, this is still impractical. This is were Docker [1] comes in: It is a command line tool that allows to create, manage and deploy containers. It requires very little technical background compared to using LXC for the same task and provides various features that are essential to work with containers in a production environment. Docker started in March 2013 and got a lot of attention since then, resulting in a rapid growth of the project. By combining technologies to isolate processes with different other useful tools, Docker makes it easier to create container images based on other container images. It allows the user to access a public repository on the internet called *Docker Hub* [20] that contains hundreds of ready usable images that can be downloaded and started with a single command. Users have the ability to change and extend those images and share their improvements with others. There is also an API allowing the user to interact with Docker remotely. Docker brings a lot of improvements compared to the usage of LXC, but introduces also new attack vectors on a layer that is not that technical anymore because now users and their behaviour play a bigger part in the security of the complete system. Before talking about that in greater detail, some major improvements over LXC Docker introduced are outlined.

4.3.1 One tool to rule them all

LXC is a powerful tool enabling a wide range of setups and because of that, it is hard to come by when not being deep into the topic and the tools. Docker aims to simplify the workflow. The Docker command line tool is the interface for the user to interact with the Docker daemon. It is a single command with memorizable parameter names allowing the user to access all the features. Depending on the environment, there is little to no configuration required to pull (download) images from the online repository (Docker Hub) and run it on the system. Apart from not being required, it is of course possible to create a configuration file for Docker and specific images in order to ease administration, set specific parameters and to automate the build process of new images.

4.3.2 Filesystem layers generated and distributed independently

One major improvement over classical hypervisor-based virtualization Docker introduced is the usage of filesystem layers. Filesystem layers can be thought of as different parts of a filesystem, i.e. the root file system of a container and filesystems containing application specific files. There may be different filesystem layers, for example one for the base system which may be always required and additional layers containing the files of e.g. a webserver. This means that the webserver that is ready to run can be distributed as an independent filesystem layer. By way of illustration a web service is consisting of a SQL database server like MySQL, a webserver like Apache, a programming language and the framework a web application is built on like python using Django and a mail transfer agent like sendmail is assumed. Of course it is possible to a certain extent to split those tools and run them in different environments (mail, web and database may run on completely different machines), this example assumes that the setup is the developers setup and differs from the deployment in production use. So, after installing Docker, it is possible to fetch the filesystem layers of all of those tools and combine them to one image that runs all the software presented above and may now be used in order to develop the web app. The web app itself may now be the content of a new filesystem layer allowing to be deployed in the same manner later on. If a new version of one of the components is released and the developer wants to update its local layer, only the updated layer has to be fetched. The configuration of the user usually remains in a separate layer and it is not affected by the update. Because there is no need to fetch the data for other layers again, space and bandwidth is saved. If another user wants to use the web app the developer pushed to the Docker Hub, but he prefers PostgreSQL over MySQL and nginx over Apache webserver and the web app is capable of being used with those alternatives to the developer's setup, he may use the respective filesystem layers of the software he wants to use.

4.3.3 Docker Hub

As said, one of the novelties Docker introduced to the virtualization market was the Docker Hub [20]. It is one of the things nobody demanded because it was simply not there and everybody was fine reading documentation and creating virtual machines and container images from scratch over and over again. The Docker Hub is a web service that is fully integrated into the Docker software and allows to fetch images that are ready to run from the Docker Hub. Images created or modified by users may be shared on the Docker Hub, too. The result is that the Docker Hub contains a lot of images of different software in different configurations and every user has the ability to fetch the images and documentation of the images, use and improve them and get in touch with other users by commenting on images and collaborating on them. The Docker Hub can be accessed via a web browser, too. To ease the choice of the *right* images of often used images like Wordpress, MongoDB and node.js, the Docker Hub allows such projects to mark their images as *official* images that come directly from declared project members. The ability for everyone to push their images to the Docker Hub led to a great diversity and a lot of software available as container images. Even more complex or very specific configurations of some programs can be found there. The Hub introduces also some security problems that are covered in Section 5. Users are not tied to the Docker Hub. It is possible to set up private registries that can be used natively the same way, but are e.g. only accessible for authenticated users in a company network or in order to build a public structure that is independent to the already existing structure.

4.4 Managing great numbers of containers

Running containers is cheap regarding resource consumption. Building large-scale services relying on container structures may involve thousands of containers located on different machines all over the world. Today this may still be a corner case, but some companies are already facing management problems because they have too much containers to administrate them by hand. Google introduced Kubernetes [21] in order to ease the administration, group containers and automate working with groups of containers.

5. SECURITY OF CONTAINER-BASED VS. HYPERVISOR-BASED VIRTUALIZATION

Hypervisor-based virtual machines are often used in order to introduce another layer of security by isolating resources exposed to attackers from other resources that need to be protected. “Properly configured containers have a security profile that is slightly more secure than multiple applications on a single server and slightly less secure than KVM virtual machines.“ [10] The better security profile compared to multiple applications next to each other is because of the already mentioned mechanisms making it harder to escape from an isolated environment, but all processes in all containers still run together with other processes under the host kernel. Because KVM is a hypervisor emulating complete hardware and another operating system inside, it is considered even harder to escape this environment resulting in a slightly higher security of hypervisor-based virtualization.

It is possible to combine both technologies and due to the small resource fingerprint and the introduced security layer of container-based virtualization, this is considered a good idea. Nevertheless, container-based virtualization is not proven to be secure and container breakouts already happened in the past [22].

Multiple security issues arise from the spirit of sharing images. Docker has for a long time not had sufficient mechanisms to check the integrity and authenticity of the downloaded images, meaning that the authors of the image are indeed the people who claim they are and that the image has not been modified without being recognized, to the contrary their system exposed attack surface and it may has been possible for attackers to modify images and pass Docker’s verification mechanisms [23]. Docker addressed this issues by integrating a new mechanism called *Content Trust* that allows users to verify the publisher of images [24]. There is nothing like the Docker Hub for hypervisor-based virtualization software, meaning that it is usual to build the virtual machines from scratch. In case of Linux, on the one hand it is common that packages are signed and the authenticity and integrity can be validated, on the other hand each virtual machine contains a lot of software that needs to be taken care off by hand all the time.

One of the biggest problems of Docker is that it is hard to come to grips with the high amount of images containing security vulnerabilities. Origin of this problem is that images on the Docker Hub do not get updated automatically, every update has to be built by hand instead. This is a rather social problem of people not updating their software neither updating the containers they pushed to the Docker Hub in the past. This is a somehow harder problem compared to the same phenomenon of software not being up to date on a virtual machine: There is actually no update available on the Docker Hub, the latest image contains vulnerabilities. On virtual machines when running an operating system that is still taken care of, someone needs to log in and install the security updates, which may be forgotten or simply ignored because of various reasons, but building the fixed container as a user is usually way more effort.

A study [25] found out that “Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities“. The numbers were generated by a tool scanning the images listed on the Docker Hub for known vulnerabilities. More than 60% of official images, which means they come from official developers of the projects contained in the container, contained medium or high priority vulnerabilities. Analyzing the vulnerabilities of all images on the Hub showed that 75% contained vulnerabilities considered medium or high priority issues. Overcoming these problems requires permanent analysis of the containers which means scanning them for security problems regularly and inform their creators and users about problems found. This means that it may be easier to come by the problem of outdated containers because the fact that they are stored centrally allows to statically scan all images in the repository.

Mr. Hayden points out how to build a secure LXC container from scratch that can be used as foundation for further modifications [10].

6. RELATED WORK

One of the use cases for hypervisor and container-based virtualization is improving security by isolating processes that are untrusted or are connected to other systems and expose attack surface, i.e. a webserver on a machine connected to the internet. Of course, there are different ways to improve security that may be applied additionally to virtualization.

The already described Mandatory Access Control (see Section 4.2.3) goes already into the direction of hardening the operating system. This means to add new or improve already present mechanisms in order to make it harder to successfully attack systems. On Linux, the grsecurity [19] project is well-known for their patch set adding and improving a lot of kernel features, for example MAC through their RBAC system, Address Space Layout Randomization (ASLR) and a lot of other features making it harder to attack the kernel.

Another approach of separating applications can be found in the Qubes OS [26] project. They build an operating system based on Linux and the X11 window system on top of XEN and allow to create *AppVMs* that run applications in dedicated virtual machines — somehow similar to the container approach, but with hypervisor-based virtualization instead of mechanisms built inside a standard Linux kernel. According to their homepage, it is even possible to run Microsoft Windows based application virtual machines.

MirageOS [27] is also an operating system setting up on XEN, but deploying applications on MirageOS means deploying a specialized kernel containing the application. Those unikernels [28] contain only what is needed in order to perform their only task. Because there are no features not absolutely required, unikernels are usually really small and performant. MirageOS is a library operating system that may be used in order to create such unikernels. The probably biggest weakness of this approach is that applications need to be written specifically to be used with unikernels.

7. CONCLUSION

Container-based virtualization is a lightweight alternative to hypervisor-based virtualization for those, who do not require or wish to have separate emulated hardware and operating system kernel — a system in a system. There are many scenarios where speed, simplicity and only the need to isolate processes are prevalent and container-based virtualization fulfills these needs. By using mostly features already present in the Linux kernel for several years, container-based virtualization builds on a mature codebase that is already running on the user's machines and kept up to date by the Linux community. Especially Docker introduced new concepts that were not associated and used together with virtualization. Giving users the social aspect of working together, sharing and reuse the work of other users is definitely one of Dockers recipes for success and a completely new idea in the area of virtualization. Because it is relatively easy to run a huge number of containers, there are already tools allowing to manage large groups of containers.

When it comes to security, there is no need for a "vs." in the title of this paper. Generally, it is of course possible to run containers on an already (hypervisor-) virtualized computer or to run your hypervisor in your container. The hypervisor layer is considered a thicker layer of security than the application of the mechanisms described above. Nevertheless, applying these lightweight mechanisms adds additional security at literally no resource cost.

Both approaches and of course their combination is hardware-independent, allows a better resource utilization, improves security and eases management.

8. REFERENCES

[1] *Docker project homepage*, `https://www.docker.com/`, Retrieved: September 13, 2015

[2] *KVM project homepage*, `http://www.linux-kvm.org/page/Main_Page`, Retrieved: September 16, 2015

[3] *qemu project Homepage*, `http://wiki.qemu.org/Main_Page`, Retrieved: September 16, 2015

[4] *XEN project homepage*, `http://www.xenproject.org/`, Retrieved: September 16, 2015

[5] *Microsoft TechNet Hyper-V overview*, `https://technet.microsoft.com/en-us/library/hh831531.aspx`, Retrieved: September 16, 2015

[6] *VMware Homepage*, `http://www.vmware.com/`, Retrieved: September 16, 2015

[7] *VirtualBox project homepage*, `https://www.virtualbox.org/`, Retrieved: September 16, 2015

[8] M. Riondato *FreeBSD Handbook, Chapter 14. Jails*, `https://www.freebsd.org/doc/handbook/jails.html`, Retrieved: September 26, 2015

[9] E. Windisch: *On the Security of containers*, `https://medium.com/@ewindisch/on-the-security-of-containers-2c60ffe25a9e`, Retrieved: August 18, 2015

[10] M. Hayden: *Securing Linux containers*, `https://major.io/2015/08/14/research-paper-securing-linux-containers/`, Retrieved: August 18, 2015

[11] chroot (2) manpage, release 4.02 of Linux man-pages project, `http://man7.org/linux/man-pages/man2/chroot.2.html`, Retrieved: September 16, 2015

[12] *LXC project homepage*, `https://linuxcontainers.org/`, Retrieved: September 13, 2015

[13] namespaces (7) manpage, release 4.02 of Linux man-pages project, `http://man7.org/linux/man-pages/man7/namespaces.7.html`, Retrieved: September 16, 2015

[14] user_namespaces (7) manpage, release 4.02 of Linux man-pages project, `http://man7.org/linux/man-pages/man7/user_namespaces.7.html`, Retrieved: September 16, 2015

[15] Docker Docs: *Docker Security*, `https://docs.docker.com/articles/security/`, Retrieved: August 18, 2015

[16] P. Menage, P. Jackson, C. Lameter: *Linux Kernel Documentation*, `https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt`, Retrieved September 13, 2015

[17] *SELinux project homepage*, `http://selinuxproject.org/page/Main_Page`, Retrieved: September 26, 2015

[18] *AppArmor project homepage*, `http://wiki.apparmor.net/index.php/Main_Page`, Retrieved: September 26, 2015

[19] *grsecurity project homepage*, `https://grsecurity.net/`, Retrieved: September 16, 2015

[20] *Docker Hub*, `https://hub.docker.com/`, Retrieved: September 26, 2015

[21] *Kubernetes project homepage*, `http://kubernetes.io/`, Retrieved: September 16, 2015

[22] J. Turnbull: *Docker container Breakout Proof-of-Concept Exploit*, `https://blog.docker.com/2014/06/docker-container-breakout-proof-of-concept-exploit/`, Retrieved: August 18, 2015

[23] J. Rudenberg: *Docker Image Insecurity*, `https://titanous.com/posts/docker-insecurity`, Retrieved: August 18, 2015

[24] D. Mónica: *Introducing Docker Content Trust*, `https://blog.docker.com/2015/08/content-trust-docker-1-8/`, Retrieved: August 18, 2015

[25] Jayanth Gummaraju, Tarun Desikan and Yoshio Turner: *Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities*, `http://www.banyanops.com/blog/analyzing-docker-hub/`, Retrieved: August 18, 2015

[26] *Cubes OS project homepage*, `https://www.qubes-os.org/`, Retrieved: September 16, 2015

[27] *MirageOS project homepage*, `https://mirage.io/`, Retrieved: September 16, 2015

[28] *XEN Unikernels wiki page*, `http://wiki.xenproject.org/wiki/Unikernels`, Retrieved: September 16, 2015

Analyzing "Global Access to the Internet for All" Projects

Sascha Rushing
Betreuer: Edwin Cordeiro
Seminar Future Internet WS2015/16
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: sascha.rushing@tum.de

ABSTRACT

The world has acknowledged the opportunities and possibilities the Internet offers. It is not only a convenience, but genuinely helps to improve many different and important parts of life, such as education, health and economy. Unfortunately only around one third of the world's population currently has access to it. The open research group *Global Access to the Internet of All* (*GAIA*) is an initiative of the *Internet Research Task Force* (*IRTF*), with the goal that in the near future everyone is provided with affordable Internet access. This paper will introduce the most promising projects related to *GAIA*, including Google's *Project Loon*, *internet.org* by Facebook, Microsoft's approach using TV whitespaces and *Community Networks.* Additionally, proposals concerning how the chair *Network Architectures and Services* of the Technische Universität München could contribute regarding its field of research will be presented.

Keywords

IRTF, GAIA, Internet for all

1. INTRODUCTION

1.1 Motivation

The Internet has grown to be a fundamental part of everyday life. Sending emails to colleagues, video-chatting with family or checking the latest sports results at home on the computer or via smartphones on the go. The Internet has created an entirely new way of communication and information exchange, connecting societies all around the globe.

Unfortunately this privilege is still not accessible to everybody. Even though the number of internet users is increasing, only around 40% of the entire world's population is online. This means that about 4 billion people do not have access to the Internet, with more than 90% for those from developing countries[1].

Figure 1 visualizes this issue. The size of the countries embodies the amount of people with Internet access, whereas the color (ranging from white to dark red) depicts the percentage of the population. As stated earlier and visible in the image, the developing countries are very poorly connected.

The *Global Internet User Survey 2012* conducted by the *Internet Society* interviewed over 10.000 internet users across 20 different counties. Its goal was to provide general information on the behavior and opinions of internet users about various topics regarding the Internet. The majority of the participants agreed, that the internet plays an important role in their everyday life concerning subjects such as knowledge, education, health, economy and many others. Furthermore 83% of the interviewees agreed that internet access should be considered a basic human right.[3] The initiative *Global Goals* by the United Nations released in September 2015 also includes the desire for universal and affordable Internet access in least developed countries by 2020.[4]

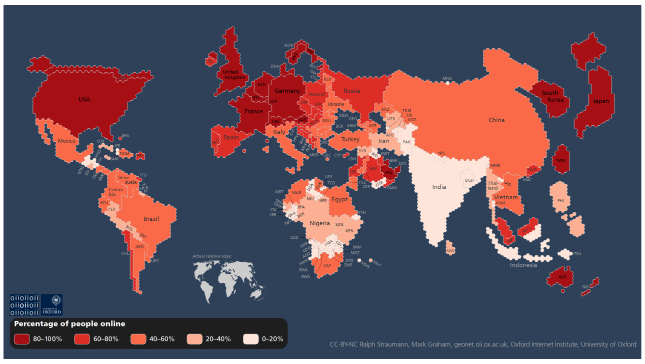

Figure 1: The world online[2]

With this information in mind, the question why so many people do not have Internet access arises. Building broadband internet infrastructure can be very costly, especially if the population is scattered across huge areas. Because of this low cost effectiveness Internet Service Providers (ISP) do not pursue this issue.[5]

Another problem is the pricing and affordability of Internet. In developing countries access costs can be as much as 40 times their national average income. Even in more developed countries this can be an issue, due to unstable financial circumstances people cannot commit to a lengthy broadband contract.[5]

This paper presents a research group and its most promising ideas to try and make Internet access possible for everyone.

1.2 Internet Research Task Force (IRTF)

The *Internet Research Task Force* (*IRTF*) is a research group focusing on internet related topics. It consists of multiple sub groups investigating specific subjects concerning Internet protocols, applications, architecture and technology. In contrast to the *Internet Engineering Task Force* (*IETF*), a parallel organization dedicated to short term subjects of engineering and standard making, the *IRTF* commits to long-term related research. Any committed individual can contribute and participate.[6]

Since the creation of *IRTF* in 1989, 25 research groups have been chartered and concluded their activities. As of September 2015, there are a total of nine active research groups.[7]

1.3 Global Access to the Internet for All (GAIA)

One sub group of *IRTF* is called *Global Access to the Internet for All* (*GAIA*). It was chartered on October 15th 2014, as a result of the *The Internet Society's Global Internet User Survey 2012*[8], which revealed that a majority of the participants classify Internet access as a basic human right[3]. As the name let's one suspect, its research focuses on solutions that will provide Internet access to everyone.

The *Global Access to the Internet for All* (*GAIA*) initiative is officially defined by the following six goals[8]

1. "to create increased visibility and interest among the wider community on the challenges and opportunities in enabling global Internet access, in terms of technology as well as the social and economic drivers for its adoption; "

2. "to create a shared vision among practitioners, researchers, corporations, non-governmental and governmental organizations on the challenges and opportunities; "

3. "to articulate and foster collaboration among them to address the diverse Internet access and architectural challenges (including security, privacy, censorship and energy efficiency); "

4. "to document and share deployment experiences and research results to the wider community through scholarly publications, white papers, presentations, workshops, Informational and Experimental RFCs; "

5. "to document the costs of existing Internet Access, the breakdown of those costs (energy, manpower, licenses, bandwidth, infrastructure, transit, peering), and outline a path to achieve a 10x reduction in Internet Access costs especially in geographies and populations with low penetration. "

6. "to develop a longer term perspective on the impact of *GAIA* research group findings on the standardization efforts at the *IETF*. This could include recommendations to protocol designers and architects. "

2. PROPOSALS

2.1 Google - Project Loon

Project Loon is an initiative by Google. The first pilot tests were launched in June 2013 in New Zealand. The core concept of *Project Loon* is the use balloons in the stratosphere with an altitude of approximately 20 km as wireless connection points. It utilizes the phenomenon that the winds in the stratosphere are stratified, meaning they are layered with each layer having a different wind direction and speed, as shown in Figure 2.[9] Controlled by one large communication network, the balloons can be arranged as to evenly cover a desired area. This is done by altering the balloon's altitude and moving it to a different wind layer, which in turn will carry the balloon to its designated position.[9] The

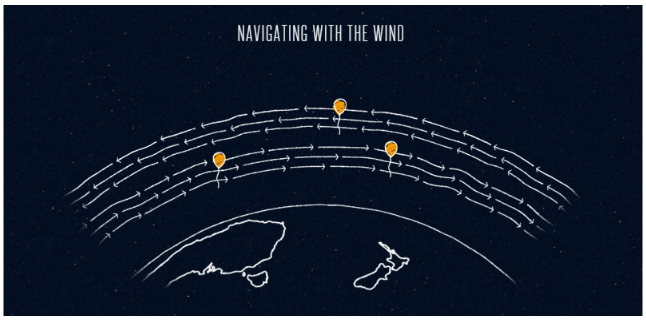

Figure 2: *Project Loon* utilizes the properties of stratified winds in the stratosphere.[9]

wind data for these calculations is provided by the *National Oceanic and Atmospheric Administration*(*NOAA*).[10]

These high altitudes provide unique advantages, but they also create particular engineering challenges. As the balloon rises up in the air, the air pressure and temperature are constantly decreasing. At its optimal altitude air pressure is only 1% of that at sea level, with temperatures around -50°C. Additionally, due to the thinner atmosphere, the balloons are exposed to more direct UV light. To ensure that the balloon does not pop, all these factors have to be taken into consideration in its design.[11] Currently balloons can last for up to 100 days, before deliberately descending down to earth[9]. Besides these technological issues, there are political concerns as well. Many countries may not want to be dependent on a U.S. located company, due to the current status of their relationship with the United States of America.[12]

On the other hand there are very positive things concerning the altitude of the balloons. The winds in the stratosphere are fairly steady with 8 - 32 km/h, which enables controlled position alteration. Furthermore, they are well over commercial airplane altitudes as well as weather phenomenon. This ensures that the balloons can move freely with little danger of collision.[11]

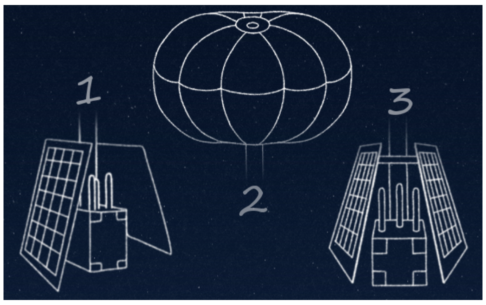

Figure 3: Main components of a balloon 1) solar panels 2) envelope 3) electronics[9]

Each of these solar powered balloons, consisting of three main components as shown in Figure 3, can cover a ground area up to 40 km in diameter. Specialized antennas enable two kinds of connections. Firstly, with other balloons in the mesh network and secondly, with the end user or ground station connected to the internet.[11]

Google has developed antennas which can be installed outside of buildings, for example at home or the work place. In the beginning, these antennas establish a wireless connection via ISM (industrial, scientific and medical) radio bands, using 2.4 and 5.8 GHz. These frequencies are not governmentally owned and unlicensed around the world, which saved Google from frequency negotiations and purchase. The end device in turn could be connected via Wi-Fi to the local antenna. The same method is used to establish a connection between the balloons and a ground station, connected to an internet backbone. Google claimed they were able to reach speeds equivalent to 3G.[11]

As of now, Google has pursued another connection approach using a communication technology called LTE, for which Google partnered with various telecommunication companies. Using the cellular spectrum, every phone or LTE-enabled device will be able to connect directly to the balloon covering the area.[9]

Now that the user can connect to the balloon network and the balloon network to the ground station, the only link missing is the connection between the balloons themselves. The official algorithm used by Google to control and manage their mesh network of balloons is disclosed. It is very probable that their approach builds upon a common method such as the IEEE 802.11s, which is a standard for mesh networking using many mesh points.[11]

2.2 Facebook - internet.org

internet.org is a project initiated by Facebook in partnership with various other companies, non-profit organizations and governments. The core of their idea is to tackle the problem of connectivity using a divide-and-conquer approach, as shown in Figure 4, for different population densities need different solutions. The goal is to create a wireless network, using *Free Space Optics* and radio waves [13], that can be received no matter where. The reason of using wireless over terrestrial technology is the justified by mainly two reasons. Building cell towers and fiber cable infrastructure requires physical construction, which may not be possible due to geographical reasons, or even prohibited due to regulatory approval. Another downside of terrestrial networks is their exposure to insecurities such as war or natural disasters.[14]

Depending on the population density and geographical possibilities, Internet access can be provided by either a radio mast, an unmanned aerial vehicle, a LEO satellite or a GEO satellite. Following simple laws of physics, each method has a different range and signal strength. The closer the device is to earth, the stronger is its signal, yet the smaller is the area it can cover. A relatively forward approach is the use of satellites. Unfortunately the costs of building and shooting them into space are still very high. They will undoubtedly be important to provide connectivity in remote locations, but cannot be used to create a continuous network.[14]

To overcome this issue, Facebook is currently researching a similar approach as Google's *Project Loon*. Instead of using balloons, unmanned solar powered drones called *Aquilla* are to be used, as depicted in Figure 5. They are made of carbon fiber and have a wing span of around 42 meter, which is roughly the size of a Boeing 737. These 400 kilogram heavy, v-shaped crafts will too, like Google's balloons, be stationed at an altitude of approximately 20 km.[13] Facebook is confident that drones are a better solution than balloons as they claim that drones can fly towards their desired position more accurately. Additionally they think that the drones can return to earth more easily for maintenance, creating the opportunity of a cost effective method.[14]

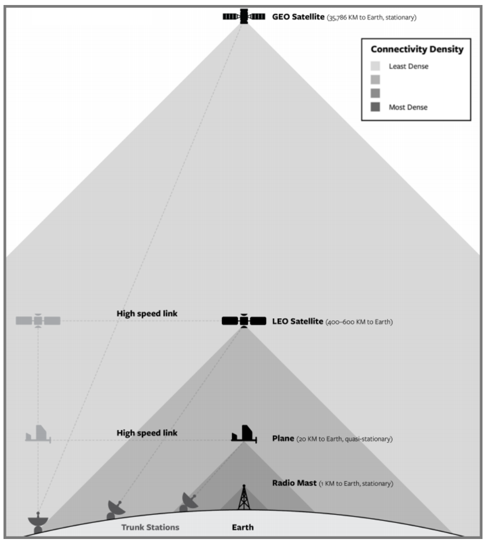
GEO Satellite (35,786 KM to Earth, stationary)
Connectivity Density
Least Dense
Most Dense
High speed link
LEO Satellite (400-600 KM to Earth)
High speed link
Plane (20 KM to Earth, quasi-stationary)
Radio Mast (1 KM to Earth, stationary)
Trunk Stations
Earth

Figure 4: Divide-and-Conquer approach by internet.org[14]

Figure 5: Facebook's solar powered drone, Aquilla.[13]

Solely providing Internet access is not enough if the connection speed is too slow. To overcome this problem, Facebook is working on improving a wireless transmission technique called *Free Space Optics* (*FSO*)[14].

FSO is a procedure of transmitting optical signals through free space of air. To propagate these optical signals through air *FSO* uses light, generated by either LEDs or lasers. The concept of *FSO* transmission can be compared to regular optical transmission using fiber-optic cables, the only difference being the medium they use. In the case of fiber-optic cables light travels through glass, with a speed of approximately 200,000 km/s. *FSO*, using air as the transmission medium, has a speed of around 300,000 km/s, the speed of light. The projected beam size at the receiving end is a lot bigger than at the transmitter. Meaning the shape between the transmitter and receiver is rather equal to a cone than a straight line. This implies that not all the information is arriving at the receiving end. This phenomenon is called *geometric path loss*. To reduce this effect, by making the beam narrower, the transmitter and receiver should have fixed positions.[15]

Unfortunately there are still other drawbacks to the *FSO* method, such as obstruction by physical objects. This is not the case when using cables, since they do not need to be laid out in a straight line between transmitter and receiver. Another issue, taking into account that the transmitters (drones) are in the stratosphere, is fog and clouds. The humid air conditions have a huge impact on light, as the water can absorb, scatter or reflect it, which interrupts the light beam and breaks the connection.[15] For this reason, Facebook is working on a combination of *FSO* and radio waves as a compromise for bad weather conditions.[13]

Public reactions to the *internet.org* project have been very mixed and Facebook has been criticized on what their internet will offer. The main issue is that Facebook reserves the right to only give access to a selected amount of websites and not the entire internet.[16] On May 18th 2015, an open letter concerning these issues has been sent to Mark Zuckerberg. This letter was signed by many international organizations including Germany based *Digitale Gesellschaft* and *Förderverein freie Netzwerke e.V./freifunk.net*[17].

2.3 Community Networks

In contrast to relying on networking structures provided by companies such as Google or Facebook, *Community Networking*, often referenced as *bottom-up networking*, is built and operated by citizens for citizens. In this model of the Future Internet communities build, operate and own open IP-based networks, meaning that anybody who wants to participate can join and therefore increase the size and connectivity of the network. With the help of non-profit organizations managing these networks, it is possible to develop a variety of different services for these networks including local networking, voice connections and Internet access.[18]

Since anybody can participate in these networks, they can be of very large scale. Additionally they are distributed and decentralized as well as very dynamic, for people can join but also abandon the network at any time. The main components of *Community Networks* are nodes which are linked with each other, either wirelessly, using IEEE 802.11a/b/n technology, or via fiber cable. Due to the convenience of installing wireless equipment in contrast to terrestrial construction, wireless technology is mostly used. The nodes are owned and maintained by members of the network, with the only requirement being the acceptance of the peering agreement such as the *Pico Peering Agreement*[19], with the purpose of diminishing participation barriers.[18]

There are already many of such *Community Networks* up and running all around the world. One of the biggest is located in Spain called *Guifi.net*, which consists of over 20,000 nodes.[18] The German initiative *freifunkt.net* is distributed all over Germany, present in currently 209 cities and towns. Each community has its own website including information, such as a live map of all connected nodes and more importantly on how interested persons can join the project. There are also regular meetings to help new members get started, by supporting them getting their equipment configured and installed.[20]

There are still a lot of open research questions concerning *Community Networking*. A research project, called *community-lab* initiated in 2011, is investigating various properties of these networks, which include IEEE 802.11a/b/n connectivity and interference problems, privacy issues and many more.[18]

The idea of *Community Networking* is not only of use in combination with actual Internet access, but as well in secluded locations using an ”Internet in a box” approach. This means that each network contains its own content and services only available in that particular network. It is not connected to the actual Internet. Unlike the way we know it, the content of this network is not stored on servers, but on mobile devices owned by the people of the community, such as smartphones, tablets and laptops. The main requirement of these devices is the availability of either a Wi-Fi or Bluetooth module and a moderate amount of persistent memory.[21]

The way content is distributed in the network can be compared to human verbal interaction. If two people are close to each other they can talk and exchange information. After going separate ways, these two individuals can in turn talk to new people, passing on the information gathered from their previous conversation partner. This way information is spreads arbitrarily.

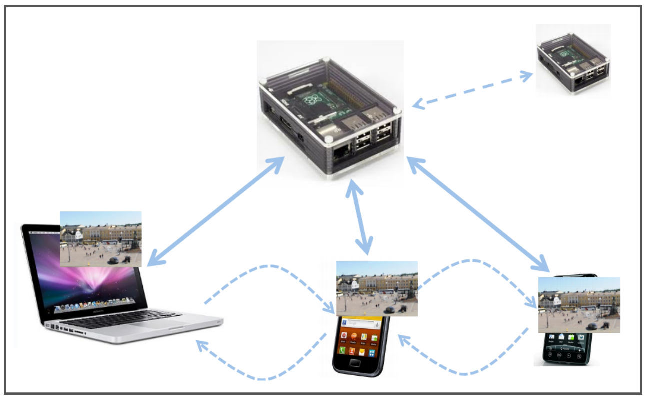

Figure 6: Content is being shared between devices and Liberouters[22]

The mobile devices follow the same principle. If two devices are close enough to each other, they establish a Wi-Fi or Bluetooth connection with each other and exchange content saved on their persistent memory. Since all the information available in the network is stored on these portable devices, storage space is a very limited yet vital resource. To ensure that one's device is not being ”polluted” with uninteresting content, it is possible to *subscribe* to certain content and information categories, filtering what will actually be downloaded and saved to the device's persistent memory.[21]

Considering persistent memory availability, connection distances and battery issues mobile devices alone may not be enough. Public access points, via so called *Liberouters*, distributed in various often visited places can help solve this problem. A *Liberouter* contains a USB flash drive on which information can be stored. Via Wi-Fi connection mobile devices can exchange information with the *Liberouter*, downloading new, subscribed to content as well as uploading information gained from other *Liberouter* or devices as shown in Figure 6. *Liberouters* are made of Raspberry Pis, a WLAN module and a flash drive, creating a very affordable solution for approximately 80 Euro.[21]

Even though this approach is fairly simple, it can have a huge impact on various parts of life in remote locations. For example, in a farming village farmers can subscribe to weather data gathered by a local weather station, which can help increase the harvest. The local news can be distributed like this as well, giving everyone the chance to be up to date on local events. Sectors like healthcare, education and many others would also gain a lot by this distribution method.

2.4 Whitespaces

Due to the complexity of terrestrial Internet access construction, it has become obvious that wireless connectivity in rural and remote areas is a lot more cost effective. This can be done by using the IEEE 802.11 standard, *Free Space Optics* or other radio bands. The problem when using the electromagnetic spectrum is licensing. Most frequencies are strongly regulated by either governments or companies and can therefore not be used.[23]

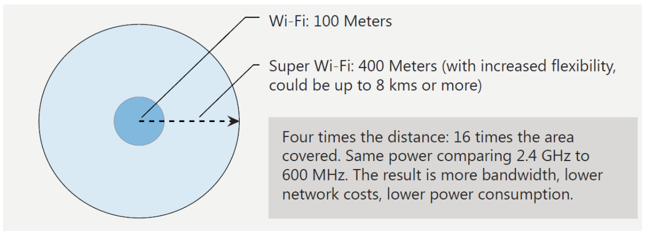
Wi-Fi: 100 Meters
Super Wi-Fi: 400 Meters (with increased flexibility, could be up to 8 kms or more)
Four times the distance: 16 times the area covered. Same power comparing 2.4 GHz to 600 MHz. The result is more bandwidth, lower network costs, lower power consumption.

Figure 7: TV whitespaces can cover a considerable larger area than regular Wi-Fi.[24]

In analog TV transmission, broadcasting channels cannot use frequencies continuously, due to interference issues. This means that there has to be a ”gap” between two channels, leaving certain frequencies unused. These empty channels are referred to as *white spaces*[23]. They can be used, similar to Wi-Fi, to establish an Internet connection. Comparing the Wi-Fi and TV whitespace frequencies, the latter can service an area 16 times the size of the Wi-Fi range. This means that fewer access points are needed to provide internet access for a larger region. Another advantage of TV whitespaces are their penetration, meaning that they are not as easily obstructed by physical objects, such as walls. Due to its properties TV whitespaces are often denoted as *Super Wi-Fi*.[25]

Microsoft has been actively researching the field of TV whitespaces for the last couple of years. They have been involved in many projects with other companies and governments to provide internet connectivity with this technology. Most of which were conducted in Africa, but there have also been projects in Asia and South America.[26]

2.5 Others

Virtual Public Networks. Even though Internet access is available in an area, it does not mean that everyone has the ability to use it. As the *Nottingham Citizens Survey 2011* revealed, affordability is also a very genuine issues in developed countries[27]. Since most households do not occupy their entire bandwidth at all times, the remaining capacity could be shared. This approach is realized in *Virtual Public Networks* (*VPuN*), splitting the connection into best (home user) and less than best effort (guest user) connections.[27][28]

Social Wi-Fi: Hotspot Sharing with Online Friends. Making one’s WiFi connection openly accessible to others can bear risks, such as the sharer being accountable for illegal actions of the guest user. To prevent the probability of harmful guest users *Social WiFi* makes use of the high online social network penetration. Only people who are friends or contacts on social networks (e.g. Facebook, LinkedIn, Google+, etc.) can automatically connect to your network and make use of your Internet access.[29][30]

A4AI - Alliance of Affordable Internet. *A4AI* is a global initiative, composed of private sector, public sector, and civil society organizations dedicated to making Internet access affordable for everyone. Their ultimate goal is to decrease Internet access costs to less than 5% of monthly income.[31] *A4AI* is rather a political than technological initiative, trying to shift policies and regulations in order to create a competitive and innovative broadband market.[32]

Traffic Optimization. Traffic optimization can help save bandwidth usage, airtime in wireless networks and therefore energy. This will be an important part of creating high performance Internet accessibility in remote locations, which only have limited resources at their disposal. A new approach of optimizing network traffic is called *Simplemux*, which is a generic multiplexing protocol.[33][34]

Local Initiatives. There are various programs and projects actively working on getting the world online, such as *AirJaldi* in India[35] and TUCAN3G in Peru[36].

3. TECHNISCHE UNIVERSITÄT MÜNCHEN

3.1 Chair of Network Architectures and Services

The chair for *Network Architectures and Services* at the Technische Universität München focuses on topics in the field of Telematics, the combination of telecommunication and informatics. This includes issues concerning network security, peer-to-peer communication, mobile communication, high speed networks and many more.[37]

The following section will provide ideas on how the Technische Universität München, in particular the chair for *Network Architectures and Services*, could contribute to the *GAIA* initiative with their research.

3.2 Contribution Proposals

3.2.1 Recent Publications

IP Spoofing. IP spoofing describes the action of forging the source IP address in packets. This is done to increase anonymity on the Internet, but also to impersonate other sources.[38] It is therefore a big issue concerning authentication and DoS attacks. The chair has been involved in investigations regarding this problem and how to implement spoofing protection for firewalls.[39]

This could be a very interesting topic concerning the Google balloons and Facebook drones. With so many new Internet users the occurrence of such attacks will very likely increase.

A potential research project could be a spoofing protection implementation for these aerial crafts, for packets coming into the "local network", "local" meaning all end users connected to one particular craft, as well as packets leaving the "local network" into the mesh network and therefore the Internet. Successfully filtering all packets in the craft firewall will ensure that only relevant and honest packets will use the resources and bandwidth of the mesh network. Due to the fact that these aerial crafts are moving and the users connected to it will vary, the filtering criteria will have to change dynamically. The findings of the chair state that the introduced approach can process thousands of rules within a second, which would make this method possible.

Software-based Packet Processing. Another topic the chair has been actively researching is software-based Packet processing. Instead of using rather expensive dedicated hardware, software-based Packet processing runs on commodity and therefor cheaper hardware.[40]

In *Community Networking*, more specifically freifunk München the Wlan routers use a Linux-based operating system called OpenWrt instead of readymade firmware.[41] OpenWrt does not provide too much functionality, but offers the ability of adding packages to obtain new functionalities.[42]

The chair could get involved in creating a package for OpenWrt concerning packet processing. Fast packet processing is especially vital in networks where only a few routers act as a bridge between two bigger clusters. Additionally, mobile devices can also be used as nodes. Here it could also be of interest to the chair to investigate packet processing on mobile devices concerning power efficiency and performance.

Digital Certificates. The *TUM Secure E-Mail* project aims to increase the security and privacy of email communication using OpenPGP and S/MIME digital certificates. A currently open thesis topic attends the issue of how the certificate management could be handled.[43]

In local *Community Networks* (Internet-in-a-box approach) data is being distributed via various Liberouters and devices, granting everyone access to all the information. To ensure privacy in this kind of setup, a certificate approach similar to the *TUM Secure E-Mail* could be used. The fact that these networks do not have a stable connection to some sort of server calls for a creative solution, for example using one specific Liberouter as a certificate manager in the town's city hall.

3.2.2 General Field of Research

Authentication and Anonymity. One of the main research fields of the chair is network security. This includes methods for authentication and security protocols as well as privacy and anonymity on the Internet.[44]

These topics are an essential part of *Community Networking.* As mentioned in section 2.3, anyone can participate in *Community Networks* and increase its size and connectivity. This implies that there are literally a lot of "men in the middle", who could easily intercept and access data sent or received by users in the network. Even though the interference and modification of data is strictly prohibited by the peering agreement[19], it does not mean that ever member of the network will abide by it. The results of the chair's research could help improve those security issues.

Traffic Measurement and Analysis. The chair is involved in research about traffic measurement and analysis. By analyzing the measured data, malicious activities as well as malfunctions in the network can be detected. Additionally the chair contributes to standardization bodies, particularly to the *IETF*, a parallel organization of the *IRTF*.[44]

The outcomes of this study could help to improve the arrangement of the *Project Loon* balloons and Facebook drones. Depending on various factors of the traffic, the crafts have to obtain new positions as to react to the current demand. In order to have low responds times, the traffic has to be analyzed accordingly and in a timely manner.

4. CONCLUSION

It has been officially acknowledged that the Internet should be accessible to everyone, not just one third of the world's population. Internet access was not only rated as a basic human right by the *Global Internet User Survey 2012*, but was also addressed more recently in the *Global Goals* initiative by the United Nations in September 2015. Open research groups like *GAIA* try to tackle this public problem as a community. Many global players are also working on solutions to make this idea reality, for example Google and their *Project Loon*, Facebook with internet.org and Microsoft using TV whitespaces. Even individual citizens are making an effort in solving this issue, as seen in *Community Networks*. The future concerning global access to the Internet for all looks very promising and hopefully someday soon it will not only be an idea anymore, but reality.

5. REFERENCES

[1] International Telecommunication Union. ICT facts and figures, 2014, Access date: 10.09.2015. URL: `http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf`.

[2] University of Oxford. The World Online, Access date: 12.09.2015. URL: `http://geonet.oii.ox.ac.uk/blog/the-world-online/`.

[3] Internet Society. Global Internet User Survey Summary Report, 2012, Access date: 09.09.2015. URL: `http://www.internetsociety.org/internet/global-internet-user-survey-2012`.

[4] United Nations: Global Goals, Access date: 26.09.2015. URL: `http://www.globalgoals.org/global-goals/innovation-and-infrastructure/`.

[5] Global Access to the Internet for All Research Group (GAIA), Access date: 05.09.2015. URL: `https://irtf.org/gaia`.

[6] A. Weinrib, Intel Corporation, and J. Postel. *IRTF Research Group Guidelines and Procedures.*

[7] IRTF Research Groups, Access date: 17.09.2015. URL: `https://irtf.org/groups`.

[8] GAIA Meetings Overview, Access date: 05.09.2015. URL: `https://sites.google.com/site/irtfgaia/home`.

[9] Google. Project Loon, Access date: 17.09.2015. URL: http://www.google.com/loon/.

[10] Steven Levy. How Google Will Use High-Flying Balloons to Deliver Internet to the Hinterlands, Access date: 22.09.2015. URL: http://www.wired.com/2013/06/google_internet_balloons/all/google.com/loon.

[11] Doowon Kim. A Survey of Balloon Networking Applications and Technologies, December 2013, Access date: 21.09.2015. URL: http://www.cse.wustl.edu/~jain/cse570-13/ftp/balloonn.pdf.

[12] Project Loon: Google's biggest obstacle isn't technology. It's politics, Access date: 22.09.2015. URL: https://gigaom.com/2013/06/21/project-loon-googles-biggest-obstacle-isnt-technology-its-politics/.

[13] Tom Simonite. Meet Facebook's Stratospheric Internet Drone, Access date: 22.09.2015. URL: http://www.technologyreview.com/news/539756/meet-facebooks-stratospheric-internet-drone/.

[14] internet.org by Facebook. Connecting the World from the Sky, Access Date: 19.09.2015. URL: https://fbcdn-dragon-a.akamaihd.net/hphotos-ak-ash3/t39.2365-6/851574_611544752265540_1262758947_n.pdf.

[15] Willebrand, Heinz, Ghuman, and Baksheesh. *Free Space Optics: Enabling Optical Connectivity in Today's Networks.* Sams, Indianapolis, IN, USA, 2001, Access date: 22.09.2015. URL: https://books.google.de/books?id=iSk7r67xyboC&printsec=frontcover#v=onepage&q&f=false.

[16] Jeremy Gillula and Jeremy Malcolm. Internet.org Is Not Neutral, Not Secure, and Not the Internet, Access date: 23.09.2015. URL: https://www.eff.org/deeplinks/2015/05/internetorg-not-neutral-not-secure-and-not-internet.

[17] Open Letter to Mark Zuckerberg Regarding Internet.org, Net Neutrality, Privacy and Security, Access date: 23.09.2015. URL: https://openmedia.org/sites/default/files/LetterMarkZuckerbergMay182015-FINAL.pdf.

[18] Bart Braem, Christ Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Vi nas, Aaron L. Kaplan, Axel Neumann, Ivan Vilata i Balaguer, Blaine Tatum, and Malcolm Matson. A Case for Research with and on Community Networks. *SIGCOMM Comput. Commun. Rev.*, 43(3):68–73, July 2013. URL: http://doi.acm.org/10.1145/2500098.2500108.

[19] Pico Peering Agreement v1.0, Access date: 21.09.2015. URL: http://www.picopeer.net/PPA-en.shtml.

[20] German based community networking initiative freifunk.net, Access date: 23.09.2015. URL: http://freifunk.net/.

[21] Jörg Ott and Teemu Kärkkäinen. Liberouter, Access date: 17.09.2015. URL: https://drive.google.com/file/d/0B1P45t6wc9rzNVp6Q29UYUc0N0U/edit?pli=1.

[22] Arseny Kurnikov, Teemu Kärkkäinen, Marcin Nagy, and Jörg Ott. Liberouter Diagram, Access date: 17.09.2015. URL: https://drive.google.com/file/d/0B1P45t6wc9rzOHJEMHJOZjFxZnlkalpmX1N6N0tkNUtPQlpZ/view?pli=1.

[23] Robert Horvitz, Ryszard Struzak, Dariusz Wiecek, Steve Song, Carlos A. Afonso, Timothy X Brown, Jon M. Peha, Cristian, Gomez, Mike Jensen, David Crawford, Linda E. Doyle, Alan Woolhouse, Ermanno Pietrosemoli, Sebastian Büttrich, Marco Zennaro, and Andrés Arcia-Moret. *TV White Spaces - A Pragmatic Approach.* ICTP - The Abdus Salam International Centre for Theoretical Physics, 2013.

[24] TV White Spaces: Super Wi-Fi, Access date: 24.09.2015. URL: http://research.microsoft.com/en-us/projects/spectrum/technology.aspx.

[25] Dynamic Spectrum and TV White Spaces, Access date: 24.09.2015. URL: http://research.microsoft.com/en-us/projects/spectrum/.

[26] Microsoft TV White Space: Pilots and Demonstrations, Access date: 25.09.2015. URL: http://research.microsoft.com/en-us/projects/spectrum/pilots.aspx.

[27] Panagiotis Papadimitriou. Virtual Public Networks, GAIA Presentation, Access date: 27.09.2015. URL: https://drive.google.com/file/d/0B1P45t6wc9rzeDNldWpxeFpuSFk/edit?pli=1.

[28] Arjuna Sathiaseelan, Charalampos Rotsos, Sriram C. S., Dirk Trossen, Panagiotis Papadimitriou, and Jon Crowcroft. Virtual Public Networks, Access date: 27.09.2015. URL: http://www.cl.cam.ac.uk/~as2330/docs/vpun.pdf.

[29] Social WiFi: Hotspot Sharing with Online Friends, GAIA Presentation, Access date: 27.09.2015. URL: https://www.ietf.org/proceedings/93/slides/slides-93-gaia-1.pdf.

[30] Zhen Cao, Jurgen Fitschen, and Panagiotis Papadimitriou. Social WiFi: Hotspot Sharing with Online Friends, Access date: 27.09.2015. URL: http://www.ikt.uni-hannover.de/fileadmin/institut/Publikationen/pimrc_2015.pdf.

[31] A4AI - Alliance of Affordable Internet, Access date: 27.09.2015. URL: http://a4ai.org/visionand-strategy/.

[32] A4AI Strategy, Access date: 27.09.2015. URL: http://a4ai.org/what-we-do/.

[33] Jose Saldana. Simplemux Traffic Optimization, GAIA Presentation, Access date: 27.09.2015. URL: http://www.slideshare.net/josemariasaldana/simplemux-a-generic-multiplexing-protocol.

[34] Jose Saldana. Simplemux Traffic Optimization, Access date: 27.09.2015. URL: http://datatracker.ietf.org/doc/draft-saldana-tsvwg-simplemux/.

[35] AirJaldi - Connecting India, Access date: 27.09.2015. URL: http://main.airjaldi.com/.

[36] TUCAN3G - Connecting Peru, Access date: 27.09.2015. URL: http://www.ict-tucan3g.eu/.

[37] Technische Universität München. The Chair for Network Architectures and Services, Access date: 26.09.2015. URL: http://www.net.in.tum.de/en/homepage/.

[38] Jun Li Toby Ehrenkranz. On the State of IP Spoofing Defense, Access Date: 05.11.2015. URL: `http://ix.cs.uoregon.edu/~lijun/pubs/papers/ehrenkranz09spoofing.pdf`.

[39] Cornelius Diekmann, Lukas Schwaighofer, and Georg Carle. Certifying spoofing-protection of firewalls. In *11th International Conference on Network and Service Management, CNSM*, Barcelona, Spain, nov 2015. URL: `http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/diekmann2015_cnsm.pdf`.

[40] Daniel Raumer, Florian Wohlfart, Dominik Scholz, Paul Emmerich, and Georg Carle. Performance Exploration of Software-based Packet Processing Systems, Access Date: 01.10.2015. URL: `http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/MMBnet15-1.pdf`.

[41] Freifunk München, Access Date: 27.10.2015. URL: `https://ffmuc.net/`.

[42] OpenWrt, Access Date: 01.11.2015. URL: `https://openwrt.org/`.

[43] Design and Implementation of a Management Service for Digital Certificates, Access Date: 15.10.15. URL: `http://www.net.in.tum.de/fileadmin/TUM/theses/pdf/announce-certservice.pdf`.

[44] Research of the Chair for Network Architectures and Services, Access date: 26.09.2015. URL: `http://www.net.in.tum.de/de/forschung/`.

Vergleich von Hardware- und Software-Traffic-Generatoren und ihrem Einsatz in der Praxis

Tobias Weiher
Betreuer: Paul Emmerich, Daniel Raumer
Seminar Future Internet WS2015
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: weiher@in.tum.de

KURZFASSUNG

Das Testen von Hardware für Netzwerkeinsätze wird immer wichtiger. Durch die wachsende Größe und Geschwindigkeit der Netzwerke werden auch ihre Anforderungen anspruchsvoller. Hardware wie Router und Switches müssen wie vorgesehen funktionieren und Last standhalten, um repräsentative Testergebnisse zu erhalten und auch realen Traffic zu unterstützen. Software-basierte Ansätze existieren, die unflexible hardware-orientierte Lösungen ersetzen wollen, jedoch sind jene meist unpräzise und letztere teuer. Hybride Software-Traffic-Generatoren mit Hardwareunterstützung sind eine Möglichkeit, um Präzision und geringere Kosten zu vereinbaren.

Schlüsselworte

Traffic-Generatoren, Software, Hardware, Performanz

1. EINLEITUNG

Das Internet und seine Netzwerke werden immer größer, schneller und ihre Anforderungen auf die im Hintergrund befindlichen Ressourcen anspruchsvoller. Darunter fallen Hardware wie Router und Switches, die wie vorgesehen funktionieren und Last standhalten müssen. Unter Last wird der Aufwand verstanden, den die Hardware standhalten muss, wenn sie sehr viele Pakete mit einer hohen Datenrate, dem Traffic, bearbeiten und weiterleiten muss.
Um dieses Verhalten zu testen, wird auf verschiedene Verfahren von Traffic-Generatoren zurückgegriffen. Häufig werden von größeren Unternehmen teure Hardware-Lösungen verwendet, die für gewisse Anforderungen entwickelt wurden, um gezielt die Performanz zu testen. Jedoch ist diese Herangehensweise nicht sehr flexibel, da die Hardware meist nicht selbst konfigurier- und änderbar ist. Manchmal wird jedoch ein flexibler Weg des Testens erwünscht, besonders im Umgang mit neueren Protokollen oder Netzwerkdesigns. Hierzu gibt es Traffic-Generatoren, die in Software realisiert sind. Allerdings kann bei diesen Alternativen nicht gewährleistet werden, ob ihre Anforderung auch von der zugrundeliegenden Maschine umgesetzt werden kann. Aufgrund dessen gibt es Bemühungen verschiedenster Forschungsgruppen, software-basierte Ansätze zu verbessern, um die teuren hardware-orientierten Lösungen zu ersetzen.
Im folgenden Abschnitt 2 wird zunächst auf die generelle Unterscheidung der Traffic-Generatoren eingegangen. Zudem werden einige Begrifflichkeiten im Zusammenhang zu diesen Generatoren erklärt. Im Abschnitt 3 wird der Unterschied zwischen Hardware- und Software-Generatoren verdeutlicht. Unterabschnitt 3.1 geht auf die Stärken, sowie Schwächen der Hardware-Traffic-Generatoren ein. Unterunterabschnitt 3.1.1 beschreibt mit NetFPGA eine Open Source Alternative zu proprietären Hardwaresystemen. Anschließend werden mit Unterabschnitt 3.2 die software-basierten Traffic-Generatoren betrachtet. Unterunterabschnitt 3.2.1 erwähnt die Vorteile gegenüber einfachen Software-Generatoren und beschreibt deren Funktion. Im Unterabschnitt 3.3 wird Bezug zu einigen Statistiken genommen. Zunächst werden die Traffic-Generatoren in Hinsicht auf ihrer Paketraten in Unterunterabschnitt 3.3.1 verglichen. Danach werden in Unterunterabschnitt 3.3.2 einige Strategien zur Umsetzung der Sendeintervalle zwischen Paketen, beziehungsweise den *inter-departure times* präsentiert. Der letzte Teil mit Unterabschnitt 3.4 zeigt einige Netzwerk-Tools, die in der Praxis eingesetzt werden.

2. ARTEN VON TRAFFIC-GENERATOREN

Traffic-Generatoren sind Werkzeuge, um Netzwerke hinsichtlich ihrer Performanz und Stabilität zu überprüfen, um womögliche Probleme frühzeitig in der Entwicklung zu entdecken. Hierbei werden Daten in das Netzwerk eingespeist, um angeschlossene Geräte wie Router oder Switches zu testen. Dabei wird unterschieden, welche Daten in welchem Maße zugeführt werden.
Diverse Traffic-Generatoren lassen sich grob in folgende Kategorien unterteilen.[1]

- Traffic-Generatoren auf Anwendungsebene
- Traffic-Generatoren auf Datenflussebene
- Traffic-Generatoren auf Paketebene
- Traffic-Generatoren im geschlossenen Kreis und mit mehreren Ebenen

Die Generatoren, die auf Anwendungsebene funktionieren, erlauben, das Verhalten von Programmen und ihrem Einfluss auf das Netzwerk zu emulieren. Ein Programm, das beispielsweise mehrere Spielserver emuliert und die Netzwerkdaten von vielen Clients und der Server selbst austauscht und versendet, würde zu diesen Traffic-Generatoren zählen. Ein Netzwerk-Tool, welches lediglich Pakete über eine gewisse Dauer für einen Datenfluss erzeugt, wird den Traffic-Generatoren auf Datenflussebene zugeordnet. Hierbei wird versucht, möglich realistischen Traffic zu generieren, beispielsweise die Anzahl und Größe der Pakete, die bei einem Webshop-Einkauf entstehen würden.

Der Großteil der Traffic-Generatoren arbeitet allerdings auf Paketebene. Hierbei werden verschiedenste, vom Benutzer definierte Parameter umgesetzt, um beispielsweise die Paketgröße, die Verzögerung von zu sendenden Pakete (auch *inter-packet delay* genannt, also die mindestens zu wartende Zeit, bevor ein Paket gesendet werden darf) oder die Paketrate, häufig gemessen in Anzahl Pakete minimaler Größe innerhalb einer Sekunde, zu konfigurieren. Auch kann das Sendeintervall bestimmt werden, welche die Zeit zwischen zwei zu sendenden Pakete so abstimmt, dass das zweite Paket erst versendet wird, nachdem eine gewisse Zeit nach dem vorherigen Versenden abgelaufen ist. Diesen Begriff versteht man unter anderem auch als *inter-departure time*, welche zu jedem Paket unterschiedlich ausfallen kann, da sie in der Regel pro Paket definiert wird. Jedoch kann dieses Sendeintervall nicht geringer sein als die minimale Verzögerung zwischen zwei Paketen, das *inter-packet delay*. Letztere wird häufig durch die physikalische Grenzleistung der Übertragung definiert, der *line rate*, im Gegensatz zur *bit rate*, die die Bitrate der Übertragung angibt.[2] Die maximale Bitrate kann nicht größer sein als die durch die physikalische *line rate* mögliche Leitung, sie kann jedoch niedriger ausfallen. Die maximale Bitrate von 10GbE, also 10 Gigabit Ethernet, ist 10.000.000.000 bps, damit 10 Milliarden Bits pro Sekunde. Um dies in einer maximalen Paketrate anzugeben, müssen die 10 Milliarden bps durch die Präambel eines jeden Paketes, der geringst möglichen Framelänge und dem sogenannten *inter-frame gap*, dem Warteabstand zwischen zwei Paketen, geteilt werden. Dadurch erhält man eine maximale Frame Rate von

$$\frac{\text{10 Gigabits/s}}{\text{Präambel} + \text{Framelänge} + \text{inter-frame gap in bits}} =$$

$$\frac{\text{10.000.000.000 bits/s}}{8 * 8 \text{ bits} + 64 * 8 \text{ bits} + 12 * 8 \text{ bits}} = \frac{\text{10.000.000.000 b}}{\text{672 b} * \text{s}} =$$

$14.880.952,38$ Pakete pro Sekunde.[3]

Die letzte Kategorie umfasst Traffic-Generatoren, die die Interaktion über mehrere Schichten des Netzwerk-Protokoll-Stacks hinweg miteinbeziehen. Hier werden Benutzer-, sowie Session- und Anwendungs-Informationen extrahiert, um Netzwerkcharakteristika zu ermitteln, die ein realistischeres Traffic-Abbild erstellen sollen.
Damit immer überprüft werden kann, dass der erwünschte Traffic durch die Generatoren erzeugt wird, sollten all diese Generatoren mit sowohl Generierungs-, als auch Überwachungs-Funktionalität ausgestattet sein. Das bedeutet, dass Traffic-Generatoren nicht bloß Daten erzeugen können sollten, sondern auch eine Bestätigung über die erreichte Rate ausgeben können oder, falls diese nicht erreicht wird, dementsprechend tatsächliche Werte präsentieren. Mithilfe gewisser Zeitstempelverfahren kann dadurch sogar eine gezieltere Paketrate realisiert werden, welche in folgenden Abschnitten näher beschrieben werden.

3. VOR- UND NACHTEILE DER GENERATOREN

Traffic-Generatoren erfüllen den Zweck, Testgeräte oder Netzwerke auf Fehler und Performanz zu überprüfen. Dabei kann auf verschiedenste Realisierungen von Netzwerk-Tools zurückgegriffen werden. Es gibt sowohl hardware-basierte Ansätze wie auch Software-Traffic-Generatoren. Um zu differenzieren, welche Tools für welchen Gebrauch geeigneter erscheinen, werden generelle Lösungen betrachtet und verglichen.

3.1 Hardware-basiert

Traffic-Generatoren auf Hardware-Basis sind geschlossene, zu meist nicht näher konfigurierbare Systeme, die vordefinierte Daten nach Angaben des Herstellers erzeugen. Beispiele hierfür sind Netzwerktester von Ixia[4] oder Spirent[5], die Traffic-Generatoren für 1/10/40/100 GbE beziehungsweise sogar höhere Gigabit Ethernet Werte anbieten. Diese werben damit, Daten mit *line rate* erzeugen zu können, damit also die höchste Paketrate des jeweiligen GbE-Wertes erreichen, teilweise sogar darüber hinaus, indem sie die minimale Framelänge der kleinsten Ethernetpakete nochmals kürzen, beispielsweise auf 58 Byte. Da die Hersteller kompletten Zugriff auf ihre Hardware haben, erzielen sie sehr genaue Zeitwerte, die auch dank der präzisen Zeitstempel im Nanosekundenbereich besonders niedrig ausfallen. Die Anzahl der versendeten Bytes, unabhängig der Größe der Pakete, der Hardware-Traffic-Generatoren kann dabei nach ihren Angaben durchwegs die maximale Rate erreichen.[6]
Allerdings können nur diejenigen Protokolle verwendet werden, die zum Zeitpunkt der Erstellung der Hardware beziehungsweise ihrem Erwerb unterstützt wurden. Neuartige Protokolle werden durch die Hardware nicht erkannt. Immerhin kann bei den meisten Hardware-Generatoren der Test-Traffic durch ein Script benutzerdefiniert eingestellt werden, sodass ein Testgerät nach eigenen Bedürfnissen beansprucht und überprüft werden kann.[6]
Jedoch sind diese präzisen, hardware-basierten Traffic-Generatoren teuer und somit für Forschungseinrichtungen nicht sinnvoll erwerbbar, um beispielsweise die Daten und Spezifikationen der Hardware-Generatoren genauer zu untersuchen. Ein gebrauchter Traffic-Generator dieser Art kostet bereits 12.000$[7], neuere Geräte können damit gut das Doppelte des Preises wert sein. Das Hinzufügen weiterer unterstützter Protokolle und Funktionen für diese Module erhöht den Preis zusätzlich.[15]

3.1.1 NetFPGA

Um den hohen Preisen der Hardwarehersteller entgegenzuwirken und mehr Flexibilität in das Testen durch Traffic-Generatoren zu bringen, wurde das Projekt NetFPGA[8] ins Leben gerufen. Das Projekt begann im Jahr 2001 an der Stanford University, um Studenten das Netzwerkverhalten und die benötigte Hardware zu veranschaulichen. Da das Projekt jedoch größer wurde und die Hardware immer ausgereifter, stieg die Nachfrage nach einem offizielleren Gerät speziell für Forscher an.[9]
NetFPGA bietet Software und Hardware an, um neue Designs, Simulationen und das Testen auf einer Open Source Netzwerkplattform zu vereinfachen. Der Projektname leitet sich von *network* und dem *FPGA*, einem *field-programmable gate array*, ab. Mit diesen *FPGAs* sind Architekturen gemeint, die nach der Produktion noch vom Benutzer umprogrammierbar sind. Die erste NetFPGA-Plattform, erstellt für Forschungs- und Lehrgruppen, war das NetFPGA-1G im Jahr 2007, eine kostengünstige Platine für 1 Gigabit Ethernet.[10]
Der Nachfolger von 2010 war das NetFPGA-10G (Abbildung 1), welche mit einer 40 Gigabit pro Sekunde fähigen PCIe Schnittstelle ausgestattet ist und 4 10 GbE Verbindun-

gen mitbringt. Alle Karten des NetFPGA-Projektes sind mit *FPGAs* der Firma Xilinx[11] verbaut. Die aktuellste Karte ist das NetFPGA SUME, welche Anwendungen mit Anforderungen von 40 und 100 Gigabit pro Sekunde unterstützen können soll.[12]
Dieser Hardware-Traffic-Generator ist bereits für etwas unter 10.000$ zu erwerben, für den akademischen Gebrauch sogar unter 5.000$.[13]

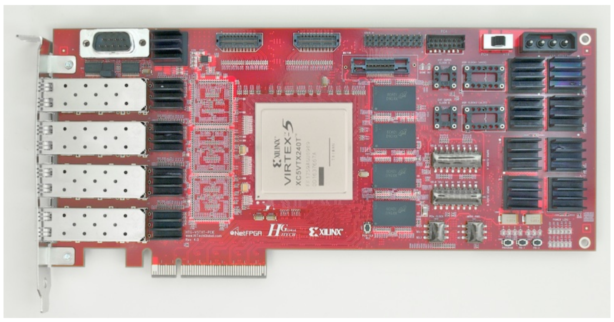

Abbildung 1: NetFPGA-10G[14]

3.2 Software-basiert

Im Gegensatz zu den hardware-basierten Traffic-Generatoren, wird für Software-Generatoren lediglich das Programm und eine ausführbare Maschine benötigt, die das jeweilige Programm umsetzen soll. Jedoch ergeben die Software-Lösungen auch unerwünschte Probleme, die nicht immer sofort zu erkennen sind. Falls nicht anders angegeben, so bezieht sich der Abschnitt der software-basierten Traffic-Generatoren auf die Quelle [1].
Bereits das Erzielen einer gewissen Paketrate mit minimalen Paketgrößen ist stark abhängig vom Softwaredesign des jeweiligen Generators. Da es auch andere Programme auf der auszuführenden Maschine gibt, die für den Ablauf des Systems benötigt werden, muss der Traffic-Generator die Ressourcen mit den anderen Anwendungen teilen. Dadurch sinkt die Performanz, die durch die CPU erreicht werden kann, um eine hohe Paketrate zu erzielen. Somit ist es möglich, dass eine angezielte Paketrate von 150.000 Paketen pro Sekunde schon bei unter 80.000 Paketen seinen Grenzwert erreicht. Dies erfolgte in einem Test mit Pentium IV Maschinen und einem Linux 2.6.15 Betriebssystem, auf denen nicht benötigte Anwendungen ausgeschaltet wurden.
Auch wenn die größtmöglichen Pakete verwendet werden, gibt es Performanzprobleme. So kann die angestrebte Bitrate von 1 Gigabit pro Sekunde bereits den maximal möglichen Durchsatz von 500 Megabit pro Sekunde erschöpfen. Dieser Wert entspricht in der kleinst möglichen Paketgröße einer Paketrate von unter 45.000 Paketen pro Sekunde, also weitaus weniger als die im vorherigen Fall erreichten 80.000 Pakete. Dies liegt an der Struktur der Software-Generatoren, die im Gegensatz zu Hardware-Traffic-Generatoren keine dedizierten Speicherbereiche besitzen, aus denen effizient gelesen oder geschrieben werden kann. Somit muss die Software diese Speicherbereiche teuer umkopieren, bis diese ausgesendet werden können, worunter die Präzision dieser Generatoren leidet.
Doch nicht nur die Konkurrenz um Ressourcen des Systems vermindern die Präzision der Software-Generatoren, sondern auch die Genauigkeit der Zeitauflösung. Dies ist besonders für die *inter-departure time* zwischen Paketen bemerkbar. Falls eine gewisse Paketrate eingestellt wird, für die das Sendeintervall zwischen zwei Paketen kleiner ausfällt als die Genauigkeit der Zeitmessung es erfassen kann, so senden einige Software-Traffic-Generatoren die gesamte Menge der Pakete in einem Schwall beziehungsweise *Burst* aus.
Ein anderer Fall, der bei einem solchen Test womöglich nicht beachtet wird, ist eine längere Wartedauer, bevor ein erneutes Paket versendet wird. Wenn die *inter-departure time* bei einigen Software-Generatoren einen Wert um die 4 Millisekunden annimmt, so könnte der Scheduler des Betriebssystems unter dieser geringen Last den Prozess des Software-Generators von der CPU entkoppeln. In diesem Augenblick wird der Prozess allerdings wieder benötigt, sodass dieser teure Kontextwechsel des Prozesses viel Zeit beansprucht und damit die Paketrate durch die Verzögerung reduziert wird. Aufgrund dessen ist es für Software-Traffic-Generatoren notwendig, diese Phänomene zu bedenken und beispielsweise eine Polling-Funktion einzuführen, die innerhalb kleinerer Zeiteinheiten durchwegs auf neue Informationen hin überprüft und damit die Ressourcen auf der CPU behält.
Obwohl die software-basierten Traffic-Generatoren flexibel sein können und durch Aktualisierungen der Versionen neuartige Protokolle unterstützt werden können, so haben diese jedoch, abhängig ihrer Implementierung, besondere Schwächen in ihrer Präzision und sind damit zumeist inakkurat.

3.2.1 Hardware-unterstützt

Werden Software-Traffic-Generatoren jedoch auf Basis gewisser Netzwerkkarten entwickelt, so gewinnen sie eine höhere Präzision durch unterstützte Funktionen der Hardware. Hierzu zählen genauere Zeitstempel durch Taktraten der Chips auf den Netzwerkkarten oder auch eine Zeitkorrektur im Laufe der Ausführung, die sich besonders bei reinen Software-Generatoren von der tatsächlichen Zeit durch Fehler, einem *Clock Drift*, entfernt.
Ein bekanntes Beispiel in der Open Source Community ist OSNT, ein *Open Source Network Tester*, welcher auf das NetFPGA-10G (Abbildung 1) mit 4 10GbE Schnittstellen basiert. OSNT ermöglicht durch eine Virtualisierung der zugrundeliegenden Hardware, dem *NetV* (Abbildung 2), eine Aufteilung des NetFPGA-10G für sowohl Traffic-Generierung, als auch Traffic-Überwachung.[15]

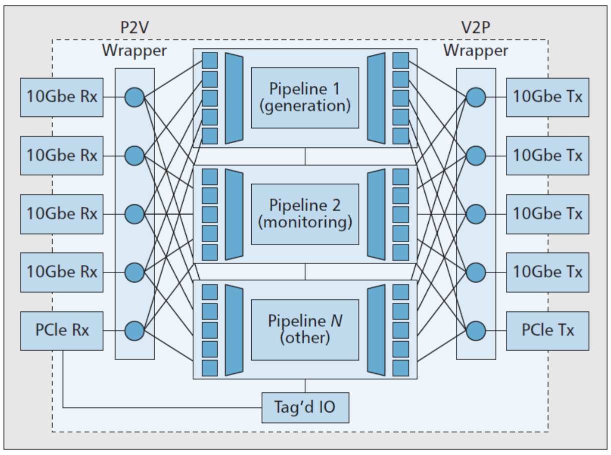

Abbildung 2: NetV-Virtualisierung[15]

Diese Software besteht aus einem *OSNT Traffic-Generator*,

welcher Pakete an allen 4 10GbE Schnittstellen erstellen und empfangen kann. Indem ausgehende Pakete mit Zeitstempel versehen werden, können Informationen bezüglich Verzögerungen und Verlust der Pakete ermittelt werden. Damit kann ein Netzwerkelement wie ein Router oder Switch getestet werden oder ein kleineres Netzwerk, da Ein- und Ausgang der Pakete an derselben NetFPGA-Karte verbunden werden können.
Ebenso ermöglicht OSNT durch seinen *OSNT Traffic-Monitor*, ankommende Pakete an den 10GbE Schnittstellen aufzuzeichnen und an die überliegende Software weiterzuleiten, um diese zu analysieren und weiterzuverwenden. Diese Datenrate kann sehr hoch ausfallen, wenn alle oder die meisten der eingehenden Pakete aufgezeichnet werden sollen. Da die Aufzeichnung einen Flaschenhals für die PCIe Bandbreite verursacht, werden Pakete optional gehashed, also effizient tabellarisiert, und in der Hardware bereits gekürzt. Damit soll eine geringe Verlustrate mit einer hohen Auflösung sowie Präzision des Zeitstempels unterstützt werden.
Durch das *hybride OSNT System* kann auf derselben NetFPGA-Karte mit ihrer *NetV*-Virtualisierungstechnik und den Zeitstempeln eine Charakterisierung des Netzwerkes beziehungsweise des Messgerätes auf *full line-rate* pro Datenfluss erzielt werden. Mit dem Zeitstempel werden ein- und ausgehende Pakete markiert, sodass OSNT im Netzwerk diese Pakete analysieren kann. Da die Zeit jeweils im Paket vermerkt wurde, lässt sich eine Aussage bezüglich des Netzwerkes treffen.
Außerdem bemüht sich die Software als ein *skalierbares OSNT System*, eine große Anzahl von mehreren Traffic-Generatoren und -Überwachungen zu koordinieren. Durch eine Zeitsynchronisierung aller OSNT-Geräte soll ein größeres Netzwerk zuverlässig auf beispielsweise Latenz, Paketschwankungen oder -verlust überprüft werden können.[15]
Eine weitere Software, die die Hardware eines Systems, beispielsweise eines NetFPGAs, ausnutzen kann, ist das OFLOPS, ein offenes Framework für OpenFlow Switch Evaluationen.[16] OpenFlow ist ein System, mit welchem Router durch eine Konfiguration lediglich Regeln ausführen, die ihm durch einen Kontrollrechner beigebracht wurden. Somit wird seine Routing-Logik ersetzt und aus dem System ausgegliedert. Dieses Verfahren kommt in software-definierten Netzwerken, auch *Software Defined Networks*, kurz SDN, vor, indem die Logik, auch *Control Plane*, ausgelagert wird und der Router als einfache Weiterleitungsplattform, auch *Data/Forwarding Plane*, fungiert. Wenn eine höhere Zeitpräzision verlangt wird, so kann OFLOPS die NetFPGA-Hardware nutzen, andernfalls kann diese auch mit Performanzeinschränkungen auf üblicher Hardware mit der Softwarelösung *OpenVSwitch* genutzt werden.[16]
Ebenso ist eine Verbindung der Hardware NetFPGA-10G mit der Virtualisierung durch OSNT und der Host-Software OFLOPS für einen OpenFlow-Einsatz denkbar. Mit diesem sogenannten OFLOPS-Turbo-Host können mehrere Switches in unterschiedlichen Netzwerkstrukturen verbunden werden, um gewisse Aspekte der Netzwerkarchitektur mit hoher Präzision zu messen. Dies ist sowohl mit der *Data Plane* der Switches oder Router möglich, als auch mit der ausgegliederten *Control Plane*.[17]
Ein weiterer hardware-unterstützter Software-Traffic-Generator ist BRUNO, ein Traffic-Generator für einen Netzwerkprozessor, im Speziellen dem Intel IXP2400. BRUNO, welcher für *BRUte on Network prOcessor* steht, basiert auf einer modifizierten BRUTE (Browny and RobUst Traffic Engine) Version. BRUTE wurde dafür designt, um Sendezeiten von Paketen abhängig von gegebenen Traffic-Modellen zu ermitteln. Das System schreibt diese Informationen in einen Speicherbereich, der mit der Paket-verarbeitenden Einheit des Netzwerkprozessors geteilt wird. Der Netzwerkprozessor nutzt diese Daten für die Erstellung der Pakete und sendet diese mit einer geeigneten Zeitschranke, der *inter-departure time*, los. BRUNO soll für eine 1GbE Verbindung die *line rate* ausnutzen können und versendet mit einer hohen Präzision mit einer Paketrate von bis zu 1.488.000 Pakete pro Sekunde.[18]
Um höhere Flexibilität bei gewohnter Präzision durch Hardware zu erhalten, verwendet MoonGen einen etwas anderen Ansatz. MoonGen ist ein Hochgeschwindigkeits-Paket-Generator, welcher 2014 an der Technischen Universität München entwickelt wurde. Durch MoonGen wird die gesamte Paketerstellungslogik zu Benutzer-kontrollierbaren Lua-Scripts übertragen, um ein Höchstmaß an Flexibilität zu erzielen. Dies wird durch die Einbindung von LuaJIT ermöglicht, einem *Just-In-Time*-Compiler für Lua, womit direkt mit Bibliotheken der Sprache C und dessen Structs gearbeitet werden kann. Dadurch können Pakete effizient mit MoonGen erzeugt werden. Außerdem wird das Paketverarbeitungs-Framework DPDK, das *Data Plane Development Kit*, verwendet, um schnell und präzise den Input und Output der Pakete auf unterstützter Hardware durchzuführen. Dadurch wird eine Rate von 14,88 Megapakete pro Sekunde ermöglicht, einer *line rate* von 10GbE mit minimaler Paketgröße (vergleiche Abschnitt 2). Derzeit werden durch DPDK und MoonGen die Hardware-Funktionen auf den Intel-Karten 82599, X540 und 82580 unterstützt. Es können zwar andere Netzwerkkarten verwendet werden, die von DPDK unterstützt werden, jedoch kann dabei die Zeitstempelfunktion und die Datenraten-Kontrolle der Hardware nicht genutzt werden. Durch die Lua-Scripts ist es außerdem möglich, jedes einzelne Paket, das versendet werden soll, zu manipulieren. Dies kann ohne große Performanz-Einbußen stattfinden, da durch Lua eine kleinere Schleife durchlaufen wird, wenn nicht alle Bereiche des Paketes verändert werden. Durch MoonGen wird nur dann mit Performanz bezahlt, wenn es durch aufwändigere Aktionen benötigt wird, beispielsweise dem Ver- oder Entschlüsseln einzelner Felder in eigen definierten Protokollen eines Paketes. Wenn hingegen nur die IP-Adresse vieler vordefinierter Pakete im Puffer geändert werden soll, bevor diese versendet werden, so kann MoonGen mehrere Pakete erstellen und nur das nötige IP-Feld bearbeiten. Andere Paket-Generatoren, die ebenfalls auf DPDK aufbauen wie *Pktgen-DPDK*, können dabei nur langsamer Pakete versenden als MoonGen, da jene in einer komplexeren Ausführung alle möglichen Konfigurationen durchlaufen müssen, obwohl nur ein Feld im Paket bearbeitet werden müsste.[19]

3.3 Eigenschaften im Vergleich

Um nun einige der hier erwähnten Traffic-Generatoren zu vergleichen, wird zunächst nur auf eine der vielen denkbaren Charakteristika eingegangen. Allerdings ist dieser Vergleich nicht einfach, da es keine direkt übereinstimmenden Metriken gibt, auf die Traffic-Generatoren sich reduzieren lassen.[20] Aufgrund dessen werden die durch die Forschungsgruppen spezifizierten Statistiken bei Übereinstimmung aufgezählt und bewertet.

3.3.1 *Paketraten*

Unter dem Begriff der Paketrate wird der maximale Durchsatz an Paketen mit minimaler Größe verstanden. Dies entspricht in der Regel einer Paketgröße von 64 Bytes. Zusätzlich mit dem Abstand zweier Frames und der Ankündigung eines weiteren Paketes ergibt dies eine Länge von 84 Byte (Abschnitt 2).
Hierbei erzielen die hardware-basierten Traffic-Generatoren wie beispielsweise von Ixia[4] oder Spirent[5] den meisten Durchsatz bis hin zur maximal möglichen *line rate* der jeweiligen GbE-Anbindung. Jedoch sind diese auch die teuersten Ableger in ihrem Bereich.
Die einfachen Software-Traffic-Generatoren erzielen weitaus weniger Pakete in der Sekunde, ob nun die kleinstmögliche Länge oder nicht, da diese Ressourcen mit dem Betriebssystem teilen müssen und sonst weniger optimiert auf die jeweilige Hardware der Rechner sind.
Im Gegenteil dazu sind die neueren Software-Generatoren basierend auf spezieller Hardware, wie zum Beispiel aus dem NetFPGA-Projekt (Unterunterabschnitt 3.1.1) oder unterstützten DPDK-Modellen, wesentlich effizienter und erreichen zudem Paketraten bis zur *line rate*.[19]
Jedoch sind die meisten dieser hardware-unterstützten Software-Traffic-Generatoren für 10GbE-Schnittstellen entwickelt, während die Hardware-Generatoren schon mit 100GbE oder teilweise sogar 400GbE für Datenzentren werben.[21] Einige Testversuche zu Software-Traffic-Generatoren für höhere Raten scheinen aber zuversichtliche Ergebnisse zu liefern. So erreicht MoonGen mit einem Test von 120 Gigabit pro Sekunde bis zu 178,5 Megapakete pro Sekunde.[19] Ein Maximalwert bei dieser Anbindung wäre eine Paketrate von

$$\frac{120 \text{ Gigabits/s}}{84 * 8 \text{ bits}} = \frac{120.000.000.000 \text{ bits/s}}{672 \text{ bits}} =$$

$178.571.428,57$ Pakete pro Sekunde.

3.3.2 *Sendeintervall*

Das Sendeintervall zwischen zwei zu sendenden Paketen wird unterschiedlich gebraucht. Zum einen gibt es den *inter-packet delay*, welcher den minimalen Zeitabstand zwischen allen Paketen entspricht. Hier darf nur dann ein Paket versendet werden, wenn diese Zeit erreicht wird. Damit kann eine gewisse Paketrate eingestellt werden, die durch den Testverlauf eingehalten wird. Dieser Zeitabstand kann nicht kleiner ausfallen als es die maximale Paketrate durch die *line rate*, der physikalischen Grenze, erlaubt, durch welche sie häufig definiert wird. Dieses *inter-packet delay* ist abhängig von der Fähigkeit des Generators, der angebundenen Verbindung und dem dahinterliegenden Netzwerk.
Zum anderen gibt es den Begriff der *inter-departure time*, welche für jedes Paket unterschiedlich konfiguriert werden kann. Diese entspricht der Zeit, nachdem ein Paket versendet wurde, bis ein nächstes Paket gesendet werden darf. Diese kann nicht kleiner ausfallen als das *inter-packet delay*.
Einfache Software-Traffic-Generatoren können eine *inter-departure time* von wenigen Milli- und Mikrosekunden schon nicht mehr garantieren, wodurch bei einem sehr kleinen Sendeintervall die Software zu einem *burst* neigt, also Pakete in einem Schwall so schnell wie möglich versendet werden. Da die Genauigkeit in der Zeitstempelauflösung fehlt, können gezielte Tests für Netzwerkgeräte kaum ausgeführt werden, wenn die Ankunfts- und Sendezeiten der Pakete nicht genau gemessen werden kann.[1] Hardware-unterstützte Software-Generatoren haben diesbezüglich eine wesentlich bessere Chance, die Zeiten bei der Paketmarkierung einzuhalten. Das NetFPGA-10G besitzt eine 6,25 Nanosekunde Zeitstempel-Auflösung mit einer Koordination der Zeitfehler, verursacht durch *Clock Drifts*, durch GPS.[15] Somit dürften die Zeitgrenzen zwischen Paketen bei unterschiedlichen *inter-departure time*-Verteilungen mit hoher Genauigkeit erkannt werden. Lediglich die Hardware-Traffic-Generatoren werben mit einem höchst akkuraten Zeitstempelmodul für Zeitsynchronisationen direkt über Kabel oder ebenfalls GPS von einer Auflösung von 2,5 Nanosekunden für Generierung und Analyse.[6]

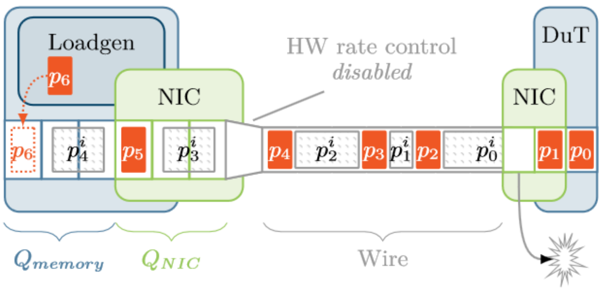

Abbildung 3: MoonGen's Paketverzögerung[19]

MoonGen bietet für gewisse *inter-departure times* eine weitere Strategie an. Anstatt eine Zeit lang zu warten, bis Pakete gesendet werden dürfen, was Fehler in der Zeitabweichung verursachen kann, werden die Lücken zwischen regulären Paketen mit ungültigen Paketen gefüllt. Durch die Veränderung der Größe der ungültigen Pakete kann genau bestimmt werden, wann Pakete versendet werden. Zudem können beliebig komplexe Traffic-Muster dadurch erzeugt werden. Jedoch muss das Testgerät im Netzwerk dazu in der Lage sein, ungültige Pakete zu erkennen und diese, ohne die Paketverarbeitung negativ zu beeinflussen, zu verwerfen. Diese Pakete besitzen eine falsche CRC-Summe im Ethernetframe und, wenn nötig, eine ungültige Länge für kleinere Lücken. Obwohl in der Theorie willkürliche Lücken zwischen Paketen möglich sein sollten, füllen einige Netzwerkkarten Pakete kleiner als 76 Bytes, inklusive Präambel und inter-frame gap (siehe Abschnitt 2), auf, also 8 Byte weniger als das übliche Minimum von 84 Bytes (Unterunterabschnitt 3.3.1). Daher können Lücken von 1 bis 75 Bytes, also 0,8 bis 60 Nanosekunden bei 10 Gigabit Ethernet, nicht erzeugt werden. Die generierte Paketrate bei dieser Alternative muss der jeweiligen *line rate* entsprechen, es müssen also durchwegs Pakete generiert und von der Hardware versendet werden. Die transparenten Pakete p^i in Abbildung 3 sind die ungültigen Pakete, die von der Testhardware frühzeitig verworfen werden.[19]

3.4 Werkzeuge (Tools) in Verwendung

Traffic-Generatoren, die vermehrt in der Praxis zum Einsatz kommen, werden in diesem Abschnitt beschrieben. Sie entsprechen weit verbreiteten Software-Generatoren, die unterschiedliche Einsatzbereiche haben, da sie verschiedenen Arten von Traffic-Generatoren (Abschnitt 2) entsprechen. Viele von ihnen existieren bereits seit einiger Zeit, einige sind etwas moderner.

Mausezahn ist ein Traffic-Generator auf Paketebene, mit welchem beinahe jedes denkbare Paket versendet werden kann. Es wird hauptsächlich für *Voice-over-IP*-Programme, aber auch für Sicherheitstests gegen bestimmte Angriffe, wie einem *Denial of Service*, kurz *DoS*, durch einen *TCP SYN-Flood*, verwendet. Hierbei können Verzögerungen von Paketen zwischen zwei Endgeräten präzise überprüft werden, Tests und Angriffe auf Firewalls, Einbruchserkennungssystemen für Netzwerke oder Netzwerke selbst ausgeübt werden, sowie das Netzwerkverhalten unter besonderen Bedingungen, wie Lasttests oder defekte Pakete, getestet werden. Mausezahn existiert seit 2007, wurde aber im Juli 2013 in das *netsniff-ng toolkit* übernommen.[23] *Netsniff-ng* ist ein Netzwerkkit, welches für Linux frei zur Verfügung steht.[24] Dadurch, dass bei Paketempfang und -versand keine Daten umkopiert werden müssen aufgrund von *Zero-Copy-Buffers*, also gemeinsam verwendetem Speicher (zwischen Kernel- und Userspace), erzielt netsniff-ng eine bessere Performanz als einfache Software-Traffic-Generatoren.[22]
Iperf ist ein weiterer Software-Traffic-Generator auf Paketebene.[25] Iperf ist ein Tool, um die maximale TCP-Bandbreite zu messen. Ebenso können diverse Parameter und UDP-Charakteristika eingestellt werden. Iperf gibt die gemessene Bandbreite, den Delay und Paketverlust wieder, nachdem so viel Traffic wie möglich über eine gewisse Zeit versendet wurde. Da durch Iperf jedoch nicht speziell definiert werden kann, welche Pakete wie versendet werden, werden diese Bandbreiten-Mess-Werkzeuge nicht direkt als Traffic-Generatoren bezeichnet.[1] 2003 wurde Iperf der Version 1.7.0 veröffentlich. Seit 2014 existiert Iperf3, eine Reimplementierung, um eine kleinere, einfachere Basis zu bieten, mit der nun Funktionalitäten verwendbar sind, die es in der ersten Iperf-Version nicht gab[26], wie TCP Retransmit-Informationen, welche nun standardmäßig aktiviert sind, und eine genauere Ausgabe bezüglich CPU-Verbrauch.[27]
Ein weiterer Traffic-Generator, der seit 2014 durch Hardware mit DPDK unterstützt werden kann, ist *Ostinato*, eine Paket-basierte Generator- und Analyse-Software mit benutzerfreundlicher Oberfläche. Mit dieser Open Source Software können Pakete von verschiedenen Datenströmen mit unterschiedlichen Protokollen und Raten versendet werden. Ostinato wurde im April 2010 veröffentlicht.[28]
Zudem gibt es Paket-Traffic-Generatoren, die schon 2007 versuchten, Hardware zur Unterstützung einzusetzen. *D-ITG*, ein *Distributed Internet Traffic Generator*, welcher zu damaliger Zeit neuartige Protokolle unterstützte und auf dem Intel IXP-425 Netzwerkprozessor aufbaut. D-ITG erzeugt IPv4 und IPv6 Traffic, welche durch die *inter-departure time* und Paketgrößen in einem stochastischen Prozess mit mehreren zur Verfügung stehenden Mustern erzeugt wird. Damit kann die Verzögerung, die Hin- und Rückzeit eines Paketes, also die *Round Trip Time*, Paketverlust, Schwankungen und Durchsatz gemessen werden. Zudem ist es möglich, jedes Experiment mit einem *Random Seed* zu versehen, welche die Reproduzierbarkeit eines zufälligen Traffic-Musters garantiert. Ebenso können viele andere Bereiche des Paketes verändert werden.[29]
Ein Traffic-Generator auf Datenflussebene ist Harpoon. Es verwendet eine Menge von Verteilungseigenschaften, die automatisch aus Netzwerkverläufen erfasst werden können, um Datenflüsse mit denselben statistischen Angaben, wie aus den Internetverläufen, zu erstellen. Ebenso kann Hintergrund-Traffic generiert werden, welcher zum Testen von Anwendungen, Protokollen oder Router und Switches genutzt wird. Im Juni 2004 wurde Harpoon für die Öffentlichkeit zur Verfügung gestellt.[30]
Ein Software-Traffic-Generator, welcher auf mehreren Protokoll-Ebenen arbeitet, ist *Swing*. Dieses Programm zeichnet akkurat die Paketinteraktionen von Anwendungen auf und extrahiert Verteilungen von Nutzer-, Anwendungs- und Netzwerkverhalten. Damit kann anschließend sofort Paket-Traffic generiert werden, welcher auf den zugrundeliegenden Modellen der Netzwerkemulations-Umgebung mit gewöhnlichen Protokollstacks arbeitet. Durch die extrahierten Verhaltensweisen kann detailliert *Burst* im Traffic über zahlreiche Zeiträume reproduziert werden. Zudem kann der Benutzer in Swing Annahmen über den Traffic ändern, wie beispielsweise Paketgrößen oder zusätzliche Anwendungen, um einen neuen Traffic zu erzeugen.[31]

4. ZUSAMMENFASSUNG

In dieser Arbeit wurden die Stärken der hardware-basierten und -unterstützten Traffic-Generatoren bezüglich Performanz und Präzision gegenüber den einfachen, aber günstigen Software-Generatoren aufgezeigt. Besonders Generatoren mit hoher Zeitstempelauflösung sind für Delay-Messungen wichtig, da sie akkurat ausgehende, sowie eintreffende Pakete bezeichnen können und damit eine detaillierte und genaue Zeitinformation erhalten. Ebenso sind Generatoren wichtig, die ungewollte Burst-Situationen beim Umsetzen einer gewissen Paketrate durch Einhalten der *inter-departure times* vermeiden. Letztendlich sollte der Traffic-Generator auch im Stande sein zu überprüfen, welcher Traffic durch ihn verursacht und ob das angestrebte Ziel erreicht wurde, also Daten auch überwachen, beziehungsweise aufzeichnen können. Auch der Trend der immer schnelleren Anbindung von 100 GbE oder 400 Gigabit Ethernet sollte von moderneren Software-Traffic-Generatoren präzise auf *line rate* erreicht werden, um eine Alternative zu geschlossenen und meist unflexiblen Hardware-Traffic-Systemen zu bieten.

5. LITERATUR

[1] Alessio Botta, Alberto Dainotti, Antonio Pescapé: *Do You Trust Your Software-Based Traffic Generator?*, in IEEE Communications Magazine, Seite 158-165, September 2010

[2] *Line rate and bit rate*, `http://blog.ipspace.net/2009/03/line-rate-and-bit-rate.html`, zuletzt besucht am 21.09.2015

[3] Spirent: *HOW TO TEST 10 GIGABIT ETHERNET PERFORMANCE*, Rev. B 03/12, März 2012

[4] ImpairNet, `http://www.ixiacom.com/products/impairnet`, zuletzt besucht am 22.09.2015

[5] Spirent TestCenter, `http://www.spirent.com/Ethernet_Testing/Software/TestCenter?docfilter={FF146BE3-9E89-476F-AB1E-3C176C0AB3A4}#Overview`, zuletzt besucht am 22.09.2015

[6] SPIRENT: *MX 100 GIGABIT TEST MODULES*, `http://www.spirent.com/~/media/Datasheets/Broadband/PAB/SpirentTestCenter/Spirent_mX_100G_CFP2_Datasheet.pdf`, Seite 3, Rev. A 05/13, Mai 2013

[7] Spirent TestCenter Module,

http://www.smartechconsulting.com/NG-100G-F2-HyperMetrics-40G-100G-Ethernet, zuletzt besucht am 22.09.2015

[8] NetFPGA, www.netfpga.org, zuletzt besucht am 22.09.2015

[9] Greg Watson, Nick McKeown, Martin Casado: *NetFPGA: A Tool for Network Research and Education*, in 2nd workshop on Architectural Research using FPGA Platforms (WARFP), 2006.

[10] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, Jianying Luo: *NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing*, in IEEE International Conference on Microelectronic Systems Education (MSE'07), Seite 160-161, Juni 2007

[11] All Programmable FPGAs and 3D ICs, http://www.xilinx.com/products/silicon-devices/fpga.html, zuletzt besucht am 22.09.2015

[12] Noa Zilberman, Yury Audzevich, G. Adam Covington, Andrew W. Moore: 'NetFPGA SUME: Toward 100 Gbps as Research Commodity,' IEEE Micro, vol.34, no.5, Seite 32-41, September-October 2014

[13] NETFPGA-SUME, http://digilentinc.com/Products/Detail.cfm?NavPath=2,1301,1311&Prod=NETFPGA-10G-SUME, zuletzt besucht am 22.09.2015

[14] NetFPGA 10G Board, https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Board, zuletzt besucht 23.09.2015

[15] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, Adam Covington, Marc Bruyere, Nick McKeown, Nick Feamster, Bob Felderman, Michaela Blott, Andrew W. Moore, Philippe Owezarski: *OSNT: Open Source Network Tester*, in IEEE Network, Seite 6-12, September/Oktober 2014

[16] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, Andrew W. Moore: *OFLOPS: An Open Framework for OpenFlow Switch Evaluation*, in Volume 7192 der Serie Lecture Notes in Computer Science, Springer-Verlag GmbH Berlin Heidelberg, Seite 85-95, März 2012

[17] Charalampos Rotsos, Gianni Antichi, Marc Bruyère, Philippe Owezarski, Andrew Moore: *OFLOPS-Turbo: Testing the Next-Generation OpenFlow switch*, European Workshop on Software Defined Networks (EWSDN), Budapest, Hungary, 2 Seiten, September 2014

[18] Gianni Antichi, Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci: *BRUNO: A High Performance Traffic Generator for Network Processor*, in SPECTS 2008, Seite 526-533, Edinburgh, UK, Juni 2008

[19] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, Georg Carle: *MoonGen: A Scriptable High-Speed Packet Generator*, http://arxiv.org/abs/1410.3322, 13 Oktober 2014, zuletzt besucht am 23.09.2015

[20] Sándor Molnár, Péter Megyesi, Géza Szabó: *How to Validate Traffic Generators?*, in IEEE International Conference on Communications 2013: IEEE ICC'13 - 1st IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios, Seite 1340-1344, Juni 2013

[21] 400GbE Load Modules, http://www.ixiacom.com/products/400gbe-load-modules, zuletzt besucht am 24.09.2015

[22] Mausezahn, http://www.perihel.at/sec/mz/, zuletzt besucht am 24.09.2015

[23] netsniff-ng v0.5.8-rc1, https://github.com/netsniff-ng/netsniff-ng/releases/tag/v0.5.8-rc1, zuletzt besucht am 24.09.2015

[24] netsniff-ng toolkit, http://netsniff-ng.org/, zuletzt besucht am 24.09.2015

[25] Iperf - The TCP/UDP Bandwidth Measurement Tool, http://web.archive.org/web/20081012013349/http://dast.nlanr.net/projects/Iperf/, zuletzt besucht am 24.09.2015

[26] iperf / iperf3, https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf-and-iperf3/, zuletzt besucht am 07.11.2015

[27] iperf3, http://software.es.net/iperf/, zuletzt besucht am 24.09.2015

[28] Ostinato, http://ostinato.org/, zuletzt besucht am 24.09.2015

[29] Alessio Botta, Alberto Dainotti, Antonio Pescapè: *Multi-protocol and Multi-platform Traffic Generation and Measurement*, in INFOCOM 2007 DEMO Session, 2007

[30] Harpoon: A Flow-level Traffic Generator, http://cs.colgate.edu/~jsommers/harpoon/, zuletzt besucht am 24.09.2015

[31] The Swing Traffic Generator, http://cseweb.ucsd.edu/~kvishwanath/Swing/, zuletzt besucht am 24.09.2015

The Interface to the Routing System

Elias Hazboun
Supervisor: Edwin Cordeiro
Innovative Internet Technologies and Mobile Communications WS2015
Chair for Network Architectures and Services
Department of Informatics, Technical University of Munich
Email: hazboun@in.tum.de

ABSTRACT

Management of today's ever growing communication networks is posing a challenge for network operators around the globe; networks are expected to react quickly to change, and automation is steadily becoming a necessity to cope with complexity. This paper presents one of the protocols proposed by the Internet Engineering Task Force (IETF) to cope with such a challenge, called the Interface to the Routing System (I2RS). The protocol focuses on routing operations in networks by offering operators standardized programmatic interfaces to the routing information stored in their devices. I2RS is based on NETCONF protocol while its data models are based on YANG modeling language. We argue that I2RS is a viable solution for the challenge at hand and has a good business case for operators since it leverages existing routing protocols and does not require a potential overhaul of network architectures.

Keywords

I2RS, Software defined networking, network management, NETCONF, YANG, routing system.

1. INTRODUCTION

Traffic carried by networks whether mobile, global or intra-data-center is rapidly increasing, one estimate by Cisco predicts that the annual global IP traffic will surpass 2.0 zettabytes (10^{21} bytes) by 2019 which translates to a three-fold increase from 2014 [1]. Moreover, businesses are moving fast and are expecting their underlying networks to be able to follow suit [2]. Therefore, network operators today are facing more and more pressure to be flexible to match business change and to be efficient to cope with the ever-growing traffic.

Software defined networking (SDN) is one possible approach to offer this flexibility. SDN development and large scale implementations [3] show that SDN is a viable solution for the current network challenges [4]. In traditional networks, the control plane and the data plane are distributed across complex devices in the network. On the other hand, SDN is an approach to networking in which control of the network or what is called the control plane is partially or fully centralized in a logical entity and decoupled from the actual forwarding devices which carry the traffic [5].

Nevertheless, both SDNs and traditional networks are facing a challenge to support complex automation and quick policy-based interaction with network operations. Today, these aspects are supported usually by proprietary and limited protocols such as Cisco ACI [6] which do not facilitate use in multi-vendor networks [7].

This gap between the needs of network operators and the standard solutions available motivated vendors and carriers to investigate a new protocol, which culminated in early 2013 with the Internet Engineering Task Force (IETF) creating a working group to find such a solution particularly aimed towards routing operations in networks, known as the Interface to the Routing System (I2RS) [8]. Figure 1 illustrates where the new protocol I2RS interacts with a network compared to the currently used SDN protocol Openflow [9] (Not shown in the figure is the possible interaction between I2RS and the data plane for retrieving data flow information).

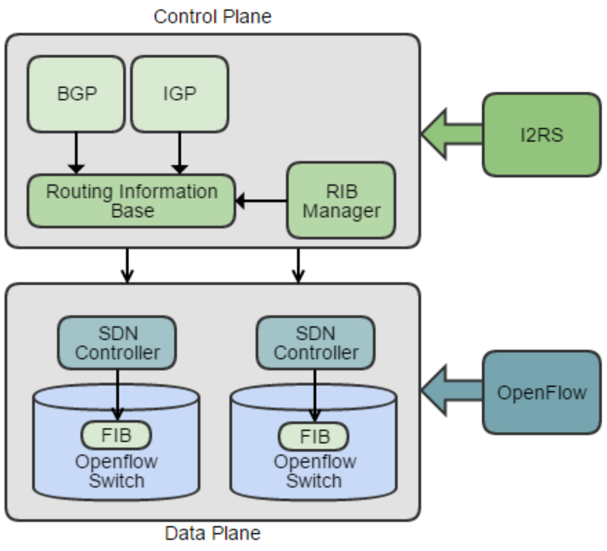

Figure 1: I2RS and Openflow interactions

In this paper, we present and shortly analyze the idea of I2RS and the protocol being developed at IETF. The rest of the paper is structured as follows: section 2 describes the driving force behind I2RS. Section 3 focuses on the architecture of I2RS and the different elements in it, while section 4 describes its data model. Section 5 mentions typical use-cases for the protocol as set out by the IETF and section 6 offers brief analysis of I2RS. Finally, we conclude the paper in section 7.

2. THE NEED FOR A NEW PROTOCOL

We must first understand the current state of affairs of accessing information on routing devices to recognize the key requirements expected from I2RS and its goals.

Today, a typical network operator manages a large number of routing devices purchased from multiple vendors and running a variety of routing protocols. These devices maintain information that is integral to their function. For example, a Routing Information Base (RIB) contains routes to network destinations learned from routing protocols such as BGP or OSPF. Additional information might include counters and statistics in addition to packet forwarding rules pertaining to the forwarding plane of these devices.

Access to the previous information is an essential part for successful network management. To this end, operators usually use a combination of the following three methods.

- Command line interface (CLI): vendor specific commands are entered by a network engineer in a Unix like shell to edit or learn device states.
- Simple Network Management Protocol (SNMP): a historic and popular protocol that is most often used to retrieve (and to lesser extent modify) state information about devices. It is based on simple scalar data types to represent network configuration data [10].
- Network Configuration Protocol (NETCONF): a modern protocol that uses remote procedure calls (RPC) to configure devices; it is focused on being simple and extensible therefore it uses XML for data encoding [11].

Vendors have realized the shortcomings of using legacy techniques in network-oriented applications and automation, for example CLI scripting is not easy to use when it comes to data retrieval and filtering, while SNMP users are faced with lack of both configuration semantics and expressing power of models. Consequently, proprietary protocols and interfaces to access information have emerged through the years to solve specific use-cases. These protocols however are limited and hard to integrate into multi-vendor networks, nevertheless they are used today. Their deployment demonstrates the dire need for a standardized method of accessing and manipulating routing information [7].

This is where I2RS steps in. It is in a sense, not a replacement of the three methods of management mentioned above, but a new method that focuses on creating a standardized data-model driven interface for the secure and dynamic access of information in routing devices. Thus, I2RS can be defined as "a programmatic asynchronous interface for transferring state into and out of the internet routing system" [12].

I2RS is still a work-in-progress and some of its aspects have not been agreed upon completely yet. Nevertheless, from the previous discussion, we can already derive four key aspects that drive I2RS development as proposed by the IETF. First, it has to offer a fast, programmatic, asynchronous access for atomic operations. Second, it has to offer access to information not easily accessible by existing configuration protocols. Third, it has to offer the ability to retrieve data as well as to subscribe to event notifications from devices. Fourth, it has to be data-model driven to facilitate extensibility and standardization [12].

3. ARCHITECTURE

The architecture of I2RS is designed in a way that facilitates control and enables network applications to be built on top of networks. In this section we explain I2RS architecture, its properties and interactions.

3.1 Architectural Properties

I2RS protocol has to possess some defining properties to achieve its goals. The IETF has laid these out in [12] and can be summarized as such:

Perhaps the most logical property is *simplicity*, since most network operators agree that complex protocols are often error prone and are difficult to operate and implement correctly. However, maintaining simplicity of a protocol that accesses a wide variety of data types stored on different types of devices can be a challenge of its own.

Additionally, for a protocol that is reliant on modeling current and future data, I2RS must have *easy extensibility* or otherwise its limitations will quickly catch up to its potential and hinder its adoption. That is why the IETF is being careful with designing models that are extensible in a straightforward fashion.

Moreover, *model-driven programmatic interfaces* are required since current routing data models and the mechanisms to access them on devices are not standardized and are governed by vendor specific rules. This hinders interoperability and the ease of application implementation. Hence, I2RS must utilize a standard model-driven protocol which facilitates data access through automated applications.

Finally, *performance and scalability* are expected of I2RS because routing systems are anticipated to have a high number of operations and changes per second while requiring low latency execution to ensure smooth management. One method for achieving scalability is through filterable access to data, while another is through multi-channel communication between I2RS clients and agents as discussed in subsection 3.2.

3.2 Major Architectural Components

In terms of the architecture of I2RS, we identify five major components [12] and discuss their interactions below.

Network application: a network oriented piece of software with the goal of accessing or manipulating network states. It achieves its goal by communicating with I2RS clients.

I2RS client: an entity that implements the I2RS protocol and communicates with I2RS agents to access their services, in order to gain access to network information or to modify it. A client could be either an external I2RS library or simply the piece of code that is I2RS aware inside an application.

I2RS service: a set of functions for information access and modification coupled with their usage policy. They are defined by a given data-model such as MPLS or BGP services which provide access to MPLS and BGP related states respectively.

I2RS agent: an entity that actually interacts with the routing element sub-systems to obtain and modify their states; it provides this functionality as an I2RS service for requesting clients. How agents access this information is out of the scope of I2RS. However, how the data is presented, i.e. its models, is an integral part of I2RS.

Routing element: in the scope of I2RS, a routing element is any device that implements some functionality pertaining to routing; it could be a traditional router implementing BGP or the logical control plane of an SDN controller.

No matter what the implementation of a specific routing element is, an I2RS agent's behavior to clients must not be affected. For example, in the case of a physically distributed routing element, an agent should still support the access of data from the whole element. Additionally, multiple agents can reside on a single routing element; in that case, they must be responsible for the service of separate sets of information to ensure simplicity.

3.3 Roles, Identities, and Priorities

Access to such delicate information on routing elements must have some kind of access control and tracking; to this end I2RS defines roles that can be assigned to clients, where a role of an I2RS client defines its read/write and subscription rights (called scopes). Furthermore, I2RS assigns identities to clients which agents use for authentication [13]. Finally, a client may have a priority attribute that can aid in case of state access conflicts which we will discuss at the end of subsection 3.4.

3.4 I2RS Interactions

Figure 2 illustrates the components mentioned in subsection 3.2 and their associated connections. An application can communicate with multiple local and remote I2RS clients and conversely an I2RS client can respond to multiple applications. Clients on the other hand are served by agents running on routing elements, where if necessary, a single client can request from multiple agents on the same or different routing elements. Finally, an agent is able to serve more than one client at a time.

I2RS is mainly focused on the interactions between agents and clients, where agents provide - through their advertised services - the ability for clients to access and modify data on the routing elements in addition to the ability to subscribe to events affecting these elements.

Agents can access the data of three components on the routing elements: The *routing subsystem* which includes the RIB manager and routing protocols like BGP, the *dynamic system state* which includes the various counters and data flow information, and finally the *static system state* which includes data pertaining to the system itself such as interface information. One point to note is that I2RS agents are not directly accessing the Forwarding Information Base

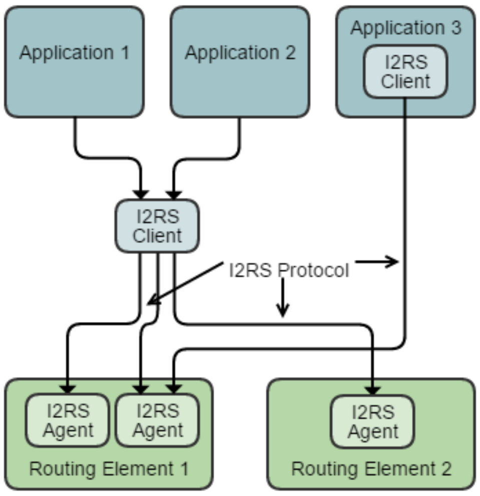

Figure 2: Interactions between I2RS clients and agents

(FIB), but rely on devices themselves to translate I2RS RIB changes into corresponding FIB entries on their own.

Routing elements and their agents keep track of I2RS state changes in the I2RS data store which comprises records of changes and their requesting clients as well as the active subscriptions by clients. A data store is only stored in memory and will be lost upon reboot, hence it is called an ephemeral state [14]. Therefore, any implementation must be careful to specifically assign I2RS changes as ephemeral, so that even when the running configuration is copied to a persistent memory for example, these changes will not. Using the data-store, an agent must have the ability to roll back changes it has applied since the client interaction when necessary; this roll back usually reverts to the state specified by other means of device configuration.

As agents modify these states on routing elements, conflicts may arise between requested modifications and the configuration provided by means other than I2RS such as SNMP or CLI. In such cases, a clear operator policy must be in place to enforce a pre-determined behavior. Whatever the policy may be, the agent must notify the requesting client if their request was blocked.

Additional conflicts can arise from two clients trying to modify the same state on an I2RS agent; however, this case should be avoided as I2RS considers it an error. Nevertheless, if such a case does occur, an agent must try to resolve this issue by first considering the priorities of clients or by a first come first served basis. No matter the conflict scenario that might occur in agent implementation, the design property of simplicity dictates that the behavior must be predictable and the error reportable to affected clients.

3.5 Notable Protocol Considerations

The IETF working group opted for a model of I2RS where existing protocols are utilized and leveraged as much as possible. To that end, it was agreed that the I2RS shall be based on NETCONF [11] and its close sibling with a REST interface RESTCONF [15]. Nevertheless, the I2RS working group has also recognized that some modifications must be made to these protocols before being used in I2RS, especially in regards to security aspects. As for the transport layer protocol, the working group has left it as an operator chosen aspect, as long as features of integrity, authentication and ease of deployment are fulfilled.

In regards to the atomicity of operations performed by agents, no guarantees beyond a single client message shall be made. An agent will not try to have roll back mechanisms for multiple client messages. This limitation is in a sense a feature that pledges simplicity and predictable behavior which are part of the goals of I2RS. In the case of error handling within a single message, a client can signal to the agent one of three kinds of behavior; perform all operations or none at all in case of error, perform all operations up to the point of error or attempt to perform all operations regardless of errors. No matter the behavior set by the client, an agent must reply with explicit success or failure messages to requesting clients.

3.6 Security Considerations

I2RS exposes sensitive interfaces to the routing system, the access of which requires security guarantees in order to be adopted by operators. In this section, we present major I2RS security aspects.

First of all, I2RS assumes that a routing element can trust an I2RS agent residing on it, since it is a part of it, whether as a part of the operating system image or as a signed add-on to it. However, such trust cannot be established between a client and an agent, therefore some kind of mutual authentication must take place before operations can be permitted. A client must be able to verify the identity of the agent it is trying to communicate with and its attached routing element. Additionally, an agent must be able to authenticate a client based on its supplied identity in the communication channel [13].

Using this identity, an agent can link a client to a role that has a set of specific scopes (read/write and subscription rights); these in turn will be used for authorization purposes before performing any operations on behalf of the client.

Moreover, data confidentiality is vital for sending sensitive configuration and statistics over the network; operators are reluctant to transport network information in plain text. However, I2RS acknowledges that there should be support for cases where confidentiality is not explicitly needed by an operator. As a result, communication channels may support unsecure transport layer protocols. As for data integrity, the proposed I2RS protocol should be able to protect against data modification in transit as well as replay attacks where messages are merely repeated by attackers [13].

4. INFORMATIONAL MODEL

At the heart of I2RS are the standard data models and their semantics, which serve as interfaces for information in routing elements. These models should be extensible by design and try to - if possible - use preexisting data models. The I2RS working group has chosen YANG [16] to be the language used for modeling. YANG was also developed at the IETF and is an acronym for ”Yet Another Next Generation”, it is a modern data modeling language that uses trees to model configuration and state data. Although not based on XML, it has an associated language called YIN which maps its data models into XML. Yang features a small set of prebuilt standard models but supports extensibility through derivation for vendor created data types [17].

I2RS aims primarily to use YANG to model the states and elements in the RIB, which is where a standardized programmatic interface is currently critically missing. However, I2RS agents shall support services for other states in a routing element ranging from Border Gateway Protocol (BGP) and Inter-gateway Protocol (IGP) to Quality of service (QoS) and policy mechanisms.

To be vendor agnostic, the I2RS information model must be compatible with all the various routing elements in the network and their different implementations. To that end, I2RS borrows from object oriented paradigm to define object classes, types and inheritance. For example, a parent class can have all the common attributes found in all routing devices, while its subclasses add vendor or use-case specific attributes. Moreover, an agent may not support all classes or attributes in a service and shall communicate to requesting clients through a capability model what it currently offers.

Finally, objects in routing elements seldom exist alone and are rarely unaffected by other objects, thus the I2RS information model must express these relationships as clear and robust as possible.

Figure 3 shows an example of partial modeling of a route object in the RIB, a route is matched based on one of five criteria and of course for a given route a next hop must be given.

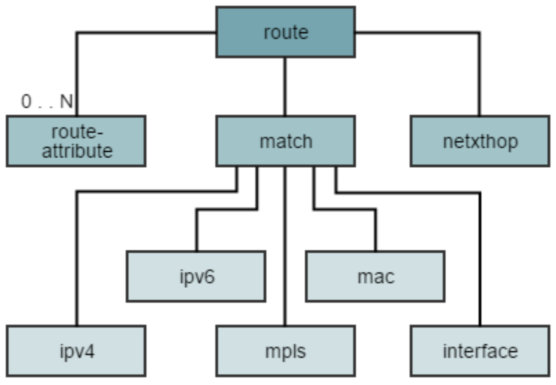

Figure 3: Interactions between I2RS Clients and agents

5. USE CASES

For a new protocol to succeed and be adopted by network operators, it has to meet their needs and expectations. It has to stem from their current and anticipated requirements. Therefore, I2RS must be designed with its envisioned use-cases in mind. Nevertheless, covering all possible use-cases can lead to the protocol bearing too many responsibilities and becoming too complex. That does not mean however, that a protocol must simply ignore proposed use-cases, but perhaps leave their support for the future or vendor specific extensions.

So for a protocol that promises standardized access for routing information across a wide variety of devices, the task of adopting and supporting use cases becomes a balancing act that should be carefully analyzed. The following are two example use-cases accepted by I2RS working group [18].

Distributed Reaction to Network Based Attacks: I2RS can be used to quickly modify control planes in case of attacks to either filter or direct suspicious traffic to analyzers. I2RS here is essential for three key elements: quick reaction times, distributed control, and the injection of temporary states that do not affect long term policies installed. Today, administrators handle this use-case by either manually entering (and later deleting) commands to filter traffic, or by asking their network provider to do such a task from their end. Both of which are not as quick and are more error prone than an automated solution provided by I2RS.

Intra-Data Center Routing: Data centers today are rapidly increasing in size and network operators are resorting to applying large multi-tiered topologies with BGP and IS-IS as routing protocols. I2RS provides operators in data centers quick access to topology changes and data flow information, which in turn translates into faster adaption and insertion of new routing policies as needed.

One possible example to the first use case is an application that collects statistics in a network. The application could use a library that acts as an I2RS client to subscribe to relevant notifications offered by I2RS agents. In one scenario, the application receives enough notifications that could lead it to conclude that an attack is being mounted on a part of the network. The application could then - using the I2RS client - issue write operations to agents in the affected part to defend against the attack by changing the routing policy. Moreover, when the attack is sensed to have come to an end, the application could revert its policy change. Such automation and interaction using complex reasoning is a key element for the future of traditional networks and SDNs.

From the previous two use-cases in addition to others, some frequent interactions of I2RS can be deduced:

- Accessing routes currently installed in the RIB as well as receiving near real-time notifications in case of their removal or change.
- Installing source and destination based routes in the RIB with all their related information.
- Interacting with traffic flow and other network traffic measurement protocols to determine path performance and make path decisions.

6. ANALYSIS

I2RS as a protocol is still in development. Nevertheless, based on the working group drafts and its reliance on NETCONF and YANG, we can analyze it and discuss its possible future implementations.

6.1 Possible Implementations

I2RS still has no existing real world implementations yet. However, some implementation efforts are already underway; one of which is currently planned at our chair for network architectures and services; we present here two approaches for possible I2RS implementations. The first approach, currently spearheaded by the chair of I2RS working group (Susan Hares) [19], relies on integrating I2RS into OpenDaylight (ODL) which is an open source platform for programmable SDNs [20]. This approach benefits from the capability of ODL to access the Linux kernel and is also supported by industry vendors who already expressed interest in supporting ODL. The other approach which is based on Bird (an open source software routing project) [21], relies on programming I2RS as another protocol like BGP or OSPF, which will allow I2RS to directly access and manipulate routes as needed [22].

These implementation efforts are essential for the development of I2RS because they prove its feasibility and deployability and most importantly they expose unanticipated shortcomings, as stated by David Clark and added to IETF Tao: "We believe in rough consensus and running code" [23]. For example, at the 94th IETF hackathon, the team working on I2RS found out that I2RS lacked any information regarding secondary pathways for sending analytical data by other protocols such as IPFIX [24], which prompted that more work and specification must be done in that area [19].

6.2 SDN & I2RS

A comparison can be made between I2RS and OpenFlow which drives SDN at the moment. OpenFlow is focused on direct interaction with the forwarding plane and essentially treats devices as simple switches [9]. On the other hand, I2RS is focused on policy change and the RIB, and actually depends on the device itself to do the appropriate forwarding plane changes. Furthermore, I2RS relies on much more interaction with existing routing protocols and technologies to enable reuse and easy deployment.

For that reason, we believe that I2RS succeeds in proving its business case to network operators better than a complete SDN solution, since its adoption won't mean the complete change of the network architecture like most SDN solutions require. They can still use all the existing solutions for routing and control plane to FIB communication, but they now have the power to automate complex operations across all their various devices. Nevertheless, it can also be projected that I2RS will help the gradual introduction of SDN approaches to traditional networks and even facilitate the adoption of what is called hybrid SDNs [25].

6.3 Reliance on NETCONF and YANG

The possibility of using NETCONF and YANG for automation of network operations was of interest long before the I2RS working group was created. A 2011 paper [26] by Tail-f Systems, highlighted the benefits of using a rich language such as YANG compared to solutions based on SNMP, in addition to the importance of transaction-based management protocol that supports consistency checking such as NETCONF.

Based on these standards, we can already have some understanding of I2RS expected performance. For example, the use of XML in NETCONF grants three main benefits: human-readability which facilitates debugging, flexibility in structuring data, and extensibility of message format [27]. Moreover, multiple studies have analyzed resource usage and efficiency of XML based solutions compared to other management protocols and the consensus was that with proper processing and data compression, XML outperforms legacy management solutions [28] [29]. Additionally, several implementations of NETCONF/YANG have been done by both the industry and the academia with partial support of older protocols like SNMP. This means that although I2RS is a new protocol, its reliance on modern yet tested protocols gives it robustness and a starting point for early implementers.

7. CONCLUSION

In this paper, we presented the I2RS protocol which is currently in development by the IETF as a solution for the challenge of accessing information stored in the routing system of today's complex and ever growing networks. I2RS aims to offer programmatic standardized interfaces in which read/write operations and event notification subscriptions are offered to network oriented applications. Its base protocols of NETCONF and YANG have both been proved to be efficient and are currently supported by major vendors and operators. Some considerations must still be amended to both of them before being adopted, for example, in terms of security.

Finally, a protocol's specification and architectural soundness do not guarantee its adoption, it must also be able to present a case for its adoption to network operators. To this end, we have shown that I2RS offers a middle ground between traditional networks and complete SDN solutions, where operators are not required to change their entire infrastructure but can deploy I2RS to leverage existing routing protocols.

8. REFERENCES

[1] Cisco Inc. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. White paper, Cisco Inc., May 2015. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf` (Accessed 9 Dec. 2015).

[2] Susan Hares and Russ White. Software-Defined Networks and the Interface to the Routing System (I2RS). *IEEE Internet Computing*, 17(4):84–88, 2013.

[3] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[4] Hyojoon Kim and Feamster, N. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, February 2013.

[5] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA, April 2012. `http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf` (Accessed 20 Dec. 2015).

[6] Cisco Inc. Cisco Application Centric Infrastructure Microsegmentation Solution. White paper, Cisco Inc., 2015. `http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-736420.pdf` (Accessed 9 Dec. 2015).

[7] Alia Atlas, Tom Nadeau, and David Ward. Interface to the Routing System Problem Statement. Internet-Draft draft-ietf-i2rs-problem-statement-08, IETF Secretariat, December 2015. `https://datatracker.ietf.org/doc/draft-ietf-i2rs-problem-statement/` (Accessed 20 Dec. 2015).

[8] I2RS Working Group. Interface to the Routing System Charter. Working Draft, October 2015. `https://datatracker.ietf.org/doc/charter-ietf-i2rs` (Accessed 16 Dec. 2015).

[9] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[10] Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall, and James R. Davin. Simple Network Management Protocol (SNMP). STD 15, RFC Editor, May 1990.

[11] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241, RFC Editor, June 2011.

[12] Alia Atlas, Joel Halpern, Susan Hares, David Ward, and Tom Nadeau. An Architecture for the Interface to the Routing System. Internet-Draft draft-ietf-i2rs-architecture-10, IETF Secretariat, November 2015. `https://datatracker.ietf.org/doc/draft-ietf-i2rs-architecture/` (Accessed 19 Dec. 2015).

[13] Susan Hares, Scott Brim, Nancy Cam-Winget, Dacheng Zhang, Qin Wu, Ahmed Abro, Salman Asadullah, Joel Halpern, and Eric Yu. I2RS Security Considerations. Internet-Draft draft-hares-i2rs-security-03, IETF Secretariat, February 2015. `https://datatracker.ietf.org/doc/draft-hares-i2rs-security/` (Accessed 11 Dec. 2015).

[14] Jeffrey Haas and Susan Hares. I2RS Ephemeral State Requirements. Internet-Draft draft-ietf-i2rs-problem-statement-02, IETF Secretariat, March 2016. `https://datatracker.ietf.org/doc/draft-ietf-i2rs-ephemeral-state/` (Accessed 19 Dec. 2015).

[15] Andy Bierman, Martin Bjorklund, and Kent Watsen. RESTCONF Protocol. Internet-Draft draft-ietf-netconf-restconf-09, IETF Secretariat, December 2015. `https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/` (Accessed 18 Dec. 2015).

[16] M. Bjorklund. YANG - A Data Modeling Language

for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor, October 2010.

[17] J. Schönwälder, M. Björklund, and P. Shafer. Network Configuration Management Using NETCONF and YANG. *Communications Magazine, IEEE*, 48(9):166–173, Sept 2010.

[18] Russ White, Susan Hares, and Alvaro Retana. Protocol Independent Use Cases for an Interface to the Routing System. Internet-Draft draft-white-i2rs-use-case-06, IETF Secretariat, July 2014. `https://datatracker.ietf.org/doc/draft-white-i2rs-use-case/` (Accessed 18 Dec. 2015).

[19] Hares, Susan. ”I2RS Implementation”. I2RS Mailing List. IETF, 16 Dec. 2015. `https://mailarchive.ietf.org/arch/msg/i2rs/KDvuzS3B_cvyF6nA-zZkKE3Ibo4` (Accessed 20 Dec. 2015).

[20] J. Medved, A. Tkacik, R. Varga, and K. Gray. OpenDaylight: Towards a Model-Driven SDN Controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6, June 2014.

[21] The Bird Internet Routing Daemon Project. `http://bird.network.cz` (Accessed 15 Dec. 2015).

[22] Private correspondence with Bird Developers. ”I2RS Implementation”. 17 Dec. 2015.

[23] Paul Hoffman. The Tao of IETF: A Novice’s Guide to the Internet Engineering Task Force, 11 2012. `https://www.ietf.org/tao.html` (Accessed 20 Dec. 2015).

[24] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. STD 77, RFC Editor, September 2013. `http://www.rfc-editor.org/rfc/rfc7011.txt`.

[25] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. N. A. CorrÃła, S. C. de Lucena, M. R. Salvador, and T. D. Nadeau. When open source meets network control planes. *Computer*, 47(11):46–54, Nov 2014.

[26] Stefan Wallin and Claes Wikström. Automating Network and Service Configuration Using NETCONF and YANG. In *Proceedings of the 25th International Conference on Large Installation System Administration*, LISA’11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[27] Gerhard Münz, Albert Antony, Falko Dressler, and Georg Carle. Using NETCONF for Configuring Monitoring Probes. In *IEEE/IFIP NETWORK OPERATIONS & MANAGEMENT SYMPOSIUM (IEEE/IFIP NOMS 2006), POSTER SESSION*, 2006.

[28] James Yu and Imad Al Ajarmeh. An Empirical Study of the NETCONF Protocol. *International conference on Networking and Services (ICNS’06)*, 0:253–258, 2010.

[29] T.F. Franco, W.Q. Lima, G. Silvestrin, R.C. Pereira, M.J.B. Almeida, L.M.R. Tarouco, L.Z. Granville, A. Beller, E. Jamhour, and M. Fonseca. Substituting COPS-PR: an evaluation of NETCONF and SOAP for policy provisioning. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*, pages 10 pp.–204, June 2006.

How-To Compare Performance of Data Plane Devices

Matthias Holdorf
Advisors: Daniel Raumer and Florian Wohlfart
Innovative Internet Technologies and Mobile Communications WS2015/2016
Chair for Network Architectures and Services
Department of Informatics, Technische Universität München
Email: matthias.holdorf@tum.de

ABSTRACT

Over the past years network technologies have grown and evolved fast. The requirements on network devices have become more complex and diverse. In order to measure and compare the performance of data plane devices, benchmarks are used. This paper describes a taxonomy of benchmark methodologies as well as the requirements and test setups to assess them. It is surveyed which of these benchmarks are used by test equipment vendors. Based on this survey, the methodologies are discussed in usefulness and feasibility of their measurements.

Keywords

Performance Testing, Benchmarks, Data Plane Devices, RFC-2544, RFC-2889, ITU-T Y.1564

1. INTRODUCTION

The convergence of several traffic types and the increasing data rates lead to higher performance requirements on networks. With new technologies emerging, such as Cloud Computing and Software-Defined Networking (SDN), today's networking architecture is transforming. [2, 3] The performance of data plane devices influences the network size, stability and reliability. The definition and tasks of a data plane device are described in chapter 2. In order to compare these devices, benchmarks are used. The term benchmark refers to a test, used to evaluate the performance of a network device under predefined starting conditions, relative to the performance of another device under the same conditions. The goal of a benchmark is to enable a meaningful comparison between two devices. [1] There are different benchmarking recommendations for data plane devices, as discussed in chapter 3.

Benchmarking was developed to measure the performance of a Device under Test (DUT). But there are additional aspects that need to be considered when defining a benchmark. One of them is comparability. In order for a benchmark to deliver a fair comparison between two DUTs, independently executed test runs should yield comparable results. This is of course, given the circumstances that both setup environments are identical. The comparability is also affected by the repeatability of a benchmark on an identical DUT in different moments in time. In order to achieve a fair comparison, the test setup and the experimentation methodologies need to be well defined. [2] The definition of the requirements, test setups and the execution of benchmark tests are depicted in chapter 4.

In chapter 5 the benchmarking methodology as described in the RFC-2544 is outlined as well as the reporting format. The usage of benchmark methodologies by test equipment vendors are investigated in chapter 6 and 7. Based on this survey, the usefulness and feasibility of the described methodologies are discussed in chapter 8.

2. NETWORK TRAFFIC PLANES

Every network device can be partitioned into three basic elements with distinct activities: (1) the data plane, (2) the control plane and (3) the management plane. Each logically separated plane classifies a different type of traffic in the network. Every plane has its own distinctive characteristic and security requirements. These three planes are the components of the layered architecture that networks have evolved to today. [4, 5, 6, 9] In traditional networking, the three planes are implemented in the firmware of routers and switches. [7, 8] In the following chapters the three network traffic planes are described in more detail. The task of each device is explained, their distinction in responsibilities as well as their interdependencies.

2.1 Management Plane

The management plane handles the administrative interface into the overall system. It is also associated with monitoring, configuration and maintenance of a system. The management plane is often considered as a subset of the control plane. [9, 10]

2.2 Control Plane

The control plane is responsible for routing decisions. It is the *Signalling* of the network. Therefore, it comprises the protocols by which routers learn the forwarding topologies and the state of the network. Implementing these complex protocols in the data plane would lead to poor forwarding performance. Thus, it maintains the information necessary for the data plane to operate. [9, 10, 20, 21] This information is collected by routing protocols like Open Shortest Path First (OSPF), Enhanced Interior Gateway Routing Protocol (EIGRP) or Border Gateway Protocol (BGP). [10]

The control plane informs the data plane about the collected information. This updates the Routing Information Base (RIB) or a separate Forwarding Information Base (FIB) of the data plane. End users rarely interact with the control plane. One exception is the ICMP ping, where a control plane protocol can be directly employed. [9]

2.3 Data Plane

The data plane is also known as forwarding plane. It is defined as the part of the network that carries the traffic. It enables data transfer to and from clients, handling multiple conversations through multiple protocols. Data Plane traffic should not have destination IP addresses that refer to any networking device. It should rather be sourced from and destined to a certain devices e.g. a server or client. The main task of a router in the case of the data plane is to merely forward a packet. [9]

Under normal circumstances transit packets constitute a large amount of traffic that enters the data plane. This is the reason why routers use specialized forwarding hardware, such as Application-Specific Integrated Circuits (ASIC), to accomplish this forwarding as fast as possible. [9, 10] However, there are exceptions that need to be taken into account. Not every transit packet belongs to the data plane and not only transit traffic is forwarded by the data plane. In the case of such an exception, additional router processing resources are consumed to forward a packet. The data plane should be focused on forwarding packets but is yet commonly burdened by other activities: NAT session creation and NAT table maintenance, NetFlow Accounting, Access Control List (ACL) logging and error signalling (ICMP). [5, 9]

In order to define data plane devices, the definition that every networking device consists of the three layered plane architecture (including the data plane) is used. Thus, in order to define a data plane device, we first need to define a network device. [4, 5, 6, 9]

After the examination of additional sources [11, 12, 13, 14, 15, 16], the following devices can be classified as data plane devices: Network Interface Card, Repeater, Hub, Bridge, Switch, Router, Firewall and Gateway. Since the survey of all devices is beyond the scope of this paper, only the performance benchmarking of routers and switches are examined.

3. BENCHMARK STANDARDS

The Benchmarking Methodology Working Group (BMWG) is a working group of the Internet Engineering Task Force (IETF). It proposes recommendations concerning the performance of networking devices and services. The BMWG defines benchmark methodologies for the management, control and data plane. [17, 18, 19]

The Request for Comments (RFC) 2544: *Benchmarking methodology for network interconnect devices* [23], which is proposed by the BMWG, is widely accepted in the industry. The proposal became an international standard for testing the performance of networking devices. [49, 50] It provides the benchmarking tests for the performance indices defined in the RFC-1242 [22], as well as the test setup conditions to apply and report format to document the tests. [1] Since the RFC-2544 does not address some of the specificities of IPv6, a new recommendation RFC-5180: *IPv6 Benchmarking Methodology for Network Interconnect Devices* [45] was proposed by the BMWG.

The BMWG further proposed the RFC-2889: *Benchmarking Methodology for LAN Switching Devices* [24], which extends the general methodology for benchmarking tests defined in the RFC-2544 to specific properties of LAN switching devices. This proposal primarily focuses on devices which can be assigned to the OSI-Layer 2.

There are additional recommendations from other organizations which can be applied to performance test data plane devices. [41] One of them is the ITU-T Y.1564: *Ethernet Service Activation Test Methodology* [42], which is also called *EtherSAM*. [43] This recommendation was developed to address drawbacks of the RFC-2544 in terms of testing today's Ethernet services and validating service-level agreements (SLA). [43] The MEF 14, proposed by the Metro Ethernet Forum (MEF), defines test procedures that may be specified as part of a Service Level Specification (SLS) for an Ethernet service. [44]

Moreover, network testing service providers define their own benchmark recommendations. The switch testing test plan from Ixia [48] builds on the RFC-2544 and RFC-2889, but extends them by additional benchmarking tests.

4. PREREQUISITE FOR TESTING

This chapter describes the prerequisites that need to be defined prior to the application of performance testing. In particular, these are the requirements, the test setup and the test execution. In order to have a fair comparison between two or more DUTs, it is essential that these topics are well defined.

4.1 Requirements

The DUT must be configured by the provided instructions. Particularly, it is anticipated that all of the supported protocols are configured and enabled. It is expected that all performance benchmarking tests run without altering the configuration of the DUT in any way other than specified in the requirements for the specific test. This should prevent manipulation to enhance the test results. For example, it is not allowed to disable all transport protocols but one; while testing that specific transport protocol. Further, the DUT should include the usual recommended routing update intervals and keep alive frequency. This procedure should ensure transparency and therefore a fair comparison between DUTs. [23]

In order to facilitate this transparency, well specified frame formats and sizes should be used while performing the tests. In addition, it is of interest to know the performance of a DUT under a number of different conditions. The performance tests should be applied under as many conditions as the test equipment can simulate. Therefore, the test suite should first be run without any modification. After that, the test should be repeated under each available condition separately. If the number of conditions or interesting condition combinations is feasible, the tests may also be performed while successively adding conditions. [23]

While not all possible manipulation can be covered, and the single DUTs vary in complexity and setting options, the exact configuration of the DUT and software, including which functions are disabled, have to be included as part of the report of a performance benchmark test. [23]

4.2 Test Setup

The RFC-2544 document moreover explains how the defined benchmark tests may be set up. Ideally, the series of tests is performed with a tester with both transmitting and receiving ports. Consequently a connection is made from the sending ports of the tester to the receiving ports of the DUT and also another connection from the sending ports of the DUT to the receiving ports of the tester (see Figure 1). This way the tester can determine how many packets received by the DUT were transmitted. In addition, the tester can verify that the correct packets were received. [23]

```
                  +------------+
                  |            |
 +----------------|   tester   |<-------------+
 |                |            |              |
 |                +------------+              |
 |                                            |
 |                +------------+              |
 |                |            |              |
 +--------------->|    DUT     |--------------+
                  |            |
                  +------------+
```

Figure 1: Test Setup [23]

4.3 Test Execution

The execution of a performance test as described in [23] consists of multiple trials. Each of these trials returns a specific piece of information, for example the throughput at a particular input frame rate. There are five phases which each trial undertakes according to the RFC-2544:

1. In case the DUT is a router, send the routing update and pause two seconds to ensure the update is done.
2. Send the learning frames to the output port and wait for two seconds. The learning frames to be used should be uniformly specified.
3. Run the performance test trial.
4. Wait for two seconds for any remainder frames to be received.
5. Wait for at least five seconds for the DUT to stabilize.

The objective for benchmarking tests is to determine the results which can be continuously expected by the DUT. Therefore, the duration of a trial must be a compromise between the accuracy and the duration of a trial. While more trials yield to better statistical evaluable results, it might not be feasible to run a lot of trials with a long duration, especially when different conditions should be taken into account. However, the duration of a trial should be at least 60 seconds.

5. TAXONOMY OF PERFORMANCE TEST

This chapter describes the performance benchmark tests and the reporting formats which are defined in the broadly accepted standard document RFC-2544. The performance indexes for these tests are derived from the RFC-1242. [23]

5.1 Throughput

The objective is to determine the packet forwarding capability as defined in the RFC-1242. This refers to the maximum amount of packets per second that a DUT can forward without losing any packet. [1] In order to determine the throughput of a DUT, the following procedure is applied: Send a specific number of frames at a particular rate through the DUT. Afterwards, the number of transmitted frames by the DUT is counted. If fewer frames are transmitted than sent to the DUT, the test is rerun with a reduced number of frames sent to the DUT. The throughput is defined as "the fastest rate at which the count of test frames transmitted by the DUT is equal to the number of test frames sent to it by the test equipment." [23]

The achieved results of this benchmark test should be reported as a graph. The X coordinate of the axis describes the frame size and the Y coordinate represents the frame rate. Further, there should be one line which shows the theoretical frame rate for the media at the specific frame size. A second line will represent the actual test findings. Additionally, the protocol, data stream format and type of media used in the test should be described. This will improve transparency even further. [23]

5.2 Latency

The purpose of this test is to determine the latency as defined in the RFC-1242. The latency test measures the time a frame needs to travel from the DUT through the network to the destination device. At first the throughput of the DUT needs to be measured, to ensure the frames are transmitted without being discarded. The second step is for the packet generator to send traffic for 120 seconds. Every 60 seconds an identifying tag is included in one frame. The time at which this frame is fully transmitted is recorded (Timestamp A). The receiver logic in the test equipment recognizes the tagged frame and records the time at which it was received (Timestamp B). The latency value is the result of the subtraction of Timestamp B and Timestamp A. [22, 23, 32] This test can be configured to measure the round-trip time. [32, 43]

The report of the result must specify which definition of latency according to the RFC-1242 was used. The latency should be reported as a table where the rows contain the frame size. The columns of the table should represent the rate at which the latency test was run, the media type and the resultant latency value. For each frame size, the measurement must be conducted 20 times. The reporting value is the average of these measurements. [23, 32]

5.3 Packet Loss Rate

The target is to determine the frame loss rate as defined in the RFC-1242. The test equipment sends a specific number of frames at maximum line rate and then measures whether the network dropped any frames. If this is the case, the values are recorded and the test is restarted at a slower rate. The granularity of reducing the frame rate must be at least 10%, while a finer granularity is encouraged. This test is repeated until there are two successive trials in which no frames are lost. [1, 22, 23, 32]

The achieved results should be reported as a graph. The X axis constitutes the input frame rate as a percentage of the theoretical rate of the media. The Y axis depicts the percent loss at the given input rate. [23]

5.4 Back-to-Back Frame

The object of this test is to characterize the ability of a DUT to process back-to-back frames as defined in the RFC-1242. It assesses the buffering capability of a DUT. The test determines the maximum number of frames received before a frame is lost. An increasing number of devices can produce bursts of back-to-back frames. Since the MTU of Ethernet networks is relatively small, many fragments have to be transmitted. The loss of even one fragment can cause an endless loop as the sender continuously attempts to send the data again.

For the execution of the test, a burst of back-to-back frames with minimum inter-frame gap is sent to the DUT. Should a frame be dropped, the burst length is decreased. If the frames are received without any errors, the burst length will be increased. Each trial should be at least two seconds long. The measurement should be repeated 50 times. The back-to-back value is the average of the recorded values being reported for each frame size. [22, 23, 32]

The back-to-back frame benchmark should be reported as a table. In that case the rows represent the tested frame size. The columns show the average frame count for each type of data stream tested. Additionally, the standard deviation for each measurement can be reported. [23]

5.5 System Recovery

The purpose of this test is to characterize the speed at which a DUT recovers from an overload condition. The test procedure starts by measuring the throughput of a DUT. Afterwards a stream of frames at a rate of the minimum of 110% of the assessed throughput or maximum rate for the media is sent to the DUT for at least 60 seconds. At a tagged frame (Timestamp A) the frame rate is reduced to 50% of the initial rate. After that change in the frame rate, the time of the last frame lost is recorded (Timestamp B). The system recovery time is obtained by subtracting Timestamp B from Timestamp A. [23]

The benchmark test should be reported in the format of a table. Whereby the rows specify the tested frame sizes. The columns of the table constitute the frame rate used as well as the measured recovery time for each type of data stream tested. [23]

5.6 Reset

The target is to determine the speed at which a DUT recovers from a device or software reset. First, the throughput benchmark needs to be performed for the minimum frame size on the media used in the trial. After that, a continuous stream of data is sent to the DUT at the recorded throughput rate. Then a reset is caused in the DUT. The time of the last frame of the initial stream being received (Timestamp A) and the first frame of the new stream being received (Timestamp B) needs to be recorded. The reset value is determined by subtracting Timestamp A from Timestamp B. The report format is a simple set of statements, one for each reset type. [23]

6. HARDWARE TEST EQUIPMENT

This chapter examines the usage of performance methodologies in hardware test equipment and describes the adjustments and notes from vendors. By that, we can make conclusions about their usefulness and applicability.

6.1 Agilent

Agilent Technologies [25] uses the RFC-2544 methodology in two of their products which are called Agilent FrameScope$^{\text{TM}}$ and Agilent FrameScope$^{\text{TM}}$ pro. [26] The company emphasizes that the RFC-2544 is not a standard, but did became increasingly popular and well-accepted to determine the performance of network devices and therefore is used for benchmarking performance tests. As described by Agilent Technologies: In order to meet the requirements of the RFC-2544, a considerable amount of configuration needs to be done. The execution as well as the set up for testing is very time consuming. The Agilent Agilent FrameScope$^{\text{TM}}$ therefore incorporates several efficiency improvements to the testing as defined in the RFC-2544. Out of the six performance tests defined in the RFC, only throughput, latency, frame loss rate and back-to-back frames are supported. Hence, the system recovery and reset tests are omitted. [26]

The test parameter setup of the FrameScope$^{\text{TM}}$ allows for greater flexibility as defined in the RFC-2544. The tester has a choice of whether the testing will be done upstream only or downstream only or both. Moreover, the performance tests can be extended to frame sizes outside the range specified by the RFC-2544. [26]

The test suite of the product allows for configuring and saving all RFC-2544 testing parameters. This enables for testing under different conditions by modifications of the given parameters in a reasonable amount of time. The RFC-2544 specifies that one trial should last at least 60 seconds. [23] This makes testing time consuming. For this reason the test equipment allows for automated testing. Another crucial requirement in order to allow for transparency and fair comparison of DUTs is a meaningful reporting format. Therefore Agilent Technologies implements a web based reporting tool which satisfies the reporting requirements as defined in the RFC-2544. This document can also serve for Service Level Agreement (SLA) verification between a service provider and a customer. [26]

6.2 Albedo

Albedo [27] implements the benchmarking tests as defined in the RFC-2544 and ITU-T Y.1564 in their test equipment Ether.Giga. [28] The test device supports up to 10 Gigabit Ethernet. In the document *Ethernet RFC-2544 explained* [29], the company describes their motive and how they apply the performance tests.

A tester consists of both transmitting and receiving ports. The tester includes sequence numbers in the frames it transmits, in order to check that all frames transmitted are also received back. The test equipment can be used to test OSI-Layer 2 and OSI-Layer 3 data plane devices. However, one criterion that is not matched by the test equipment is the test duration as defined in the RFC-2544. Since the RFC was designed for laboratory testing [23, 42], the trials may take several days to complete. This duration is not feasible when applied in practice. The time can be reduced by the selection of certain tests to be run as well as reducing their duration. This violates the requirement that every possible condition should be tested, which can be supported by a DUT. Further, it violates the requirement that a trial should run at least 60 seconds. [23, 29]

6.3 Exfo

Exfo [30] uses the RFC-2544 and ITU-T Y.1564 as performance benchmark methodologies in their test equipment Power Blazer, which supports 100 Gigabit Ethernet. [31] The motivation behind this decision is described in [32] as following: The customer's SLA dictate certain performance criteria which must be met. However, Ethernet performance criteria are difficult to prove and cannot be accomplished accurately by bit-error-rate (BER) anymore.

The portable RFC-2544 test equipment provided by Exfo enables performance testers to immediately test and demonstrate that the data plane device meets the customer's SLA. The results captured this way may serve for further comparisons of different devices. [32] The test equipment makes an addition to the defined benchmark tests in the RFC-2544, by the measurement of packet jitter. This is crucial, because exorbitant jitter can cause failures in real-time applications. This can cause dropout effects in VoIP applications. For video applications this can cause images to falsify. [32] As criticized in chapter 6.1, the duration of a trial as defined in the RFC-2544 leads to a problem. Every test should be performed by each defined frame sizes as defined in [23]. Further, each test trial has 20 iterations. This will yield to a length of almost five hours, which is not feasible. Therefore customization is needed. This can be accomplished by testing only two out of the seven defined frame sizes or conducting only two out of the six defined performance tests. It depends on the type of data plane device and the area in which it will be applied. [32]

6.4 Spirent

The Spirent TestCenter 2.0 as described in [34] is a comprehensive test suite which provides OSI-Layer 2-7 testing for up to 10 Gigabit Ethernet. The test suite implements both RFC-2544 and RFC-2889. Spirent [33] describes these RFCs as industry-standard. [34] It extends the benchmark methodology defined in the RFC-2544 similar to [26] by jumbo Ethernet frames in order to test streaming and conformance testing for certain protocols. The test suite contains six major fields of testing: Ethernet Switch Testing, Enterprise/Metro Router Testing, Carrier Ethernet Testing, Broadband Access Testing, Layer 4-7 Testing as well as IPTV and Video Quality Testing. [34]

Another test equipment product developed by Spirent is the Router Performance Tester AX/4000 (RPT). [35] The RPT allows for customizing of IP test packets and traffic generation through the DUT at full line rate and reports in real-time. The device supports testing data plane devices according to the RFC-2544. However, as in [26] and [32] the two tests of system recovery and reset as defined in the RFC-2544 are not supported. [35]

6.5 Viavi

An Ethernet testing solution provided by Viavi [36], described in the product note [38], is building on the RFC-2544 as well. The test equipment further supports the Y.1564 standard mentioned in chapter 3 and can saturate rates up to 10 Gigabit. The two standards can be tested asymmetrically with a unidirectional upstream and downstream traffic. Further test applications supported are mobile Ethernet backhaul up to LTE, cloud connectivity and fault isolation. [38]

6.6 Ixia

The IxAutomate test suite [47] from Ixia [46] implements both RFC-2544 and RFC-2889 recommendations. While the RFC-2544 was designed as a general methodology for networking devices of all types, the RFC-2889 was written specifically to benchmark the data plane performance of OSI-Layer 2 LAN switching devices. Additionally, the new recommendation RFC-2544-IPv6 [45] is integrated into the test equipment. While all benchmarking tests of the RFC-2889 are implemented, the system recovery and reset test as defined in the RFC-2544 are omitted. [47]

6.7 Xena Networks

The company Xena Networks [39] offers with XenaBay and XenaCompact two OSI-layer 2 and 3 test chassis which support up to 100 Gigabit Ethernet. These chassis can be configured with different test software provided by Xena. Specifically, software is provided for the following test methodologies: RFC-2544, RFC-2889, RFC-3918 and ITU-T Y.1564. Further, a software packet is provided for scripting test automation. [40]

7. SOFTWARE TEST EQUIPMENT

Besides testing data plane devices with hardware equipment there is the possibility to measure their performance with software. Approaches exist in PacketExpert [53], LAN Tornado RFC 2544 [54], iPerf [55], Ostinato [56] or MoonGen [57]. PacketExpert, LAN Tornado RFC 2544 and MoonGen come with direct support for the RFC-2544.

Both hardware and software testing procedures have to face the same challenges: performance, flexibility, and precision in timestamping and rate control. The advantages of software traffic generators are their flexibility and their low costs. The MoonGen [59] software e.g., uses Lua Scripts which allow for modification of each single packet and runs on Intel commodity NICs. Hardware test equipment on the other hand have advantages in performance and precision. [57]

It is important for test equipment to saturate high rates, while still being precise to assure repeatability of the same experiment in different moments in time and a fair comparison between different DUTs. If this can not be assured, the outcome of the tests have no value. This is the major downside of software packet generators, which often lack performance capabilities and precision. [58]

Botta *et al.* [58] discuss the inherent problems of software packet generator in more detail. While the MoonGen [59] project presents new approaches which overcome some of these disadvantages.

8. USEFULNESS AND FEASIBILITY

Based on the evaluation in chapter 6 and 7, we can conclude that test equipment vendors are using the RFC-2544, RFC-2889 and ITU-T Y.1564 as standards to build their test equipment. Further, academic research is interested in developing network testing equipment that supports the RFC-2544. [1, 51, 52, 57]

However, there were also criticism and disadvantages mentioned from these parties. Since the RFC-2544 was written over a decade ago, in 1999, it is no longer sufficient in terms of fully validating today's Ethernet services. The RFC-2544 does not satisfy all requirements anymore, such as packet jitter, QoS measurement or multiple concurrent service levels. Also, the method of sequential testing takes several hours to complete, as mentioned by manufacturers. [25, 27] This test method is both time consuming and costly. Furthermore, the system recovery and reset test as defined in the RFC-2544 were rarely implemented by any test equipment vendor.

In the throughput test, the RFC-2544 makes no distinction between committed and excess traffic. This is not sufficient for testing SLA. The frame delay is determined based on the assessment of a single frame during a trial. This approach does not take into account any fluctuation or peak that may occur during the testing. Furthermore, the RFC-2544 does not measure the inter-frame delay variation (IFDV). [43, 63] Most of the tests defined in the RFC-2544 are performed with one endpoint generating traffic and another endpoint placed in loopback. While this is the simplest and fastest way to perform a test trial, there are disadvantages to this test setup. When a test fails, there is no information on where packets are being dropped or where delay is being introduced. [37] A solution to that is the testing in an asymmetric mode, as adapted by manufacturers. [25, 36]

Additionally, vendors of test equipment like Exfo, Ixia and MRV have concerns about the feasibility of the test methodologies described in the RFC-2544. [63, 64, 65] This is due to the reason of time consumption of the test trials as well as the lack of validating certain features. Ixia [64] states that the RFC-2544 and RFC-2889 are good for testing best case scenarios in a laboratory environment, but they do not provide an insight into the device performance under a real-world data center traffic load. Additionally, they do not assess performance of mixed frame sizes. Both [64] and [65] therefore suggest the usage of the ITU-T Y.1564. [42] This is reflected by the usage of methodologies by test equipment vendors as surveyed in chapter 6. The evaluation (see Table 1) shows, that all examined test equipment, developed after the approval of the ITU-T Y.1564 in March 2011, implement the suggest methodology.

Test Equipment	RFC-2544	Y.1564	GbE	Year
Agilent FrameScope	yes	no	1	2006
Albedo Ether.Giga	yes	yes	10	2012
Exfo Power Blazer	yes	yes	100	2015
Spirent TestCenter	yes	no	10	2006
Viavi QT-600-10	yes	yes	10	2015
Ixia IxNetwork	yes	no	1	2007
Xena XenaBay	yes	yes	100	2014

Table 1: Usage of benchmarking methodologies

The BMWG is continuously working on the mentioned disadvantages. The proposal of the RFC-4814 [62] considers the overlooked factors in network device benchmarking and addresses the issues in the RFC-2544 and RFC-2889. In the RFC-6815 [61], an applicability Statement for the RFC-2544, the BMWG states that actual methods may vary from the methodology described in the RFC-2544. Another proposal [60] considers the issues related to conducting tests similar to the RFC-2544 in a production network. The IETF has addressed the scenario of production network performance testing by commissioning a new working group by the name of IP Performance Metrics (IPPM). [60]

9. CONCLUSION

The RFC-2544 defines a methodology for benchmarking the data plane performance of networking devices. It measures a networking device's throughput, latency and frame loss rate for specified frame sizes and further defines a system recovery and reset test. On the basis of the RFC-2544, the RFC-2889 defines benchmarking tests for an OSI-layer 2 LAN switching device. While the ITU-T Y.1564 methodology addresses issues of today's Ethernet Services and SLA validations. The definition of the requirements, the test setup as well as the reporting format ensures transparency and therefore a fair comparison of different data plane devices. These methodologies are widely accepted for benchmarking data plane devices and for this reason established themselves as a standards.

Since the RFC-2544 was written over a decade ago in 1999, the methodology does not assess all Ethernet services that are available today. Further, the sequential approach of testing is very time consuming. For this reason new standards were developed. The ITU-T Y.1564 and the MEF 14 methodologies attempt to overcome the disadvantages of the RFC-2544. Both standards are used by test equipment vendors. The BMWG itself is aware of the disadvantages of the RFC-2544 and continuous to work at the topic of benchmarking data plane devices. The RFC-6815 and RFC-4814 reference the RFC-2544 and are addressing the concerns of industry and academic research.

Even if some sources testify that the RFC-2544 is deprecated in terms of testing today's Ethernet services, it still remains a foundation for the development of performance benchmarking methodologies and will therefore continue to contribute on how the performance of data plane devices is measured.

10. REFERENCES

[1] L. Niu, G. Feng and M. Duan: *Implementation of Instrument for Testing Performance of Network Based on RFC2544 Test*, In International Journal of Hybrid Information Technology Vol. 8, No. 2, pages 323-332, 2015.

[2] S. Bouckaert, J. V. V. Gerwen, I. Moerman, S. C. Phillips, and J. Wilander: *Benchmarking computers and computer networks*, EU FIRE White Paper, 2010.

[3] Meru Networks: *Demystifying Software-Defined Networking for Enterprise Networks*, 2013.

[4] M. Brown and R., Burns: *Cisco CCNA data center DCICT 640-916 official certification guide*, 2013.

[5] I. Pepelnjak: *Management, Control and Data Planes in Network Devices and Systems*, `http://blog.ipspace.net/2013/08/management-`

control-and-data-planes-in.html, 2013.

[6] B. Salisbury: *The Control Plane, Data Plane and Forwarding Plane in Networks*, http://networkstatic.net/the-control-plane-data-plane-and-forwarding-plane-in-networks, 2012.

[7] Open Networking Foundation: *Software-Defined Networking: The Norm for Networks*, 2012.

[8] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig: *Software-Defined Networking: A Comprehensive Survey*, In Proceedings of the IEEE Vol. 103, No. 1, 2015.

[9] G. Schudel and D. J. Smith: *Router Security Strategies: Securing IP Network Traffic Planes*, Cisco Press, pages 24-30, 2008.

[10] N. Shamsee,Klebanov D., H. Fayed, A. Afrose and O. Karakok: *CCNA Data Center DCICT 640-916 Official Cert Guide*, 2015.

[11] Savvius: *Network Interconnect Devices: Repeaters, Bridges, Switches, Routers*, http://www.wildpackets.com/resources/compendium/interconnect_devices/overview, 2015.

[12] SCO OpenServer: *Networking Guide*, 2003.

[13] C. Wells: *Network Interconnection Devices, TechnologyUK*, http://www.technologyuk.net/telecommunications/networks/interconnection_devices.shtml, 2001.

[14] A. Groessler: *Internetworking: Devices*, http://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/1996-ss/rn96/CN-Title/form/intdevie.htm, 1995.

[15] ComputerNetwokringNotes: *Networking Devices: Hub, Switch, Router, Modem, Bridges, and Brouters Gateways*, http://computernetworkingnotes.com/comptia-n-plus-study-guide/network-devices-hub-switch-router.html, 2013

[16] CISCO: *Layer 2 and Layer 3 Switch Evolution*, The Internet Protocol Journal Vol. 1, No. 2, September 1998.

[17] H. Alvestrand: *RFC-3935: A Mission Statement for the IETF,* *https://www.ietf.org/rfc/rfc3935.txt*, 2004.

[18] E. Huizer: *RFC-1603: IETF Working Group Guidelines and Procedures*, https://tools.ietf.org/rfc/rfc1603.txt, 1994.

[19] BMWG: Charter for Working Group, http://datatracker.ietf.org/wg/bmwg/charter/, 2015.

[20] L. Yang, R. Dantu, T. Anderson and R. Gopal: *RFC-3746: Forwarding and Control Element Separation (ForCES) Framework*, IETF Network Working Group, 2004.

[21] M. Fratto: *What is the difference between Control plane and Data plane in the context of networking equipment like routers/switches?*, https://www.quora.com/What-is-the-difference-between-Control-plane-and-Data-plane-in-the-context-of-networking-equipment-like-routers-switches, 2012.

[22] S. Bradner: RFC-1242: *Benchmarking Terminology for Network Interconnection Devices*, IETF Benchmarking Methodology Group, July 1991.

[23] S. Bradner and J. McQuaid: *RFC-2544: Benchmarking methodology for network interconnect devices*, IETF Benchmarking Methodology Group, March 1999.

[24] R. Mandeville and J. Perser: *RFC-2889: Benchmarking Methodology for LAN Switching Devices*, IETF Benchmarking Methodology Group, August 2000.

[25] Agilent Technologies: http://www.agilent.de/home, November 2015.

[26] H. Pandya: *RFC-2544 Network Performance Testing with the Agilent FrameScope*^{TM}, Agilent Technologies, January 2006.

[27] Albedo: http://www.albedo.com/, November 2015.

[28] Albedo: *Ether.Giga*, http://www.albedotelecom.com/pages/fieldtools/src/ethergiga.php, February 2016.

[29] P. Fan: *Ethernet RFC-2544 explained*, Albedo, April 2003.

[30] Exfo: http://www.exfo.com/, November 2015.

[31] Exfo: Power Blazer http://www.exfo.com/products/field-network-testing/bu2-transport-datacom/ethernet-testing/ftb-88100nge-power-blazer, February 2016.

[32] B. Giguère: *RFC 2544: How it helps qualify a Carrier Ethernet Network*, Exfo, August 2004.

[33] Spirent: http://www.spirent.com/, November 2015.

[34] Spirent: *Get There Faster with Spirent TestCenter*^{TM}, May 2007.

[35] Spirent: *Router Performance Tester: AX/4000*, September 2005.

[36] Viavi: http://www.viavisolutions.com/en-us, November 2015.

[37] Viavi: *Viavi Solutions*^{TM} *Ethernet Testing*, November 2015.

[38] Viavi: *QT-600-10 10 Gigabit Ethernet Test Head*, June 2015.

[39] Xena Networks: http://www.xenanetworks.com/, November 2015.

[40] Xena Networks: *Xena Scripting*,http://www.xenanetworks.com/test-software/xena-management-sw/xenascripting/, February 2016.

[41] Ixia: *Ultra Low Latency (ULL) Testing*, June 2011.

[42] ITU-T: *Ethernet service activation test methodology*, March 2011.

[43] T. Diallo and M. Dorais: *EtherSAM: The new Standard in Ethernet Service Testing*, January 2014.

[44] Metro: *Abstract Test Suite for Traffic Management Phase 1*, November 2005.

[45] C. Popoviciu, A. Hamza, G. V. de Velde and D. Dugatkin: *RFC5180: IPv6 Benchmarking Methodology for Network Interconnect Devices*, IETF Benchmarking Methodology Group, May 2008.

[46] Ixia: http://www.ixiacom.com/, November 2015.

[47] Ixia: *IxAutomate RFC Benchmarking Test Suites*,

July 2007.

[48] Ixia: *Switch Testing: IxAutomate*, Test Plan, April 2006.

[49] Eantc: *Huawei Technologies Service Activation Using RFC 2544 Tests*, Mai 2008.

[50] Tolly: *Araknis Networks AN-300 Series Gigabit PoE Switch: Performance and Feature Comparison Versus Cisco Systems and Pakedge Device & Software*, Test Report, March 2014.

[51] C. Both, C. Battisti and F. Kuentzer: *FPGA implementation and performance evaluation of an RFC 2544 compliant Ethernet test set*, In Int. J. High Perform Systems Architecture Vol. 2, No. 2, pages 107-115, 2009.

[52] Y. Wang, Y. Liu, X. Tao and Q. He: *An FPGA-based high-speed network performance measurement for RFC 2544*, In EURASIP Journal on Wireless Communications and Networking, No. 1, pages 1-10, 2015.

[53] GL Communications Inc.: *PacketExpert*, `http://www.gl.com/packetexpert-rfc-2544-ber-loopback-testing.html`, February 2016.

[54] SoftDevTeam: *LAN Tornado RFC 2544*, `http://lan-tornado-rfc-2544.soft112.com/`, August 2012.

[55] iPerf: `https://iperf.fr/iperf-doc.php`, February 2016.

[56] Ostinato: `http://ostinato.org/`, February 2016.

[57] P. Emmerich, F. Wohlfart, D. Raumer and G. Carle: MoonGen: *A Scriptable High-Speed Packet Generator, ArXiv e-prints*, October 2014.

[58] A. Botta, A. Dainotti and A. Pescapé: *Do You Trust Your Software-Based Traffic Generator?*, IEEE Communications Magazine, IEEE 48 No. 9, pages 158-165, 2010.

[59] MoonGen:`https://github.com/emmericp/MoonGen/tree/master/rfc2544/benchmarks`, December 2015.

[60] R. Bonica and S. Bryant: *RFC2544: Testing in Production Networks, IETF Benchmarking Methodology Group*, October 2012.

[61] S. Bradner, K. Dubray, J. McQuaid and A. Morton: *RFC6815: Applicability Statement for RFC 2544: Use on Production Networks Considered Harmful*, IETF Benchmarking Methodology Group, November 2012.

[62] D. Newman and T. Player: RFC-4814: *Hash and Stuffing: Overlooked Factors in Network Device Benchmarking*, IETF Benchmarking Methodology Group, March 2007.

[63] Exfo: *Are You Still Testing to RFC 2544?*, `http://www.exfo.com/corporate/blog/2013/still-testing-rfc-2544-really`, December 2015.

[64] Ixia: *Is RFC2544 Enough to Benchmark the Data Center Switching Fabric Performance?*, `http://www.ixiacom.com/about-us/news-events/corporate-blog/rfc2544-enough-benchmark-data-center-switching-fabric`, December 2015.

[65] MRV: *Why RCF2544 is not sufficient anymore*, `http://www.mrv.com/blog/why-rcf2544-not-sufficient-anymore`, December 2015.

MoonGen Tutorial

Jonas Jelten
Betreuer: Paul Emmerich, Daniel Raumer
Seminar Innovative Internet-Technologien und Mobilkommunikation WS2015
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: jelten@in.tum.de

ABSTRACT

This paper provides a short introduction into the usage of MoonGen, a high performance packet generation framework written in Lua. It is based on DPDK which mediates the hardware access. You will learn how you can interact with the MoonGen API to craft and send custom packets, gather statistics and verify received data. Communication between tasks running in parallel is demonstrated. The usage of hardware features like queues and rate control is illustrated and explained. You will see that MoonGen is simple to use for many load generation use cases.

Keywords

networking, MoonGen, tutorial, howto, Lua, Linux

Version

This is the tutorial version `v1.0`.

1. WHAT'S MOONGEN?

After reading this tutorial, you will be able to use MoonGen to benchmark and test your network setup in any way you like. You'll learn the concepts, the architecture and basics of the MoonGen API.

MoonGen [2] is a software based packet generator framework, designed for easy use and high speeds at 10Gbit and more. Executed on common hardware, it can be used for just load generation in benchmarking applications, or to check the response validity for error detection by executing custom code for each packet without expensive special hardware. This way, firewalls, network address translation and quality of service setups can be tested and verified to operate correctly even under enormous load. Sub-microsecond latency and packet drops can be measured and checked if they match the expected behavior in benchmarks.

MoonGen is based on Data Plane Development Kit`DPDK` [1], which is granting direct hardware access via `DMA`, thus allowing the `LuaJIT` [6] machine to interact with the network interface at maximum speeds. In order to use MoonGen, you should have a basic knowlege of Lua, for example from a quick tutorial at
https://learnxinyminutes.com/docs/lua/ [4].

2. SETUP

MoonGen is intended to run on any GNU/Linux distribution. This guide was created on Ubuntu 14.04.

To install, clone the `git` repository from the upstream url at https://github.com/emmericp/moongen [5], then follow the prerequisite requirements and installation directions in the `README` file.

After you built the project successfully, try if you can execute `./MoonGen` and get the usage information printed. If that works, you may continue with the tutorial.

3. ARCHITECTURE

In principle, MoonGen is a high-level frontend for DPDK. DPDK provides a low-level API for hardware access, packet generation and response processing, mainly designed for data plane development [1]. MoonGen's core is a convenient lua wrapper for that API. To use it, you create custom lua files containing code instead of config files: This allows much more flexibility for any measurement application you intend to conduct.

The entry point for all the custom code is a control script, containing a `master` function. It is called by MoonGen, should set up the interfaces and request the desired settings. Internally, configuration is then passed to dpdk, which performs the actual hardware setup.

This control script can spawn new tasks as separate LuaJIT VMs. That way, packet generation, receive measurements, verifications, etc. can easily be implemented apart and executed in parallel.

After packet fields are composed in some task, they're passed to dpdk which crafts the payload and sends it out of the device. Data received from the hardware is mediated through dpdk to the lua script which can then do any verification and processing.

The running tasks can only communicate through the MoonGen API, for example via pipes and namespaces, as tasks are separate Luajit VMs in a different address space.

A graphical representation of the just described data and control flow can be seen in Figure 1. Only the custom scripts ("Userscript") are visible to the user, which then communicates with the config or data api with the hardware device.

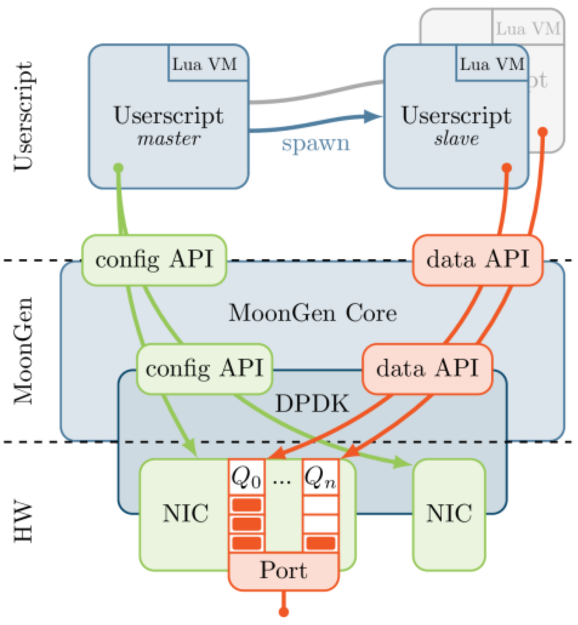

Figure 1: MoonGen architecture [2]

4. USAGE INTRODUCTION

After successful compilation of MoonGen, the invocation is very simple:

```
./MoonGen yourscript.lua [yourargs...]
```

`yourscript.lua` is your lua file is the entry script. It must contain a `function master(...)` which then performs the device setup, described in Section 4.1. Once this is done, worker tasks can be spawned as explained in Section 4.3, which then perform their duty to generate packets (Section 4.2), receive and count them (Section 4.4), communicate with each other (Section 4.6) or do whatever is appropriate for your use case.

4.1 Device setup

Before a NIC (network interface card) can be used, it must be taken away from the Linux driver so that dpdk can use it. The Python script `deps/dpkg/tools/dpdk_nic_bind.py` can detach PCI devices and set their driver to e.g. `igb_uio` or `vfio-pci` to use them with MoonGen.

Modern network interfaces have hardware features that allow a huge speedup by parallelization: They have send and receive queues, around 32 to 128 each (depending on NIC model) which allow to prepare sending out packets in parallel or assign received packets already on hardware by custom filter rules like hashing protocol header fields. This comes in handy for multicore processing, as the queues can be assigned to processors or threads independently [3].

In MoonGen, the queues are used independently and are usually requested in the entry point script. The devices are aquired and set up, queues registered and then, for example, used to send some packets.

```
local device = require "device"

function master(txNum)
    -- use specified NIC number with
    -- no listening and one transmission queues
    txDev = device.config{
        port = txNum,
        rxQueues = 0,
        txQueues = 1,
    }
    device.waitForLinks()
    send(txDev:getTxQueue(0))
end
```

To interact with an allocated queue, it is fetched from the device object by calling `queue = txDev:getTxQueue(nr)` or `getRxQueue`. You'll see this in Section 4.2.

To use the receive filter configuration mentioned earlier, configure the device upon creation. When calling `device.config`, set `rssNQueues = N` to the number of queues where packets shall be placed in. This enables automatic hashing by IPv4/6 and TCP/UDP headers to place same-header packets (from the same "flow") into the same queue selected from `0` to `N-1`.

The optional `rssFunctions` parameter controls which of the hash functions are enabled. If you don't specify it, all supported hashes are enabled. You can create a list of methods you want to use out of `RSS_FUNCTION_IPV4, .._IPV6, .._IPVX_TCP, .._IPVX_UDP`:

```
local device = require "device"
txDev = device.config{
    port = 0,
    rxQueues = 4,
    rssNQueues = 4,
    rssFunctions = {
        device.RSS_FUNCTION_IPV4,
        device.RSS_FUNCTION_IPV4_TCP,
    }
}
```

4.2 Packet generation

To compose the data to send, a memory buffer is required first. As packets are sent out asynchronously, the buffers where you are crafting them must be allocated independently, in a buffer array. The array manages many allocations of same-sized packets, maintained in a memory pool. When the data was actually sent out, the allocated buffer can be freed from the array.

In such a buffer, most data will stay the same though, so the skeleton is defined via a function previously. It's important to set up the default values of the packet in this function for the memory pool and not in the generation loop. Otherwise, performance problems will arrise.

In this example, the most simple-stupid way is used to manually set up an ethernet header in a `char` buffer.

```
local dpdk = require "dpdk"
local memory = require "memory"

function send(queue)
    local mem = memory.createMemPool(function(buf)
        local data = ffi.cast("uint8_t*", buf.pkt.data)
        for i = 0, 11 do
            data[i] = i  -- fill in mac addresses
        end

        data[12] = 0x12  -- set type to ethernet
        data[13] = 0x34
    end)
    local bufs = mem:bufArray()
    while dpdk.running()
        bufs:alloc(60)   -- size of each packet
        -- ⇑ sets up each packet with the function above
        -- ← here, single packets could be modified
        queue:send(bufs) -- schedule sending
    end
end
```

To simplify crafting of packets, the raw buffer can be casted into easy-to-use protocol header objects. Those conversions are defined in `lua/include/proto/` for all kinds of protocols, for example `getIP6Packet()` or `getUdp4Packet()`.

```
local mem = memory.createMemPool(function(buf)
    buf:getEthernetPacket():fill{
        ethSrc = txDev,   -- use device mac
        ethDst = "00:01:02:03:04:05",
        ethType = 0x1234,
    }
end)
```

If you want to implement a new protocol packet format, please copy and adapt the template file provided in `lua/include/proto/newProtocolTemplate.lua`.

To change some data for single packets, perform your operation after allocating the buffer array and before enqueuing the send-out. You can change any data of the packet and again use the convenience casts for implemented protocols. To configure the hardware transmission rate of a queue, use `queue:setRate(Mbit/s)`.

If you need to to pause the LuaJIT VM for some time period, call the `dpdk.sleepMillis(time)` function.

The following send function will only transmit data for 10 seconds, then it terminates. It creates UDP on IPv4 packets with a randomized source address.

```
local timer = require "timer"

function send(queue)
    queue:setRate(100)       -- hardware rate in Mbit/s
    dpdk.sleepMillis(1000)   -- wait one second
    local mem = memory.createMemPool(function(buf)
        buf:getUdp4Packet():fill{
            pktLength = 124,
            ethSrc = queue,   -- device mac
            ethDst = "10:11:12:13:14:15",
            -- ipSrc will be randomized
            ip4Dst = "10.13.37.1",
            udpSrc = 4321,
            udpDst = 1234,
            -- payload = \x00 (mempool initialization)
        }
    end)

    local bufs = mem:bufArray()
    local runtime = timer:new(10)   -- 10 seconds

    while runtime:running() and dpdk.running() do
        bufs:alloc(250)
        for _, buf in ipairs(bufs) do
            local pkt = buf:getUdpPacket()
            -- select a randomized source IP address
            pkt.ip4.src:set(
                parseIPAddress("10.0.42.1")
                + math.random(235))
        end
        bufs:offloadUdpChecksums()  -- harware checksums
        queue:send(bufs)
    end
end
```

4.3 Running parallel tasks

While running, MoonGen often needs parallel tasks: To send and receive packets, to create and write out statistics and counters or to do response verification. To achieve this, "slave" tasks are spawned.

The function used for this is `dpdk.launchLua("funcname", arg0, ...)`, which spawns a single slave task as a new LuaJIT VM. The task can then execute any code within the called function. Arguments are passed, this allows you to access e.g. the devices or queues within the task. The slave tasks can also be created dynamically on demand, although this should be used rarely to avoid spawning new VMs too quickly and often.

```
local dpdk = require "dpdk"
dpdk.launchLua("somefunctionname", arg0, arg1, ...)
dpdk.launchLua("otherfunction", txDev, ipaddr)
dpdk.waitForSlaves()   -- wait for child termination
```

In such a task, contents of every single received packet can be processed. The data arrives in batches, so the analysis has to loop over all packets in that batch.

```
local dpdk = require "dpdk"
local memory = require "memory"

function master(txPort, rxPort)
    local txDev = device.config{port = txPort}
    local rxDev = device.config{port = rxPort}
    device.waitForLinks()
    dpdk.launchLua("send", txDev:getTxQueue(0))
    dpdk.launchLua("recv", rxDev:getRxQueue(0))
    dpdk.waitForSlaves()
end

function recv(queue)
    local bufs = memory.bufArray()
    while dpdk.running() do
        local rx = queue:recv(bufs)
        for i = 1, rx do
            local pkt = bufs[i]:getUdp4Packet()
            print("Packet: " .. pkt.ip4:getString())
        end
        bufs:freeAll()
    end
end
```

4.4 Statistics

The `stats` module allows counting packets for statistics. After statistics were gathered, they can be written to `stdout` or to some file. The supported formats are `"plain"`, `"csv"` and `"ini"`.

To create a new packet counter that is attached to a device, call `local rxCtr = stats:newDevRxCounter(device, "plain")` or `newDevTxCounter(..)`. Information is gathered automatically by performing queries to the harware counters of the NIC.

The `newPktRxCounter("your counter name", "plain")` or `newPktTxCounter(..)` can be updated by passing packet buffers to them via its `countPacket(singleBuffer)` method.

The `newManualTxCounter("your counter name", "plain")` is suitable for counting packets and other data manually. `updateWithSize(packet_count, each_size)` must be called to increase the datameter internally.

For all those counters, their `:update()` method should be called regularly, as it will show current statistics at runtime.

The `histogram` module allows gathering statistics about a frequency distribution.

The next example incorporates the device and package counters, it just listens for packets on the given hardware port and counts the occurrences of UDP ports. The packet sizes are logged in a histogram.

```
local dpdk = require "dpdk"
local device = require "device"
local histogram = require "histogram"
local memory = require "memory"
local stats = require "stats"

function master(rxPort, saveInterval)
    local saveInterval = saveInterval or 60
    local rxDev = device.config{
        port = rxPort,
        dropEnable = false,
    }
    device.waitForLinks()

    local queue = rxDev:getRxQueue(0)
    local bufs = memory.bufArray()

    -- create the device receive counter
    local rxCtr = stats:newDevRxCounter(queue.dev)
    -- and the packet receive counter to detect
    -- packets that were dropped on the NICNIC
    local pktCtr = stats:newPktRxCounter("pkts", "plain")

    local hist = histogram:create()
    local timer = timer:new(saveInterval)
    while dpdk.running() do
        -- wait max 100ms for new data
        local rx = queue:tryRecv(bufs, 100)
        for i = 1, rx do
            local buf = bufs[i]
            local size = buf:getSize()
            hist:update(size)
            pktCtr:countPacket(buf)
        end
        bufs:free(rx)
        rxCtr:update()
        pktCtr:update()
        if timer:expired() then
            timer:reset()
            hist:print()
            hist:save("packet_sizes.csv")
        end
    end

    -- and print statistics, those should be the same.
    rxCtr:finalize()
    pktCtr:finalize()
end
```

To use the manual counter, the return value of `queue:send()` can be used. Note here that the send call is asynchronous, but the return value can still be recorded for statistics. As in the previous example, the `finalize()` call actually prints the final result, and `updateWithSize` prints the runtime status every second:

```
function send(queue)
    local mem = ...
    local packetSize = 250

    -- create manual counter
    local txCtr = stats:newManualTxCounter(port, "plain")
    local bufs = mem:bufArray()

    while dpdk.running() do
        bufs:alloc(packetSize)
        bufs:offloadUdpChecksums()
        local sentCount = queue:send(bufs)

        -- register new data: sentCount * packetSize
        txCtr:updateWithSize(sentCount, packetSize)
    end
    txCtr:finalize()
end
```

It's also easily possible to collect statistics about packet contents, for example their UDP destination port. The task is blocked until some data is received. Then, all packet buffers received are casted to UDP in IPv4 packets, which then trigger a counter creation, if it doesn't exist already. This demonstrates that counters can be created and updated dynamically as well.

```
function recv(queue)
    local bufs = memory.bufArray()
    local counters = {}

    while dpdk.running() do
        -- block until some data was received
        local rx = queue:recv(bufs)
        for i = 1, rx do
            local buf = bufs[i]

            -- cast the buffer to a known protocol
            local port = buf:getUdpPacket().udp:getDstPort()
            local ctr = counters[port]

            -- create counters dynamically
            if not ctr then
                ctr = stats:newPktRxCounter(port, "plain")
                counters[port] = ctr
            end
            -- record the packet
            ctr:countPacket(buf)
        end
        bufs:freeAll()
    end

    -- for each observed destination port, print stats:
    for _, ctr in pairs(counters) do
        ctr:finalize()
    end
end
```

4.5 Timestamping

To measure sub-microsecond delays in fiber and copper cables, MoonGen can utilize hardware timestamping features from modern NICs. The packets sent are defined in the `lua/include/proto/ptp.lua`, the precision time protocol. You can create a timestamper to use either as a layer 2 (via `timestamping:newTimestamper(txq, rxq)` or transfer it as PTP via UDP in IPv4 with `timestamping:newUdpTimestamper`.

This example measures the latency between both queues via hardware timestamping every 0.01 seconds. The queues have to be connected at the peer side, so the sent packet can take a round trip.

```
local ts = require "timestamping"

function timerTask(txq, rxq, size)
    -- create the timestamper for measuring
    -- between those queues
    local timestamper = ts:newTimestamper(txq, rxq)
    local hist = histogram:new()
    local rateLimiter = timer:new(0.01)
    while dpdk.running() do
        rateLimiter:reset()
        hist:update(timestamper:measureLatency(size))
        rateLimiter:busyWait()
    end
    hist:print()
    hist:save("histogram.csv")
end
```

4.6 Task communication

The simplest inter-task communication API provided by MoonGen are pipes. For example, you can send rate adjustment messages, pass statistics or transfer any communication that is not performance-critical through a pipe shared by two tasks. To use, create the pipe in a common context, for example the master function. This pipe can communicate accross LuaJIT VMs and can send arbitrary data, which is serialized and unserialized using the “serpent” library.

```
local pipe = require "pipe"

-- create a new pipe in the parent task
local p = pipe:newSlowPipe()
p:send(0, 13, 37, 42)          -- send array
p:send("the cake is a lie")    -- send string
-- or send a table
p:send({235, lol = "rofl", subtable = {1}})

-- number of waiting messages
local enqueued = p:count()

-- receiving
local a, b, c, d = p:recv() -- equals tryRecv(10)
local txt = p:tryRecv(100)  -- wait time microseconds
                            -- and return answer

-- pass this pipe when creating another task
-- it can then access it like above.
dpdk.launchLua("somefunction", p)
dpdk.waitForSlaves()
```

The alternative to pipes are namespaces, which are global variables between LuaJIT VMs, implemented as lua table. They are also slow like the pipes from above, as data is transferred between VMs.

`local space = namespaces::get("name")` will create or fetch an already existing global namespace, which can then be accessed like this:

```
local dpdk = require "dpdk"
local namespaces = require "namespaces"

function master()
    local space = namespaces:get("mine")
    space.string = "data!"
    space.answer = 42
    space.table = { black = "mesa", { 1 } }
    dpdk.launchLua("slave"):wait()
end

function slave()
    -- can access the same namespace!
    local slavespace = namespaces:get("mine")
    print("data? " .. slavespace.string)
end
```

4.7 Traffic patterns

The hardware rate control feature of some NICs can only be set to a constant rate. To send non-constant traffic patterns of valid packets, MoonGen fills the gaps between the packets with invalid data. That way, the sending card is kept busy and times sends correctly, but devices along the tested way will hopefully drop the broken packages and process only the correct ones. The send delay is configured in bytes, which equals the amount of garbage, as seen in Figure 2. Various traffic patterns like exponential distributed bursts are easily possible with that approach.

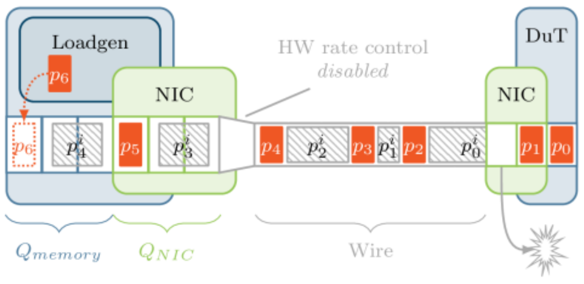

Figure 2: MoonGen rate control [2]

A MoonGen packet buffer can be set a waiting gap duration by `buf:setDelay(bytes)`. On 10GbE one byte would have a delay of 0.8 nanoseconds. A randomly exponentially-distributed delay can be generated with the poisson process by `poissonDelay(average_wait)`, where the parameter specifies the average wait time between two packets. The exponentially-distributed wait time can directly be fed into `setDelay(poissonDelay(..))`, this will then be the amount of garbage sent out between real packets.

```
function send(txDev)
    local mem = ...
    local bufs = mem:bufArray()
    while dpdk.running() do
        bufs:alloc(size)
        for _, buf in ipairs(bufs) do
            local avg = rateToByteDelay(rate, size)
            local delay = poissonDelay(avg)
            buf:setDelay(delay)
        end
        queue:sendWithDelay(bufs)
    end
end
```

5. CONCLUSION

This introduction should have prepared you to achieve any measurement task in MoonGen. Complete and directly executable examples are shipped with MoonGen in its `examples/` subfolder. This tutorial should have prepared you to implement your particular test setup and understand complex examples like `router.lua`. With namespace, you can try implementing an `ARP` lookup task that figures out peer addresses for your packet crafting.

If you encounter any issue while using MoonGen and think it's not your or your setup's fault, please create an issue at https://github.com/emmericp/MoonGen.git/issues so the developers can assist you or fix the bug you may have discovered. We hope you enjoy using this tool and encourage you to improve it further and adapt it to your needs, as it's free software for a reason.

References

[1] Data Plane Development Kit. http://dpdk.org/. Accessed: 2015-12-05.

[2] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, October 2015.

[3] Linux network stack scaling. https://www.kernel.org/doc/Documentation/networking/scaling.txt. Accessed: 2015-12-19.

[4] Lua tutorial. https://learnxinyminutes.com/docs/lua/. Accessed: 2015-12-19.

[5] MoonGen repository. https://github.com/emmericp/moongen. Accessed: 2015-12-10.

[6] Mike Pall. Luajit. http://luajit.org/. Accessed: 2015-12-05.

Middlebox Models in Network Verification Research

Julius Michaelis
Advisor: Cornelius Diekmann
Seminar Innovative Internettechnologien WS2015
Chair for Network Architectures and Services
Fakultät für Informatik, Technische Universität München
Email: michaeli@in.tum.de

ABSTRACT

To address the challenges arising from the development of computer network management over the past decades, researchers have developed a number of tools to assist the operation of networks and help administrators avoid mistakes. These tools often follow the approach to verify an existing network configuration. This poses the problem that the behavior of a lot of potentially complex networking device configration has to be supported. The usual approach to this is to develop simple models that only reflect the aspects of the system that the tool can understand. We survey the related literature for the use of this type of model.

Keywords

Computer Networks, Formal Models, Network Verification

1. INTRODUCTION

Over the past decades, computer networks have grown considerably in size and complexity. Attempts to fulfil the resulting complicated service requirements has given rise to more and more complicated middleboxes (switches, routers, firewalls, etc...). Configuring these middlboxes poses an enormous challenge to network operators, who have to be able to understand numerous configuration languages and manage interoperating distributed configuration. While this earned network operators the title "masters of complexity" [24], it is generally seen as problematic.

Computer networks researchers have recognized that other fields, e.g., programming languages, have developed high level approaches to mitigate complexity and present users with simple ways of detecting and avoiding errors. In the past decade, research transferring these approaches to networking has gained traction. A number of tools have been developed that can be invoked to analyze an existing network configuration. The usual approach of these tools is to have the user collect the configuration of his network on a single machine. This configuration is complex and can thus not be directly and likely not fully understood. A tool will parse the configuration and translate it into a representation it can reason about: a model. Depending on the type of the tool, different reasoning is possible; a very common application is to find all possible routing loops — or, if none are found, to prove the absence. However, the development of such tools holds the same challenges that network management holds: a lot of diverse configuration languages and devices has to be supported. *Software Defined Networking* (SDN) attempts to alleviate this burden for operators by proposing a central, programmable controller. This controller is connected to all network devices and can configure them. The devices can notify the controller when they receive a certain type of packet, e.g., packets belonging to a new connection. The controller hands these notifications to a program written by the user. This program can then configure the devices accordingly, e.g., create a path for the new connection. SDN allows to manage nearly all configuration in a single language and logically on a single host.[1] This greatly simplifies the operators tasks and can simplify the verification of the configuration by automated tools.

The crucial steps when attempting to verify a configuration are defining a model of how the configuration is going to be understood, and how to translate real configuration into a representation in the model. This holds various challenges, some of them are intrinsic to modeling: if the model is too simple, it may not be able to represent reality. If it is too complex, it may lose its purpose, as reasoning about it becomes as complicated as reasoning about the original objects. Other challenges are specific to the problem at hand: no tool can possibly understand all the different configuration languages. Typical tools, such as Anteater [18], Hassel [16], VeriCon [5], or Exodus [23] are able to understand subsets of a few configuration languages, such as Cisco IOS or Juniper configurations. This is usually accompanied by the claim that other configuration languages can be easily translated in the same manner, without giving further considerations to the subtleties of such a translation.

In this paper, we survey the related work for the use of models. We focus on the analysis of how systems are modelled.

The rest of this paper is organized into two parts: Section 2 contains the survey of the different models and what aspects they model. We have grouped the models into subsections by the type of the device that is modelled. Section 3 contains a short overview of what purpose the models serve and whether they could be repurposed.

2. BOX MODELS

We continue with the different box models. Section 2.1 surveys link layer switches, Section 2.2 routers, Section 2.3 SDN switches, and Section 2.4 Firewalls. Section 2.5 looks at models of devices that do not only provide a single function. Additionally, Section 2.6 shows the *big switch model.*

[1]Note that this reflects the understanding of SDN from the view of OpenFlow [20], variants exist.

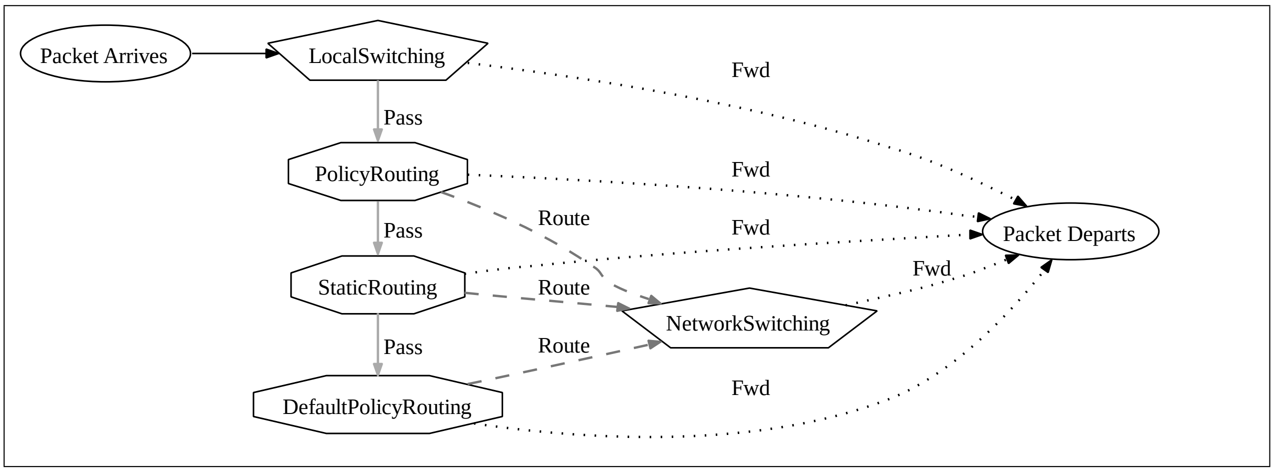

Figure 1: Margrave's router model, taken from [22].

2.1 Link Layer Switches

When considering only the basic switching functionality, Link layer switches become relatively simple devices. They neither offer many opportunities for modeling, nor are they very interesting as a target for verification, since most switches simply have no configuration to verify. Accordingly, there is not much material in the related literature.

There is one problem that arises when verifying networks that contain switches among other devices. Switches are stateful devices, while some verification systems do not support state. A simple modeling solution for that is presented in Header Space Analysis [16]:

> When we generated box transfer functions, we chose not to include learned MAC address of end hosts. This allowed us to unearth problems that can be masked by learned MAC addresses but may surface when learned entries expire.

While this modelling decision has consequences for the models of a variety of devices, it implies that switches are always in their learning phase, i.e. are effectively replaced by broadcast devices.

2.2 Routers

For this section, we will focus solely on layer 3 forwarding.

While Margrave [22] is originally a tool for the analysis of firewalls, it also has a detailed understanding of the packet forwarding process in Cisco IOS to be able to accurately perform its tasks. The model that Margrave uses is shown in Figure 1. The process of forwarding is described as follows. First, packets destined to locally attached subnets are filtered out and directly forwarded. All other packets are subjected to routing. The second step is thus to handle policy routing — packets with special routing rules that do not only depend on the destination address but e.g., on the source address, too. The third step is to consider statically configured routes. All remaining packets are processed using the default policy.

A more abstract model of routing is presented by Xie *et al.* [25].[2] They model a network of routers to be an annotated graph $(V, E, \mathcal{F})$ where the nodes V represent the routers, E contains two directed edges for each physical link, and the edge labels $F_{u,v} \in \mathcal{F}$ express which packets are allowed to flow over an edge u, v. The routing process is modelled using $\mathcal{F}$: a flow over an edge u, v will only be permitted by $F_{u,v}$ if the router u has a route to v for that specific flow.

To summarize, the Margrave's model [22] describes the routing process while the model in [25] abstracts it away to be a property of a graph.

2.3 SDN Switches

This section surveys selected works on switches in software defined networking (SDN). Various approaches to SDN exist. This section focuses on OpenFlow [20] since it is currently the most actively researched variant. We assume that the reader is familiar with its basics. While it could be said that the OpenFlow switch specification [3] itself is based on a model of a generic networking device, we are not going to explore this and instead examine models of OpenFlow switches. We will continue to denote OpenFlow switches as switches in this section for succinctness. The term *datapath element* would be more accurate since the switches can take arbitrary functions.

Guha *et al.* [14] present a fully machine-verified implementation of a compiler for the NetCore controller programming language. For this purpose, they give a detailed model of an OpenFlow switch that adheres closely to version 1.0 of the OpenFlow switch specification [2], including packet processing and switch-controller interaction. We will examine the most important details and begin with the flow table evaluation semantics: Guha *et al.* dedicate significant attention to how a packet is matched against a flow table entry. Their main concern there is related to behavior that was only made explicit in later versions of the specification, e.g. [3, §7.2.3.6]:

[2]The Anteater tool [18] mentioned in Section 1 is based on this work.

$$\frac{\begin{array}{c}\exists(n, pat, \{\!|pt_1 \cdots pt_n|\!\}) \in FT.\\ pk\#pat = \textbf{true}\\ \forall(n', pat', pts') \in FT.\ n' > n \Rightarrow\\ pk\#pat' = \textbf{false}\end{array}}{[\![FT]\!]\ pt\ pk \rightsquigarrow (\{\!|(pt_1)\cdots(pt_n)|\!\}, \{\!||\!\})}\ (\textsc{Matched})$$

$$\frac{\forall(n, pat, pts) \in FT \qquad pk\#pat = \textbf{false}}{[\![FT]\!]\ pt\ pk \rightsquigarrow (\{\!||\!\}, \{\!|(pt, pk)|\!\})}\ (\textsc{Unmatched})$$

Figure 2: Flow table semantics by Guha *et al.*, taken from [14].

> The presence of an [OpenFlow match] with a given [type] may be restricted based on the presence or values of other [matches], its prerequisites. Matching header fields of a protocol can only be done if the OpenFlow match explicitly matches the corresponding protocol.

For example, to match an outgoing SSH connection, a match must check for at least layer 4 destination port 22, layer 4 protocol TCP, and layer 3 protocol IP. If only the match for layer 4 destination port is included, some implementations of an OpenFlow switch return an error as required by the specification [3, §7.5.4.3]. Others, including the reference implementation [1], silently drop them, which has led to several severe bugs, according to [14]. Guha *et al.* specify their packet matching semantics to only evaluate matches when a previously executed match on the preconditions has assured that the necessary header fields are present.

Next, they specify a flow table to be a multiset of triples of priority, a match condition, and a multiset of output ports. Although a multiset allows for uncountably many flow table entries instead of a bounded number thereof, the implications for the validity of the model are minimal. The semantics $[\![FT]\!]\ pt\ pk \rightsquigarrow (o, c)$ for evaluating such a table is shown in Figure 2. The semantics describes the decision for a flow table FT and a packet pt arriving on a port p. It can specifiy to forward the packets on the port set o or to send the messages c to the controller. The operator $\#$ matches a packet against a rule. Note that this semantics is nondeterministic: if there are multiple matching flow table entries with the same priority, it can be said that all of their actions are executed nondeterministically. This is used to model the fact that the specification [2, §3.4] says that the switch is free to choose any order between overlapping flow entries. For this paper, we have verified that determinism can be enforced by adding the following precondition on the flow table:

$$\forall(n, pat, pts) \in FT.\ \forall\big(n', pat', pts'\big) \in FT \setminus \{(n, pat, pts)\}.$$
$$n = n' \implies \nexists pk.\ pk\#pat \wedge pk\#pat', \qquad (1)$$

i.e. for two rules with the same priority, no packet matches both. Note that this is slightly stronger than necessary to make the semantics deterministic: overlapping entries could be shadowed by a rule with higher priority.

Guha *et al.* also specify a semantics for the message processing and passing between switches and controllers. They model it as an inductively defined relation on the states of switches, controller(s) and links between them. The semantics of this is comparatively large: its 12 rules span an entire page. There is one important modelling detail that can be singled out: Switches are modelled as a tuple of their unique identifier, their ports, one flow table, and four message queues, one for each combination of in/out and controller/switch to switch. These message queues are multisets. On the receipt of a message through a link, or when obtaining a message through processing at a switch, the message is first enqueued in one of these queues. The semantics is non-deterministic and allows to accumulate arbitrary many messages and dequeue them in an arbitrary order. This models the option for switches to reorder messages. The only exception is a *BarrierRequest*, which is never enqueued but, given that the input queue is empty, directly processed. It can thus be used to ensure that all messages have been sent before it is processed.

Orthogonal to the work of Guha *et al.* stands VeriCon [5]. It is not a verified compiler for controller programs but a verifying tool for controller programs. It does not have a detailed model of single switches against which it verifies the output of its compiler. Instead, it checks its result on a high-level model of a network of switches. Its authors, Ball *et al.*, begin by presenting a simple example programming language for controllers, called CSDN, and give a formal semantics for this language. VeriCon allows to prove correctness of programs within these semantics but it does not establish the correctness of the compiler. VeriCon takes three inputs: a CSDN program, a topology invariant, and a correctness condition. The topology invariant allows to limit the possible changes in topology, e.g., the user can define that they will always ensure that no path in the network has more than 3 hops. The correctness condition is then verified to hold for all possible states and topology changes. To achieve that, VeriCon uses the following high-level model of a network of switches: Its state is modelled as 5 relations. The first relation contains the links between switches, or switches and hosts, the second one all paths that are possible over those links. These two relations are mainly used to formulate topology invariants. The third relation $S.\mathit{ft}\,(\mathit{Src} \to \mathit{Dst}, I \to O)$ records whether switch S has a rule in its forwarding table to forward packets from host Src to Dst from input port I to output port O. Similar to that $S.\mathit{send}\,(\mathit{Src} \to \mathit{Dst}, I \to O)$ records whether such a packet has actually been sent. Lastly, the relation $S.\mathit{rcv}^{\mathit{this}}\,(\mathit{Src} \to \mathit{Dst}, I)$ models whether a packet has been received at input port I. With these, given a desired post-condition Q, VeriCon can compute the weakest precondition $\mathit{wp}[\![c]\!](Q)$ for executing a command c in CSDN. For example:

$$\mathit{wp}\,[\![\mathit{pktIn}\,(s, p, i) \Rightarrow c]\!]\,(Q) :=$$
$$\Big(s.\mathit{rcv}^{\mathit{this}}\,(p, i) \wedge s.\mathit{ft}\,(p, i \to o)\Big) \implies \mathit{wp}[\![c]\!]\,(Q)\,. \qquad (2)$$

This is the precondition semantics for the event handler specification $\mathit{pktIn}(s, p, i) \Rightarrow c$. In the event of receiving a packet p at port i of switch s, c is executed. The semantics expresses the following: given that such a packet is actually received and a forwarding rule is installed, the handle has to satisfy the weakest precondition of its command.

Similar to VeriCon is NICE [8]. It uses model checking and other techniques to verify the correctness of a controller program at runtime. It models an SDN as a system of stateful "components" that communicate in a first-in first-out manner. Communication between the components is, among other things, modelled by state transitions in the system. The controller programs are modelled accordingly: as a set of event handlers that trigger state changes in the controller. For model checking, NICE executes these handlers to explore the state space and see if any of the transitions can violate correctness invariants.

NICE notes that, for model checking, it would also need to explore the state space of the switches. Since even the reference implementation Open vSwitch [1] has multiple hundred KB of state when executed, this is not directly feasible. NICE thus presents a simple model of a switch with a reduced amount of state. A switch is modelled as a set of communicating channels, two state transitions and a single flow table. Except for the control channel, which operates strictly in a first-in first-out manner, these channels may also drop or reorder messages[3]. On the receipt of at least one message, a state transition is executed. To reduce the amount of state transitions necessary, all packets present in a state are modelled as being processed as a single transition. NICE also makes an important remark on the flow table: two flow tables can be syntactically different, i.e. have a different entry structure, but be semantically equivalent, i.e. lead to the same forwarding decisions. This observation is true for all three models here. For example, a table that contains only exact flow matches (flow entries without any wildcards) makes decisions independent of the priority of the rules (i.e. the order in which they are considered)[4]. NICE uses heuristics to merge semantically equivalent states.

2.4 Firewalls

The term firewall is used for a diverse variety of devices and software. Devices by different vendors, such as Cisco, Sun Microsystems, or Sophos have a largely different set of features and purposes. Even the Linux kernel has two different firewall implementations (iptables and nftables). This means that a large number of different models exists. Nevertheless, a common principle can be factored out: most firewalls and all models considered here consist of rules, which in turn consist of at least a match and an action. The match decides whether the action is to be applied to a given packet. The firewalls' types differ in how rules are organized, i.e. in which order they are applied, and what kind of match expressions and actions are supported. Another detail of interest is how connection state tracking is modelled, i.e. how packets that belong to established connections are treated differently from packets for new connections. Many real world firewalls begin by accepting packets that belong to or are related to an established connection. Finding a simple but powerful model for state is hence important.

Accompanying a model of a firewall, there always has to be a model of packets on which the firewall operates, albeit this is often left implicit. One of the few works that explicitly specifies the packet model is [6]:

$$(\alpha, \beta)\, \mathit{packet} \coloneqq (\mathit{id} \times \mathit{protocol} \times \alpha\, \mathit{src} \times \alpha\, \mathit{dest} \times \beta\, \mathit{content})\,. \tag{3}$$

This can be read as: given arbitrary types α and β, a packet consists of a record of a unique identifier, the used protocol (http, ftp, ...), a source and destination address of type α and packet content of type β.[5] It is obvious that this format does not model real packets very closely, since neither the used (application layer) protocol is usually stated directly, nor is every connection associated with a unique ID. Nevertheless, the ID hints to how state is modelled by Brucker et Wolff in [6]. They model state by allowing the match to consider a list of all packets that the firewall has accepted so far. The ID can be used to determine if a packet is the first of its connection.

A less complicated model of state, which is also based on the packet format, can be found in ITVal [19] (however, not in a strongly formal manner). Whether a packet is part of an established connection is simply treated to be another packet field. The theory files accompanying [12] contain a proof that that model is not weaker than querying an internal state table when performing a stateful match.

The packet model is often tightly tied to which types of match expressions the firewall supports. A very common subset that can be found in many real firewalls and models is to support equality matches on (OSI) layer 4 protocol, source, destination ("ports"), the physical ingress port and additionally prefix matches on the layer 3 addresses. Some models extend this by fields for TCP flags [19, 26], or connection state [6, 22].

Besides the set of supported match expressions, firewalls and models also differ in how these expressions can be combined and how these combinations are represented. Margrave [22] supports conjunctions of disjunctions, i.e. it allows to specify several possible values for one field and allows combining fields while requiring all of them to match. *Iptables Semantics* [12] by Diekmann *et al.* allows for more complicated expressions: given *match* is a match on a single field, it supports the following match expressions *mexpr*:

$$\mathit{mexpr} \coloneqq \mathit{match} \mid \neg\mathit{mexpr} \mid \mathit{mexpr} \wedge \mathit{mexpr} \mid \texttt{True} \tag{4}$$

This model is a superset of what iptables supports: iptables supports negation of matches only on the lowest level, i.e. it only supports constructing $\neg\mathit{match}$ but not $\neg\mathit{mexpr}$. This is an example (but not the only one) of a model that could have been easily made to mirror a system more closely but instead was made more powerful. In this case, *mexpr* allows to express arbitrary boolean functions, which can be used to compute the expression for packets that are not matched by a rule.

Packet and match models that are tailored to be suitable for the implementation of an analysis can be found in FIREMAN [26] and ITVal [19]. FIREMAN models a packet as a

[3]Note the difference to [14] where the control channel does also not operate in a first-in first-out manner and can reorder messages.

[4]Assuming that the switch does not accept overlapping rules.

[5]Brucker et Wolff later specify α to be a four-tuple of integers to represent the IP-address in dotted-decimal notation and a port, also represented by an integer (i.e. a number from $\mathbb{Z}$).

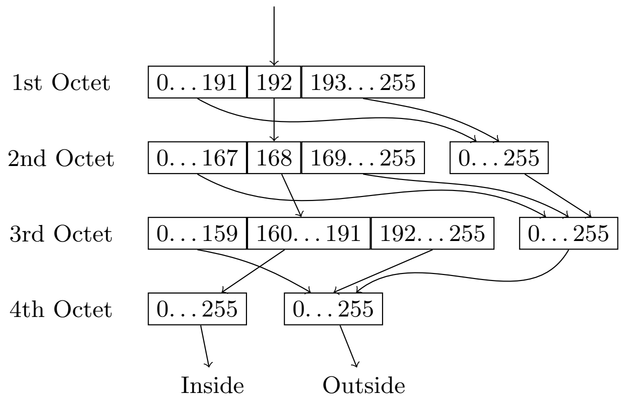

Figure 3: Example of how ITVal [19] would represent a match for 192.168.160.0/19, given that the packet consists only of a single IP address to match on.

vector of bits that represent its header. Match expressions are boolean expressions that can be efficiently represented by *Reduced Ordered Binary Decision Diagrams (ROBDD)* [7]. ITVal [19] extends this to *MDDs* [17], a structure that is similar to a *ROBDD* but allows continuous values for its variables. Consequently, ITVal models packets as the vector of bytes that represent source and destination for IP address and layer 4 port. Additionally, it keeps separate fields for the layer 4 protocol type, the TCP flags and the connection states. Each byte and field is then represented by one level in the MDD. Figure 3 shows an example of how an MDD is used to match a single IP prefix. The purpose of this subdivision of the packet header is to create a balance between too many levels and too much information on a single level of the MDD.

After considering the match of a rule, the action has to be modelled. The common subset of actions that can be found in all models we analyzed is to either let packets pass the firewall or to stop them. Iptables supports a number of other actions that are directly executed, such as `LOG`, which will generate debug output but have no effect on forwarding, or `REJECT`, which will stop the packet and additionally send an error message. The semantics by Diekmann *et al.* [12] shows a way to translate action types with behavior unkown to the system back to only forwarding or discarding the packet. The model of actions given by Brucker *et* Wolff [6], allows for something more complicated: the action returns a packet. This allows to model packet modification by firewall rules.

The last important property of a firewall model is how rules are combined to form the firewall. Most models can be categorized to either use what Yuan *et al.* [26] call the *simple list model* (used e.g. in [22]) and the *complex chain model* (used e.g. in [19]). The list model states that the firewall rules are written as one list that is traversed linearly. Each rule either applies and the execution terminates or the execution continues with the next rule. The chain model extends this by allowing for multiple lists and the possibility to conditionally jump to the start of such a list and to conditionally return to the origin of the jump. Diekmann *et al.* [12] formalized both and present a translation from the chain model to the list model.

If a firewall is an actual networking device, it also needs to decide on which port to forward packets, i.e. become a switch or router additionally to its firewall function. This type of combined functionality is considered in the next section.

2.5 Complex devices

Real network devices often fulfil more than one of the functions described above. A common example of this is a Cisco IOS router, which is usually configured with both an ACL (i.e. its firewall function) and routing information.

An important insight is that these functions usually have very little or no relevant shared state. The key implication of this is that the different stages of such a system can be analyzed separately and then *pipelined* together. In the example of the IOS router, this would mean to first analyze the incoming ACL, then the routing configuration and then the outgoing ACL.

Dobrescu and Argyraki [13] have realized that this holds true even for controller software that is written for SDN switches. When attempting verification of software in general, one has to deal with the path explosion problem. By dividing a network system into m independent elements with maximally n branches each, pipelined analysis can reduce the amount of paths that has to be analyzed exponentially from $\mathcal{O}(2^{mn})$ to $\mathcal{O}(m2^n)$. Combined with further optimization for relevant data structures and symbolic computation, their tool *ClickVerifier* is able to verify controller programs. Dobrescu and Argyraki make an explicit point of using the pipeline model only to explain why their system has the desired performance. For the verification, the actual generated controller program bytecode is passed to the analysis engine S^2E [10] to avoid any abstraction errors that might happen when modeling OpenFlow switches.

Another interesting instance of the pipelining model can be found in the tool Margrave by Nelson *et al.* [22]. A schematic representation of how it is used to decompose a Cisco IOS configuration can be found in Figure 4. Each element of the pipeline is used to provide the further elements of the pipeline with necessary information to continue the analysis, e.g., the *Internal Routing* step is used to decide which outbound NAT and ACLs apply.

A very similar approach to this is taken in Hassel, the tool that implements Header Space Analysis by Kazemian *et al.* [16]. Hassel translates dumped Cisco IOS configurations into functions represented in a model on which symbolic computation is possible. Representing the full function of a router with a single function of this model would result in very large representations. The transfer through one Cisco IOS router is thus modelled as traversing three layers in the model, one input ACL and VLAN untagging step, one routing step, and one output processing step. The difference to how Margrave uses the pipeline model is that Header Space Analysis reuses the exact same model for each step.

This solution of size reduction is also applied by Exodus [23]. It translates Cisco IOS configurations into controller config-

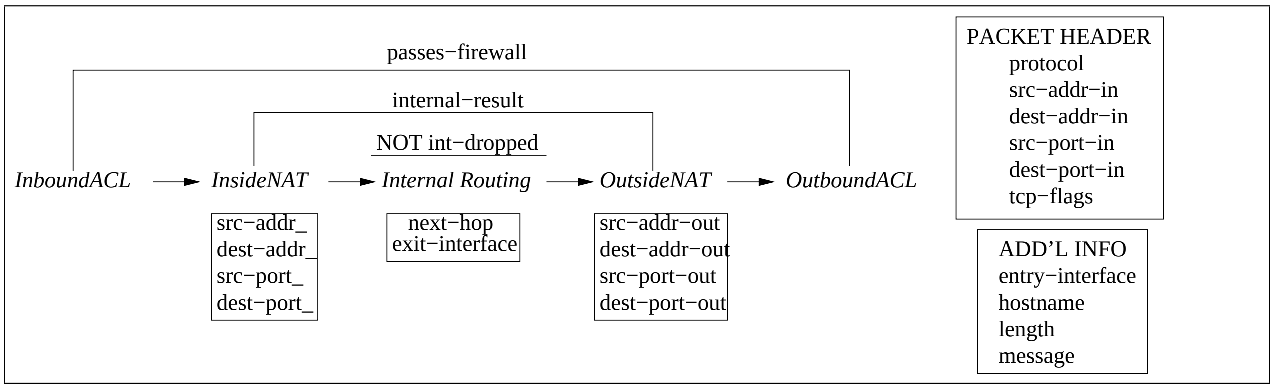

Figure 4: Margrave's decomposition of IOS configurations, taken from [22]

uration for an OpenFlow switch (*cf.* Section 2.3). Much of the available OpenFlow capable hardware only supports a single table of matches and output ports. Similar to the functions of Hassel, compressing the entire functionality of a Cisco switch into a single table would create very large representations through the use of cross products. Exodus thus uses multiple switches (i.e. physically multple devices). Their pipeline steps are, in order, VLAN untagging, input ACLs, routing, NAT, routing (2), layer 2 rewriting, output ACLs, and VLAN tagging.

2.6 Big Switch Model

While the *big switch model* is not strictly speaking a model of a networking device but a model of the entire network, we feel that it is important enough to mention it here. It is used in various places, first and foremost in SDN [9, 15, 21] programming languages, but also e.g., for network verification [4]. The general idea is that a network or subnetwork of switches, routers, firewalls displays a forwarding behavior to all attached devices that are not part of the subnetwork. The subnetwork usually has a complicated distributed configuration that defines its forwarding behavior. Even in SDN programming languages, this state is often exposed to the programmer. Proponents of the big switch model usually attempt to represent this state as if it was the configuration of a single switch with each of its ports representing one connection from the subnetwork to something outside of it. The rationale for this is that a representation for the configuration of the big switch could be smaller and thus easier to understand or verify.

Anderson *et al.* [4] propose a slightly different interpretation of the big switch model: for a network to be an instance of the big switch model, they require that the network displays the behavior of a big learning switch, i.e. implements all-pairs reachability. Since NetKAT [4] allows testing the equality of two network descriptions, they formulate a formal condition for this to be true. Showing this condition here would require explaining the formalism used by NetKAT and is thus out of scope.

3. COMPARISONS

In the previous section, we surveyed for existing models and what they express. This section gives an overview of what type of model they are, i.e. how they are used and integrated in their environment. The results of this section are summarized in Table 1. For the table, we differentiated between two types of model usage: are the models used in a *proof* that establishes correctness properties of the system, or are they used when *implementing* a surrounding system. Additionally, we checked if some kind of formalization was available and ready for reuse. Some of the attributions are not entirely clear. We will explain them in the following paragraphs.

Work		Proof	Implementation	Reusable
NetKAT	[4]	✔	✔	✔
VeriCon	[5]	✔	✔	✘
HOL-TestGen	[6]	✔	✔	✔
NICE	[8]	✘	✔	✘
Iptables Semantics	[12]	✔	✔	✔
(Dobrescu and Argyraki)	[13]	✘	✘	✘
(Guha *et al.*)	[14]	✔	✔	✔
HSA / Hassel	[16]	✘	✔	✘
ITVal	[19]	✘	✔	✘
Margrave	[22]	✘	✔	✔
Exodus	[23]	✘	✔	✘
(Xie *et al.*)	[25]	✘	✘	✔
FIREMAN	[26]	✘	✔	✘

Table 1: Usage of models in the surveyed work

Most of the models considered here fall into one of two usage categories. Members of the first category [8, 16, 19, 22, 23, 26] propose a model of networks or networking devices that simplifies reality significantly. This model is then used to justify and explain the steps that have been taken in the implementation of an accompanying tool. The level of formality of these models varies. Also, the models are usually tightly tied to the implementation and not suitable for reuse in other projects. Margrave is a notable exception to this as its model is a relatively generic representation of Cisco IOS configuration.

Members of the second category [6, 12, 14] give a model that has a high degree of formality and is accompanied by a semantics that aims to closely mirror the behavior of the real system. This semantics of all of the models from the second category is available in form of code for theorem proving software.

Some models could not be sorted into either of these categories.

- VeriCon [5] explicitly formalizes the model it uses and includes a semantics. However, its semantics does not mirror the behavior of any real device. Nevertheless, the use of the *Satisfiable Modulo Theories* solver Z3 [11] does provide proof that the programs checked with VeriCon are indeed correct wrt. the semantics. VeriCon's authors also claimed that the semantics would be made available online. To this date, it is marked as "pending".
- Similar to VeriCon's case is NetKAT [4], except the formalization is available.
- While Dobrescu and Argyraki [13] do propose a model, they do not use it in their implementation. This is, as mentioned, to avoid carrying any discrepancies between the model and reality into the implementation. The model is merely used to justify why the implementation can terminate quickly.
- The work by Xie *et al.* [25] gives a model of a network of routers but does not implement anything based on it. An implementation that is based on that network model has later been given with Anteater [18] by different authors. As such, the model by Xie *et al.* proved to be (re-)usable even though there is no readily available formalization of it other than the publication itself.

4. CONCLUSION

We surveyed related work for the use of models of networking boxes. We included models of learning switches, routers, OpenFlow switches, firewalls, and devices that include multiple of these functions. This work can provide a reference for further works in the area that want to use strong formalism and thus have to use models of networking boxes. It can answer the questions of which models have already been constructed, how they are used, and how their properties make them qualified for the specific use-case.

5. REFERENCES

[1] Open vSwitch. `http://openvswitch.org/`.

[2] OpenFlow Switch Specification v1.0.0, December 2009.

[3] OpenFlow Switch Specification v1.5.1, March 2015.

[4] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, 2014.

[5] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. VeriCon: Towards verifying controller programs in software-defined networks. In *ACM SIGPLAN Notices*, volume 49, pages 282–293. ACM, 2014.

[6] A. D. Brucker and B. Wolff. Test-Sequence Generation with HOL-TestGen with an Application to Firewall Testing. In *Tests and Proofs*, pages 149–168. Springer, 2007.

[7] R. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, C-35(8):677–691, August 1986.

[8] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, et al. A NICE way to test OpenFlow applications. In *NSDI*, volume 12, pages 127–140, 2012.

[9] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the Network Forwarding Plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[10] V. Chipounov, V. Georgescu, C. Zamfir, and G. Candea. Selective symbolic execution. In *Workshop on Hot Topics in Dependable Systems*. Citeseer, 2009.

[11] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin Heidelberg, 2008.

[12] C. Diekmann, L. Hupel, and G. Carle. Semantics-Preserving Simplification of Real-World Firewall Rule Sets. In N. Bjørner and F. de Boer, editors, *FM 2015: Formal Methods*, volume 9109 of *Lecture Notes in Computer Science*, pages 195–212. Springer International Publishing, 2015.

[13] M. Dobrescu and K. Argyraki. Software dataplane verification. In *Proceedings of the 11th Symposium on Networked Systems Design and Implementation (NSDI), Seattle, WA*, 2014.

[14] A. Guha, M. Reitblatt, and N. Foster. Machine-verified Network Controllers. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 483–494, New York, NY, USA, 2013. ACM.

[15] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "One Big Switch" Abstraction in Software-defined Networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 13–24, New York, NY, USA, 2013. ACM.

[16] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 113–126, San Jose, CA, 2012. USENIX.

[17] H.-T. Liaw and C.-S. Lin. On the OBDD-representation of general Boolean functions. *IEEE Transactions on computers*, (6):661–664, 1992.

[18] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, and S. T. King. Debugging the data plane with anteater. *ACM SIGCOMM Computer Communication Review*, 41(4):290–301, 2011.

[19] R. M. Marmorstein and P. Kearns. A Tool for Automated iptables Firewall Analysis. In *Usenix*

annual technical conference, Freenix Track, pages 71–81, 2005.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[21] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing Software Defined Networks. In *Networked Systems Design and Implementation*, pages 1–13, 2013.

[22] T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. The Margrave Tool for Firewall Analysis. In *Proceedings of the Large Installation System Administration Conference*, 2010.

[23] T. Nelson, A. D. Ferguson, D. Yu, R. Fonseca, and S. Krishnamurthi. Exodus: toward automatic migration of enterprise network configurations to SDNs. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 13. ACM, 2015.

[24] S. Shenker, M. Casado, T. Koponen, and N. McKeown. The future of networking, and the past of protocols. Talk at Open Networking Summit, 2011.

[25] G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2170–2183 vol. 3, March 2005.

[26] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. Fireman: a toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.–213, May 2006.

Topology Discovery in controlled environments

Maximilian Pudelko
Betreuer: Florian Wohlfart, Sebastian Gallenmüller
Seminar Future Internet WS2015
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: maximilian.pudelko@tum.de

ABSTRACT

This paper describes a method to collect data about the topology of networks. Further on the data is used to generate images representing the links between the hosts giving a broader overview over the network. The single steps of the process are fully automated to reduce manual interaction with the system.

Keywords

topology discovery, test environment, lldp

1. INTRODUCTION AND SCENARIO

A research networking testbed can be a rapidly changing environment as some experiments require different setups. This leads to a constantly changing wiring and positioning of the single testbeds. These changes done must be documented manually and merged into the existing documentation. As this is seen as a cumbersome process it is often skipped leading to an outdated information base. It is the goal of this paper to develop a program which aids this documentation process by utilizing existing technologies and software to detect the physical topology of the testbed. With this goal set the program has to meet the following requirements:

- The detected topology should be a correct representation of the hosts and their interconnection at the link layer of the network. Detailed information about speeds and type, in case of links, and hardware configuration in case of hosts should be collected.
- Usage of the tool should result in reduced human involvement compared to collecting the data manually.
- The output must be graphical in a way that presents the topology suitably.
- The program should integrate into the existing testbed setup. Specifically it must not interfere with other running experiments.

The reminder of this paper is organized as follows: Section 2 describes existing technologies and how they apply here, Section 3 illustrates the developed solution of which the results on the Baltikum testbed are presented in Section 4.

2. RELATED WORK

As the idea of developing a method to discover present devices and their physical topology on a Local area network, IEEE 802 (LAN) is not new, several solutions have been already developed by different parties and organisations. The following chapter will give a short overview over some of these protocols, compares them and explains why Link Layer Discovery Protocol (LLDP) was chosen as the base to solve the problem at hand. Additionally the basic operation of LLDP will be clarified as needed for the understanding of the solution as it builds on it.

2.1 Proprietary Protocols

Introduced in 1994 the Cisco Discovery Protocol (CDP) aims to provide a mechanism to discover devices connected to a network at the Link Layer (L2) by broadcasting information about itself. This enables any device wanting to collect information to simply listen for this broadcasts without any prior configuration [3]. As the usefulness of such a tool for network administration became evident, it was included in the OS of Cisco's networking hardware and over the time developed further to e.g. set-up VoIP telephones.
Similar the Link Layer Topology Discovery Protocol (LLTD) was developed by Microsoft which too operates on any IEEE 802 network [4]. Contrary to the CDP it was not included in networking hardware, but in the network stack of Microsoft's OS starting with Windows Vista to display a graphical Network Map of home networks and to detect connectivity problems of wireless networks.
While for both protocols free Linux implementations exist, neither of them are a viable solution as they are either limited to vendor specific hardware, which would result in incomplete topologies in case of mixed setups, or require the signing of a license agreement to use them.

2.2 Link Layer Discovery Protocol

In 2005 a IEEE task force created a vendor neutral protocol called LLDP to unify the up to then incompatible vendor specific protocols. In the following the basic operation will be explained.

Like its predecessors LLDP operates on the Link-Layer of LANs, which gives it the benefit of low configuration prerequisites. In particular no IP addresses are needed as communication happens via Ethernet with vendor-set or manually administered MAC addresses. A participant, referred to as chassis, may be active and/or passive depending on configuration. An active chassis will broadcast LLDP frames on all of its ports in regular intervals to inform potential listeners of its presence. If a passive chassis retrieves this frame it will store this information in a local Management

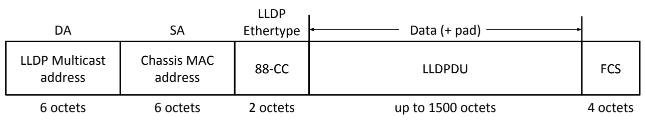

Figure 1: IEEE 802.3 LLDP frame format[2]

Information Base (MIB). While two hosts may be able to reach each other, no direct communication happens between them. Particularly it's not possible to make requests to other chassis asking for the content of their databases, as LLDP is designed as a one way protocol [2] using only broadcasted and not directly addresses frames frames. As LLDP operates on L2 every frame has to be wrapped in a Ethernet frame as shown in figure 1. Since these frames are not directed to one specific host a broadcast address has to be used in the destination address field. As normal broadcasted frames with target FF-FF-FF-FF-FF would get forwarded by bridges and switches, which would lead to an inclusion of equipment beyond the physical link of a chassis, the standard suggests a set of three reserved addresses [1] to limit frames to one link on conforming switches. The source address is filled out with the MAC of the interface (port) on which the frame goes out. To distinguish LLDP frames from other protocols like IP a separate Ethertype of 88-CC is used. Next follows the actual payload in form of the LLDP Data Unit (LLDPDU). Each LLDPDU consists of a concatenation of TLVs of which some are mandatory and some optional. The Chassis ID TLV contains the identifier of the chassis that send out the frame and has to remain constant for the time of operation of the LLDP service. This property is important as it enables us to compile a list of the available hardware in the network as explained in 3.2. The likewise mandatory Port ID TLV uniquely identifies the sending port of a chassis. With this value it becomes possible to distinguish between multiple links between two chassis or to determine the exact port number on a connected switch. Among the optional ones the organizationally specific [2] System Capabilities TLV and the MAC/PHY Configuration/Status TLV are of special interest, as they indicate the type of networking device (router, switch or simple station) and the bandwidth/type of the physical link respectively.

As this standard has been implemented by all major networking hardware manufacturer and is usable without restrictions, it is chosen as the protocol to use in the Baltikum testbed. To enable this functionality on the Linux hosts the FOSS program lldpd is employed, which additionally also implements a client for most vendor specific protocols [1] and thus ensures maximum coverage.

2.3 Other, higher layer protocols

While several other protocols like Open Shortest Path First, a IP link-state routing protocol (OSPF) and Neighbor Discover Protocol, part of IPv6 (NDP), which operate in a similar domain, exist, they are not applicable to the scenario of a testbed because of the layer they operate on. The requirement of getting a link layer topology can not be satisfied with protocols operating at network layer, as they, by design, abstract single links away. Also is the required knowledge and organization to set them up often too much of an overhead or not even possible at all.

[1]01-80-C2-00-00-0E, 01-80-C2-00-00-03 and 01-80-C2-00-00-00. Each with different meanings to provide further control
[2]Vendors and organizations can define custom TLVs and apply for inclusion into the standard

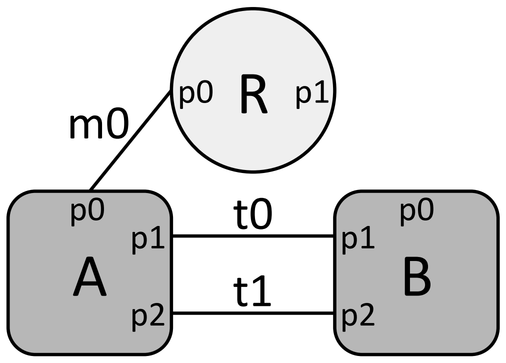

Figure 2: Example topology consisting of the management router R and testbeds A, B

3. PROPOSED ARCHITECTURE

Despite being favorable over the other protocols, LLDP comes with several shortcomings by not providing any means to solicit information from other chassis and operating on the link-layer (L2) only. On this layer communication is mostly limited to direct, neighbor-to-neighbor messages without any routing over other machines involved. The example in Figure 2 demonstrates this limitation. With R being the management router and the only host which MIB we consult, we would get correct information about the testbeds A and B and their management connections $m0_{R,A}$, $m1_{R,B}$ respectively, but the, usually more interesting, test links $t0$ and $t1$ would stay hidden. This leads to the conclusion that for a complete view of the network the content of all MIBs of the relevant testbeds, e.g. A and B too, has to be collected. This in turn complicates the setup process a bit as shown in figure 3, as it has to be ensured that every host has completed the setup step and is ready to receive/send messages or parts of the network will remain uncharted. Additionally there will be a high ratio of duplicate entries which will be dealt with in chapter 3.2.

3.1 Setup and Data gathering

The first step in the discovery process is to enable LLDP or vendor specific equivalents on the deployed hardware like switches or router and the testbeds. Depending on firmware or configuration these services may normally be disabled since they may influence experiments negatively or are generally not needed. It should be noted that they are only required during the discovery process and can safely be disabled again afterwards. On the Baltikum testbed this can be launched over the existing Command and Control (C&C) interface on the management host. Since the setup process may take a different amount of time on the hardware a synchronization barrier is employed as show in figure 3. In case of the Baltikum testbed of the TUM this is done over the management network, to which all test devices are connected, but different local viable solutions are thinkable. This barrier may only be passed once every participating host is

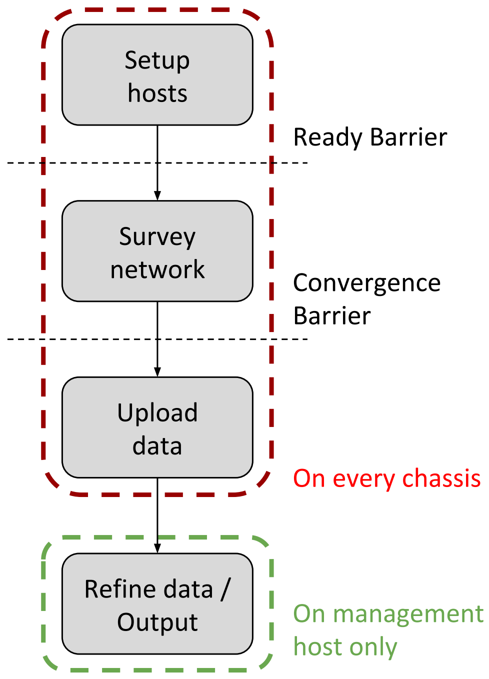

Figure 3: Proposed flow

ready, which marks the beginning of the data gathering phase.
Now every host sends the LLDP frames over its interfaces and collects information over its adjacent neighbor testbeds or hardware. The physical network should be stable at this point, so no more replugging should happen. Experiments on the Baltikum testbed have shown that this discovery process only takes a few milliseconds, but non-ideal conditions in networks spanning large distances or including wireless links may lead to dropped frames and delays. So a convergence timer of 10 sec to minimize this risk is employed with the second barrier. After this every host shuts down the discovery service and saves the gathered data locally to be collected in the next step. Additionally every chassis collects information about its hardware such as number and type of CPUs, amount of RAM and built in mainboard to enrich the dataset. As this kind of information is not part of the LLDP specification, it is done at last and simply added to the dataset.

3.2 Data aggregation and refinement

Now the collected data has to be aggregated and refined to make it usable. On the Baltikum testbed this has been realized over the already existing infrastructure for the upload of normal experiment results, but this may be solved as needed depending on the actual setup. Once the collection is completed the filtering phase begins on the management host. The need for this step can be explained with the example topology in figure 2 as we look at the theoretically collected data:

$$links = \{(A_{p1}, B_{p1}), (B_{p1}, A_{p1}), (A_{p2}, B_{p2}), (B_{p2}, A_{p2})\}^{3}$$

[3]The management links m0, m1 have been excluded for clarity. lldpd includes filter to ignore certain interfaces.

Where a tuple (X_i, Y_j) stands for a link from X on port i to Y on port j. And

$$chassis = \{A_A, A_B, A_B, B_A, B_A, B_B\}^{4}$$

where a value X_Y represents a chassis X as seen by Y.

The seemingly duplicate values come from the drawbacks of using a link layer protocol. As e.g. A is connected to B over multiple links, B will receive multiple frames from A only distinguished by the port IDs. The program will therefore detect multiple entries about the same chassis, as identified by identical chassis IDs, and merge them. The same principle applies to the set of links as each side reports the existence of a link from its Point of View (PoV), the program will try to find links with common endpoints and join them together to bidirectional ones. In the example above this would lead to the aggregation of e.g. the links (A_{p1}, B_{p1}) and (B_{p1}, A_{p1}) as they have matching chassis and ports to one link $\{A_{p1}, B_{p1}\}$.

3.3 Data presentation

To keep the output as flexible as possible the JavaScript Object Notation (JSON) was chosen to store the results for further use such as generating topology graphs later. The dataset is organized in two list. The first one contains all connections with associated metrics like link speed as detected by the LLDP:

```
[{"endpointA": {
    "interface_name": "p1",
    "device_name": "A",
    "device_id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:01"
    }
  },
  "endpointB": {
    "interface_name": "p1",
    "device_name": "B",
    "device_id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:02"
    }
  },
  "speed": "10GigBaseX"
}]
```

While the seconds list contains all discovered chassis identified by an ID and further described by a description string and fields containing their hardware info obtained by the local data collection:

```
[{"A": {
    "id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:01"
      },
      "descr": "Debian␣GNU/Linux␣3.16.0-1-grml-amd64",
      "cpu": "E3-1230␣V2␣@␣3.30GHz",
      "cpu_count": 8,
      "ram": 17179869184
    }
}]
```

The plotting process then becomes a matter of drawing all chassis and connecting them.

[4]A chassis always "discovers" itself, as it can supply the most information about it.

4. RESULTS ON THE BALTIKUM TESTBED

In this chapter the results of a prototype implementation of the strategy proposed in 3 will be presented. The program is divided in a bash script for the set-up phase which relies heavily on the existing infrastructure to start and control experiments, as the whole topology discovery process is defined as a regular experiment. For the refinement process and the image generation a Python script incorporating the graph drawing library pyGraphViz was developed. The Baltikum testbed consist of 10 Linux hosts, one switch, one router and the management network. Figure 4 shows a generated topology of the partial testbed as at the time of the discovery not every host was available. While the correctness of all the interconnections between the hosts could be confirmed via manual inspection of the interfaces, it became evident that LLDP did not detect links which start and end on the same chassis. In particular the testbeds Cesis and Nida are missing each two of these "short circuits" as seen in Figure 4. Inspection of the source code of lldpd revealed that frames that come from the same host are discarded silently to deal with faulty NIC drivers which relay broadcasted frames back. About the overall performance can be said that it's largely dependent on the boot time of the machines which can be as long as 5 minutes, while the experiment itself only takes a few seconds.

5. CONCLUSION & FUTURE WORK

While a the process of collecting the required information about a topology could be nearly automated, the goal of a complete automation could not be archived, as the process is not yet completely error free and the generated images often need manual adjustments to be prevent overlapping labels. In total this program still can provide a value help in the documentation process as a base to extent from. While the program it its prototype state is usable, certain areas need further research. In its current state the topology discovery still has to be explicitly run by the researcher and then inserted into the documentation, which can be forgotten. A possible way to solve this would be to let the LLDP service run continuously instead of running it on-demand. While it has to be determined that this does not interfere with the experiments it would provide some benefits, as now tracking changes over longer time intervals and the utilization of a central database would be possible. This would shift the responsibility to the network administrator and away from the single researcher, who could then, even retroactively, query this database to get information about the systems state at the time of his experiment. This interface again could be provided via the existing wiki to keep information central and generally available.

6. REFERENCES

[1] lldpd Development Homepage https://vincentbernat.github.io/lldpd/features.html Accessed: 2015-12-03

[2] IEEE standard for local and metropolitan area networks: Station and media access control connectivity discovery, ieee std 802.1ab, 2009.

[3] Cisco Systems, Inc. LLDP-MED and Cisco Discovery Protocol, Jun 2006.

[4] Microsoft Corporation. Link Layer Topology Discovery (LLTD) Protocol Specification, Aug 2010.

Glossary

C&C	Command and Control
CDP	Cisco Discovery Protocol
chassis	A physical component incorporating one or more IEEE 802 LAN stations and their associated application functionality.
JSON	JavaScript Object Notation
L2	Link Layer
LAN	Local area network, IEEE 802
LLDP	Link Layer Discovery Protocol
LLDPDU	LLDP Data Unit
LLTD	Link Layer Topology Discovery Protocol
MIB	Management Information Base
NDP	Neighbor Discover Protocol, part of IPv6
OSPF	Open Shortest Path First, a IP link-state routing protocol
physical topology	Physical topology represents the topology model for layer 1 of the OSI stack - the physical layer. Physical topology consists of identifying the devices on the network and how they are physically interconnected. Note that physical topology is independent of logical topology, which associates ports based on higher layer attributes, such as network layer address.
PoV	Point of View
TLV	type, length, value. A short, variable length encoding of an information element consisting of sequential type, length, and value fields where the type field identifies the type of information, the length field indicates the length of the information field in octets, and the value field contains the information, itself.

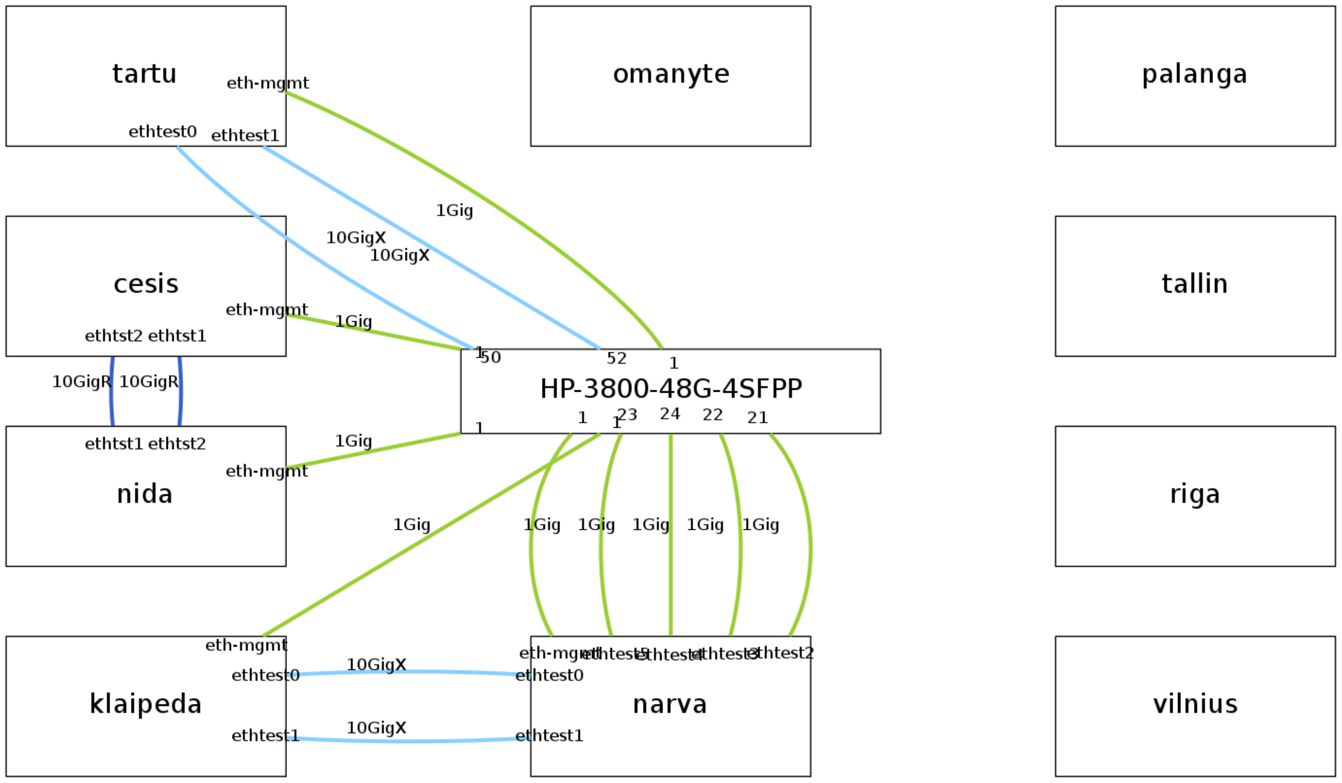

Figure 4: Generated partial topology map of the Baltikum testbed, 20.12.2015

Comparing PFQ: A High-Speed Packet IO Framework

Dominik Schöffmann
Betreuer: Sebastian Gallenmüller
Seminar Innovative Internet Technologies and Mobile Communications (IITM) WS2015/16
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: schoeffm@in.tum.de

ABSTRACT
This paper discusses the PFQ framework built for high-speed data transfers on an x86 platform. In order to facilitate the understanding of such a framework the low level mechanics of the Linux kernel are discussed. This includes historic approaches and the state-of-the-art situation. PFQs internal workings are discussed including its functional engine which is unique to this framework. The main concepts of similar frameworks are explained and compared to PFQ. Conducted measurements of PFQs performance are reviewed and compared to other measurements done by the frameworks creator.

Keywords
PFQ Linux network Internet framework

1. INTRODUCTION
The Internet is transferring more and more data by the minute. Traditionally middle boxes such as routers and firewalls were built using specialized hardware in order to speed up the processing and hence be able to handle the growing load.

A more recent approach is to use commodity hardware such as Intel x86 platforms. However normal Operating Systems (OS) kernels are not able to provide the speed needed to fully exploit the hardware provided network link speed.

Furthermore modern networking hardware is able to take away load from the CPU, for example by computing the ethernet CRC32 checksum in hardware. Another improvement is the support of multiple packet queues which allow better utilization of multi-core processors.

In order to actually use all these new possibilities, high performance frameworks need to be developed, tested and used.

The purpose of this paper is to test a framework called PFQ and compare it against other frameworks. In Section 2 the low-level mechanics of the existing network stack of the Linux kernel in which PFQ partly lives are explained. The framework itself is discussed in Section 3. Comparing PFQ to other frameworks is done in Section 4. Section 5 presents the conducted measurements and evaluates the result with respect to previous performance tests.

2. THE LINUX KERNEL
The PFQ framework mostly works inside of the Linux kernel. Also the packet retrieval from the network interface is handled in the standard Linux way. This section covers the communication between the Linux kernel and the Network Interface Controller (NIC), by explaining how this was achieved in the past and today.

2.1 Softnet
In the early days of the Linux kernel the Internet did not consist of networks capable of transferring as much data in a short period of time as it does nowadays. As an addition, the hardware design was much simpler. Most notably there were not as much multi-core processors, as today. As a result multithreading the network stack inside of the Linux kernel was no priority, even thread safety was not given until it was introduced in Linux 2.0 [1]. Thread safety was provided by using a mutex which only allowed one thread to operate inside of the network context.

All this changed with the Linux kernel version 2.3.43 in which multi-core support was added allowing multiple processors to concurrently work on network traffic as it came in from a NIC. The patchset which contained these changes was called “Softnet” [1].

One problem which led to excessive packet loss in high traffic situations however remained. At some point the hardware has to inform the OS, that there are one or more packets to be handled. Meanwhile the network card stores the packets in a DMA memory ring. Up to this point a hardware interrupt was used to signal this event for every incoming packet. This behavior is suitable when packets only arrive every once in a while, but needless and even harmful if lots of packets get received. As soon as a hardware interrupt is caught by the processor it needs to be handled immediately, delaying the work which is currently being done. As observed by Salim et al. [1] this could lead to an unequal usage of computing resources between the kernel and the user space. Therefore packets may have been queued, but never actually been processed by a process interested in these packets in the first place.

2.2 NAPI
As seen in chapter 2.1 issuing an interrupt for every single packet which arrives at the network card does not yield a very good performance for high traffic environments. The exact opposite of this approach would be to use no interrupts

at all, but to periodically poll the network card for packets. Obviously the second way will oftentimes add unnecessary latency to the further usage of the transmitted data [1].

The "New API" (NAPI) as presented by Salim et al. [1] provides a hybrid of these two worlds. When the network card is initialized, it is configured to emit an interrupt as soon as a packet arrives. Once this event occurs the interrupt for incoming packets gets disabled and the NIC is inserted into a queue of NICs having unprocessed packets. At some point in time the OS decides to handle the queued interfaces and thus the waiting packets. After all the packets from this interface are retrieved, the interrupt is re-enabled. Packets get dropped if the OS is not capable to schedule the processing of the DMA ring while it has still space left. As soon as it runs out of space no further packets can be written into the RAM, these newly arriving packets are therefore lost.

This behavior mimics the two extremes outlined before in extremely high or low traffic situations. If only a few packets arrive with a long enough time distance an interrupt is send for each packet. When a lot of packets are arriving in a rapid succession the system basically reverts to polling. Thus a good middle ground was found between these two mechanisms.

Up to now the Linux kernel uses the NAPI.

3. INNER WORKINGS OF PFQ

After understanding how the Linux kernel works for retrieving packets from network interfaces, the next upper layer of the used software stack is the PFQ framework itself.

3.1 General Structure

PFQ works inside of the NAPI context making use of the standard way to retrieve packets from network cards. When receiving a packet on the link, PFQ performs multiple steps in order to make this packet accessible to a user space program.

Figure 1 provides an illustration of the content which will be discussed in the following paragraphs.

First there is the so called *packet fetcher* [2] which operates on single packets. The one thing the *packet fetcher* does is saving a pointer to the packet inside of the *batching queue* [2]. Internally the packets are still represented by a *sk_buff* struct as used by the kernel. The whole point of the *packet fetcher* is to speed up the processing afterwards in the next stages by only working on batches of packets instead of processing every single packet on its own.

The *batching queue* is the input to the *packet steering block* [2]. Inside of the *packet steering block* it is decided to which socket the packet is forwarded. Alternatively a packet can be fed back into the Linux kernel or discarded completely. During this process the *functional engine* (discussed in section 3.2) is active. As an output location of the *packet steering block* the *socket queue* is used.

The *socket queue* is the interface between the kernel world and the user space world [2]. It is realized as a wait-free double buffer. While one buffer is being processed by a user space thread, the other buffer is used for storing new packets which are currently coming into the system. This process is facilitated by mapping the buffers into the appropriate memory spaces.

The *user space sockets* are the only instances which only operate in the user space and not inside of the kernel [2]. These sockets provide a way for threads to retrieve packets.

In order to speed up the low-level operations an additional component called an "aware driver"[2] can be used. This kind of driver provides only small code changes to the original vanilla driver. When using such a special driver the Operating System kernel does no longer receive network input. All the incoming packets are processed by PFQ. [2]

At this point it should be noted, that the normal Linux kernel would need to do a lot more preprocessing then PFQ does. For example PFQ does not implement IP reassembling or checks if a TCP segment actually belongs to a currently open connection. It does only provide a thin interface between the networking hardware and the user space processing.

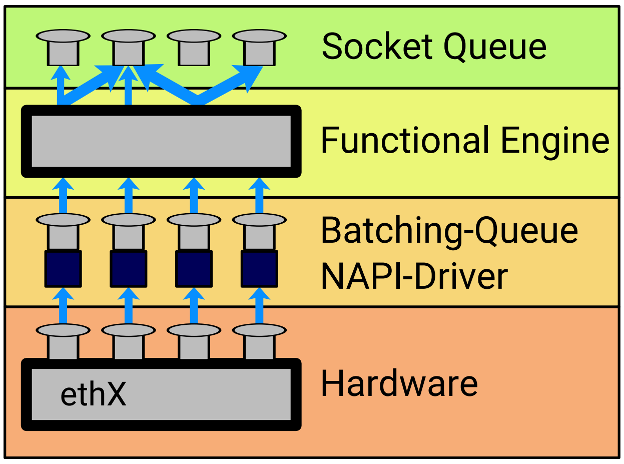

Figure 1: Inner Structure of PFQ, Image was redrawn and adapted, original by Bonelli et al. [2]

3.2 Functional Engine

The functional engine is a feature implemented within the *packet steering block*. This engine determines what to do with a packet. Possible options are: dropping, feeding it back into the kernel, sending it to a group of sockets, all sockets, one randomly chosen socket or forwarding it to a specific socket (a group with only one member socket). If a group is chosen as the destination, it may be decided if the whole group receives the packet, or only one random member, which thus results in a load-balancing situation. The used language is called "PFQ-Lang" and was first introduced by Bonelli et al. [3].

Using a functional approach inside of the kernel provides the capability to run checks against the program in order to verify properties like guaranteed termination (no loops) or that all types correct [3]. This is useful in order to avoid crashes inside of the kernel.

Inside of the program multiple processing steps can be performed, which can result in a conclusion in which direction the packet should be destined. If multiple different decisions are made during this process the last decision is used. One special case is a drop, if a packet is set to be dropped, later steps cannot overwrite this with another action. These steps are also called a “processing pipeline” [3] inside of the functional engine.

A decision on how to handle a specific packet is made by properties of the packet. These properties include the protocols used in the layers 3 and 4. Furthermore source and destination addresses and ports can be investigated. More detailed information about the packet, for example if it is fragmented, can also be used.

Having a filtering mechanism like this, which does not crash and operates inside the kernel does promise a comparably good speed. Therefore middlebox software seems like a sensible application for this type of feature.

4. COMPARISON WITH OTHER FRAMEWORKS

Comparing the basic mechanisms of PFQ to those of other frameworks is important in order to evaluate the performance and fitness for some purpose.

4.1 Netmap

Similar to PFQ Netmap also operates inside the Operating Systems kernel, although most of Netmap is based in the user space. Contrary to PFQ running Netmap means, that the network interface on which Netmap is used is no longer usable for the normal kernel. This limitation is only enforced if an application uses the Netmap framework, otherwise the interface behaves normal. Another difference is, that PFQ can work with vanilla drivers whereas Netmap requires patched drivers, which can be derived from the original Linux drivers [4]. Netmap basically works by letting the NIC write its incoming frames to the user space process memory [2].

4.2 DPDK

DPDK also exclusively uses the network interface which is switched into this exclusive mode whenever a special kernel module is loaded. Said kernel module is named “UIO” and serves as the network cards driver. The only responsibility the kernel module has, is to map the cards memory into the memory space of the process which wants to use DPDK. Obviously this process runs in the user space which means, that the framework parts inside of the kernel space are less then the ones PFQ runs inside of the kernel. This framework does not only provide a fast way to send and receive packets, but a complete framework to realize Data Plane Devices like routers or switches. One example feature which is important to such devices is an efficient implementation of longest-prefix-matching. [4]

4.3 PF_RING ZC

One feature which PFQ adapted from PF_RING ZC is the usage of aware drivers. PF_RING ZC actually was the first framework to propose such an approach [2]. Similarly to PFQ PF_RING ZC also uses shared memory rings for the kernel and the user space. The difference however is, that PFQ uses two such buffers which get swapped, whereas PF_RING ZC only has one buffer [2]. Measurements conducted by Bonelli et al. [2] showed, that PF_RING ZC does not perform as well as PFQ in regards to scaling up to multiple threads and hardware queues.

5. MEASUREMENTS

Measurements of the performance of PFQ were carried out. These include the two basic functions of such a framework, namely sending and receiving packets.

5.1 Setup

The measurements were conducted on two servers connected by a 10 Gbit/s ethernet link. As CPUs the first server used an Intel Xeon E3-1230, the other server an Intel Xeon E3-1230 V2. The used Network Interface Controllers (NICs) were an Intel 82599ES and an Intel 82599EB respectively. No further supporting kernel modules or patches to the network driver were used. The systems were running the Linux kernel version 3.19.

PFQ was compiled from source using the Glasgow Haskell Compiler (GHC) version 7.8.4. The version of PFQ itself was 5.0.4.

5.2 Traffic Generation

Using PFQ traffic was generated on the server with the Intel Xeon E3-1230 processor and the Intel 82599ES NIC. During the experiment the number of threads used for packet generation was incremented from 1 to 4 in steps of 1. The packet size was fixed to 60 Bytes. In another run the size of the packets was set to 128 Bytes, in order to measure if PFQ also performs well in situations with more realistic packet sizes. Using the second setup only 1 and 2 threads were tested. During this test hugepages were not mounted.

The test was performed with the bundled test tool “pfq-gen” using the command line `pfq-gen -l 60 -R -t 0.5.ethtest0 -k 1,...`

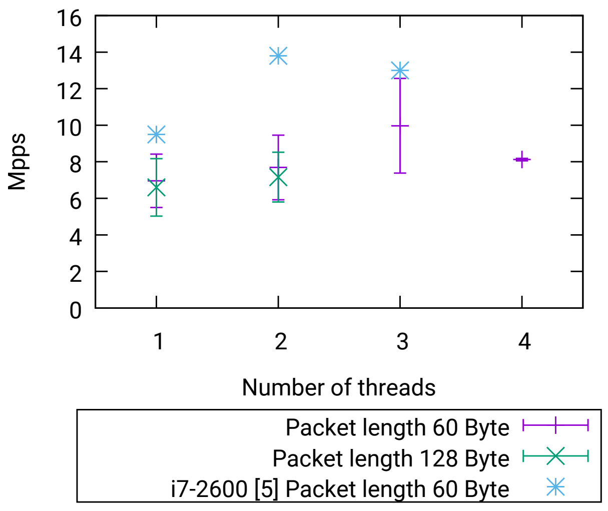

Figure 2: Packet generation

As can be seen in Figure 2, using more than one core does help to generate packets at a faster rate. The framework does also scale with rising packet sizes, since the sending rate did only decrease inside of the error margin. During the experiment it was also observed, that the sending rate dropped below the average at multiple occasions, leading to a quite high standard derivation which is also shown in Figure 2. Notably the average sending rate peaked with three cores and descended with the usage of four cores.

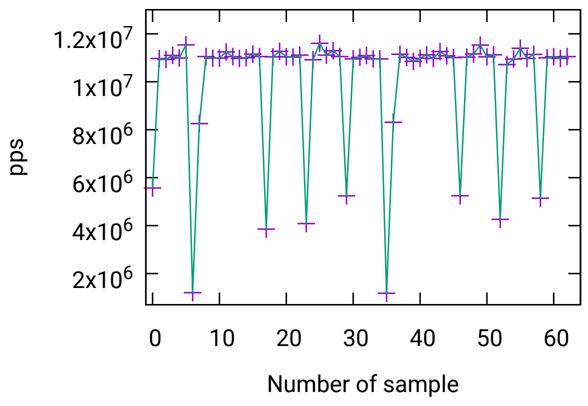

Figure 3: Packet generation on 3 cores

Figure 3 shows the performance of PFQ in packets-per-second for every taken sample of the measurement with 3 cores. It can be observed, that most samples are on one level above 10 Mpps and only a few downwards directed peaks are featured in the graph. These peaks occur relatively regular.

5.3 Traffic Capturing

Capturing traffic was performed on the Intel Xeon E3-1230 V2 processor and the Intel 82599EB Network adapter. The traffic was generated by PFQ running on 3 cores and building packets with a size of 60 Bytes. During the measurements the amount of used hardware queues was raised from 1 to 4. Each queue was bound to one processor core, which results in a multithreading situation inside of the kernel. In contrast to the traffic generation measurement hugepages were mounted.

Analogous to the traffic generation the used tool also was inside of the PFQ test suite. It is called "pfq-counter" and the used command was `pfq-counters -c 64 -t 0.5.ethtest0`. Before issuing the command the PFQ kernel module was reloaded with the appropriate queue number.

Figure 4 illustrates that the packet capturing capabilities of PFQ rise in a linear fashion with the number of queues used. As in the traffic generation test a significant derivation in the measured data was found.

5.4 Comparison with other measurements

Similar benchmarks were performed by Bonelli et al. and the results published in the projects wiki page [5]. Differences between the tests were the kind of processor, and software optimizations. The measurements in this paper were conducted with vanilla drivers, whereas Bonelli et al. [5] used

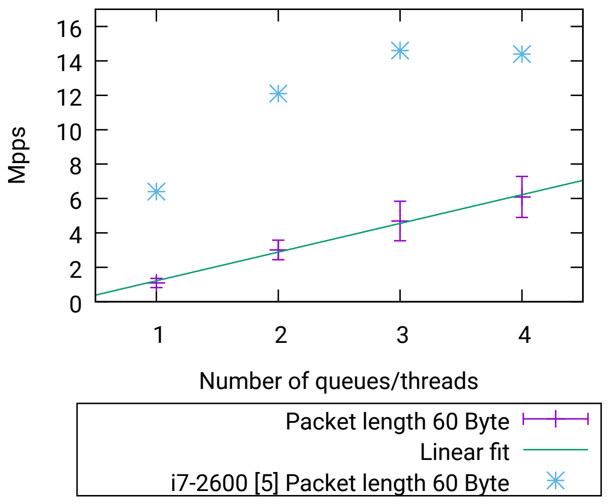

Figure 4: Packet capture

a driver which was optimized by the *pfq-omatic* tool. Furthermore Bonelli et al. loaded a special kernel module which provided support for Direct Cache Access. The used traffic generation and capturing tool were the same and even shared the same options. However the PFQ kernel module was loaded with different options. The measurements conducted by Bonelli et al. used the option "xmit_batch_len=128"[5], whereas the measurements done in this paper used the default value of one. Another big difference were the clock speeds of the used processors, which were lower in the here presented results. Using hugepages might as well have an impact on the performance (it is not stated if Bonelli et al. used hugepages, although this is very probable).

It can be observed, that the measurements conducted in this paper do not yield such a high performance as the previous measurements made by Bonelli et al. [5].

6. CONCLUSION

In this paper historical approaches to handling packets at a low-level were discussed. These included thread safeness and interrupts. Modern operating systems migrated away from issuing one interrupt per packet to issuing one interrupt per batch of packets.

Furthermore an overview of the building blocks of the PFQ high performance I/O framework was given. These included getting the packets from the low-level kernel space, steering them according to a functional engine and enqueuing them to be accessible by user space applications. The internals and capabilities of said functional engine were discussed.

Other frameworks promising similar functionality were examined for similarities and differences. Discussed were Netmap, DPDK and PF_Ring ZC. All provided the same basic functions like sending and receiving packets, but the methods achieving this do differ from one to another. All use some kind of kernel module, although the ratio of work done in the kernel space and the user space vary heavily.

Lastly measurements were taken to determine how many packets PFQ can send and receive on x86 commodity hardware. Inside of these measurements the number of used cores was variated. Within these measurements it was observed, that there were severe drops of the transmission rate. Another important note is, that when capturing traffic there is a linear correlation between the received packets and the number of hardware queues which PFQ was able to use.

7. REFERENCES

[1] J. H. Salim, R. Olsson, and A. Kuznetsov. Beyond softnet. In *Proceedings of the 5th annual Linux Showcase & Conference*, volume 5, pages 18–18, 2001.

[2] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. On multi–gigabit packet capturing with multi–core commodity hardware. In *PAM'12 Proceedings of the 13th international conference on Passive and Active Measurement*, pages 64–73. Springer, 2012.

[3] N. Bonelli, S. Giordano, G. Procissi, and L. Abeni. A purely functional approach to packet processing. In *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '14, pages 219–230, New York, NY, USA, 2014. ACM.

[4] F. W. D. R. Sebastian Gallenmüller, Paul Emmerich and G. Carle. Comparison of frameworks for high-performance packet io. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2015.

[5] Nicola Bonelli. PFQ Benchmarks https://github.com/pfq/PFQ/wiki/Intel-IXGBE-10-20G (last retrieved 8.12.2015)

ISBN 978-3-937201-51-1
DOI 10.2313/NET-2016-07-1

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