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)

Summer Semester 2016

Munich, Germany

Editors

Georg Carle, Daniel Raumer, Lukas Schwaighofer

Publisher

Chair of Network Architectures and Services

Network Architectures
and Services
NET 2016-09-1

FI & IITM
SS 16

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

Sommersemester 2016

München, 15. 04. 2016 – 01. 08. 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)
Summer Semester 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: `https://www.net.in.tum.de/~carle/`

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

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

Cataloging-in-Publication Data

Seminars FI & IITM SS 16
Proceedings zu den Seminaren „Future Internet“ (FI) und „Innovative Internettechnologien und Mobilkommunikation“ (IITM)
München, Germany, 15. 04. 2016 – 01. 08. 2016
ISBN: 978-3-937201-52-8

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2016-09-1
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2016-09-1
Series Editor: Georg Carle, Technische Universität München, Germany

Vorwort

Vor Ihnen liegen die Proceedings der Seminare „Future Internet“ (FI) und „Innovative Internettechnologien und Mobilkommunikation“ (IITM). Wir sind stolz, Ihnen Ausarbeitungen zu aktuellen Themen, die im Rahmen unserer Seminare im Sommersemester 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 Frederic Naumann, der in seiner Ausarbeitung „Garbled Circuits“ die Funktionsweise verschlüsselt ablaufender Berechnungen betrachtet. Im FI Seminar wurde dieser Herrn Dominik Scholz verliehen für seine Ausarbeitung „Diving into Snabb“, in welcher er das Paketverarbeitungsframework Snabb analysierte.

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

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

- Best Practices für Firewalls
- Metriken der Resilience
- Das Baltikum-Testbed
- Softwarebasiertes Switching in Windows
- Analyse des Snabb-Frameworks

Auf `https://media.net.in.tum.de/#%23Future%20Internet%23SS16` können die aufgezeichneten Vorträge zu diesem Seminar abgerufen werden.

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

- QUIC - Quick UDP Internet Connections
- Looking Glasses zum Einblick in die Struktur des Internets
- NFV und OPNFV
- Datenlecks in IT Systemen
- Garbled Circuits

Auf `https://media.net.in.tum.de/#%23IITM%23SS16` 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 `https://www.net.in.tum.de`.

München, September 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 Mobile Communication" (IITM) which took place in the summer semester 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 each seminar with an award. This semester the award in the IITM seminar was given to Mr Frederic Naumann who analysed the garbled circuit algorithm that allows computation without revealing the inputs in his paper "Garbled Circuits". In the FI seminar the award was given to Mr Dominik Scholz for his paper "Diving into Snabb" wherein he analysed the packet processing framework Snabb.

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

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

- Firewall Best Practices
- Resilience Metrics
- The Baltikum Testbed
- Virtual Switching in Windows
- Analysis of the Snabb Framework

Recordings can be accessed on `https://media.net.in.tum.de/#%23Future%20Internet%23SS16`.

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

- QUIC - Quick UDP Internet Connections
- Using Looking Glasses to Understand the Internet's Structure
- NFV and OPNFV
- Data Breaches in IT Systems
- Garbled Circuits

Recordings can be accessed on `https://media.net.in.tum.de/#%23IITM%23SS16`.

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

Munich, September 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

Marcel von Maltitz (maltitz@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Heiko Niedermayer (niedermayer@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

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

Quirin Scheitle (scheitle@net.in.tum.de)
Technische Universität München, Mitarbeiter I8

Lukas Schwaighofer (schwaigh@in.tum.de)
Technische Universität München, Mitarbeiter I8

Seminarhomepage

`https://www.net.in.tum.de/teaching/ss16/seminars/`

Inhaltsverzeichnis

Seminar Future Internet

Seminar Innovative Internet Technologien und Mobilkommunikation

Firewall Best Practices

Christian Winter
Advisor: Cornelius Diekmann
Seminar Future Internet SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: winterch@in.tum.de

ABSTRACT

With growing Internet usage in both private and commercial fields, filtering network traffic to prevent harmful use is gaining importance. But correctly filtering packets without restricting any wished functionality is complicated. New protocols are introduced and existing ones like ICMP and IP are updated. For each protocol, filtering rules have to be created separately. We summarize the most important best practices in firewall filtering for common protocols and give insights on why these filtering rules have to be applied. Additionally, we explain where information on creating individual policies can be found online and issues that need consideration are listed. The best practices summarized here are generic and can therefore be implemented for most firewall models. To give a concrete example, a filtering policy for a gateway firewall according to these generic best practices is given throughout this paper.

Keywords

Firewall, Best Practice, IP Filtering, ICMP Filtering, Packet Filtering

1 Introduction

Besides encryption and authentication, filtering network traffic in firewalls is an important and widely used measure to prevent attacks against online services, like online banking and shops. Therefore, creating and maintaining firewalls is a task network administrators have to face. A main part in firewall management is creating and maintaining a filtering table. Filtering tables can easily grow to several thousand lines of rules, as is evident from different rule sets collected for research [10]. Therefore, checking whether a firewall filters potentially harmful messages can be challenging. In addition, the IETF recommendations and best practices are spread over several different RFCs and often implicit in protocol standards. This makes it difficult for firewall administrators to ensure their firewall acts according to best practices. Network protocols and corresponding filtering policies are still a changing and widely researched field, evident from recently published articles and RFCs covering these topics [11, 14, 18]. This paper gives an overview over best practices for widely used protocols and summarizes them. In addition it lists where additional information on these best practices can be found and what attacks filtering accordingly can counter.

Section 2 gives recommendations on how to deal with certain address registries and port ranges and how spoofing of address registries can endanger the network. Furthermore it covers several aspects of the IP protocol family, like IPv6 extension headers and fragmentation, as well as attacks on those. Section 3 covers the dangers in the Internet Control Message Protocol (ICMP) in versions 4 and 6, lists possible attacks and gives recommendations which messages and types should be dropped in a firewall. Finally, Section 4 gives an outlook what else firewall administrators should consider, including IP tunnels, multicast messages, and other protocols. To give detailed filtering recommendations on some policy dependent best practices, this paper explains how a example firewall could be configured. We assume this firewall resides as a gateway between a local network and the Internet. Inside the protected network a web server should be reachable for HTTP requests from the Internet. This example will be used throughout the paper to show how filtering recommendations based on policies or services in the network could be implemented.

2 IP and Port Considerations

An often used method in packet filtering is deciding whether the packet may pass through the firewall or is dropped based on the source or destination addresses and the transport layer ports of a packet. Many of these rules are based on the policy of the network and dependent on what services should be available to and from the outside world. But there are best practices which should be implemented in every rule set, depending on the location of the firewall in a network. This section deals with these filtering best practices for different firewall locations.

2.1 Special Purpose IP Address Registries

Several IP address registries are reserved by the Internet Assigned Numbers Authority (IANA) for special purposes and therefore need additional consideration when implementing firewall rule sets. In this section we want to discuss the registries that should be filtered for different firewall use cases. A full list of all registries can be found in RFC 6890 [6]. There are some addresses that are not valid for use in any packet leaving a host. RFC 6890 [6] defines the address registries that are not valid as source or destination address for both IPv4 and IPv6. These registries have to be dropped in every firewall. In addition to those addresses, there are other restrictions for firewalls that are not limited to local-only traffic. When a firewall monitors traffic involving at least one interface connected to a network with a public address range, some other address registries are not valid for use in packets transiting this firewall. These address registries mainly consist of those reserved for private use and local

doi: 10.2313/NET-2016-09-1_01

Firewall type	DROP range	
All firewalls	127.0.0.0/8	192.0.2.0/24
	198.51.100.0/24	203.0.113.0/24
	240.0.0.0/4	(192.0.0.0/24)
Gateway firewalls	0.0.0.0/8	10.0.0.0/8
	100.64.0.0/10	169.254.0.0/16
	172.16.0.0/12	192.0.0.0/29
	192.168.0.0/16	198.18.0.0/15
	255.255.255.255/32	
Example firewall	192.88.99.0/24	224.0.0.0/4
	& gateway	

Table 1: IPv4 Filter Considerations

Firewall type	DROP range	
All firewalls	::1/128	2001:db8::/32
	2001:10::/28	(2001::/23)
Gateway firewalls	::/128	100::/64
	2001::/32	2001:2::/48
	fc00::/7	::ffff:0:0/96
	fe80::/10	without fe80::/64
Example firewall	64:ff9b::/96	2002::/16
	& gateway	

Table 2: IPv6 Filter Considerations

protocols. Table 1 and Table 2 list the address registries that should be blocked in these firewalls. The addresses in parentheses have to be dropped unless RFC 6890 [6] defines a more specific registry as valid. The registry of *fe80::/10* is not valid outside a local network, with exception of link local addresses *fe80::/64* used for the Neighbour Discovery Protocol (NDP) [29]. Additionally the tables show which registries will be dropped in our example firewall. For our firewall we don't want to allow non-global address registries. To keep the tables compact, the registries that have to be dropped for all gateway firewalls are not repeated for our example firewall, but have to be dropped as well. Furthermore we do not want to support protocols not essential to network communication, like addresses reserved for benchmarking. As discussed in Subsection 4.2 we are not depending on any multicast messages, so we drop the corresponding address registry in our firewall. Finally we do not want to support IPv4 to IPv6 address translation, so we also drop the registries reserved for translation.
To prevent attackers from inserting invalid packets into the network, e.g. packets with addresses that are reserved for local protocols, and thereby bypassing other security measures, all packets containing such an address as a source or a destination have to be dropped by the firewall.

2.2 Spoofing Considerations

Some Denial of Service (DoS) attacks use spoofed IP source addresses, addresses that were not assigned to the sender, making it hard to trace back this kind of attack [12].

2.2.1 Attack Example

An example for a spoofing attack is a variant of TCP SYN flooding described in RFC 2827 [12]. Thereby an attacker sends many TCP SYN packets to the victim using random spoofed source addresses. This results in a high load of TCP SYN/ACK packets to be sent out by the victim and additional a lot of TCP connections to store and track on the victim. This high load of packets might lead to a system crash or at least to a drop in overall performance for other connections.

2.2.2 Ingress Filtering

To counter this kind of attacks with spoofed packets to be started from an attacker inside a network to a remote victim, all routers of this network should implement ingress filters [12]. Spoofed packet attacks can only be fully countered if all networks are filtered accordingly to RFC 2827 [12], else an attacker in an unfiltered network could still carry out this attack. In our firewall, for example, it would be nearly impossible to tell if a packet arriving from the internet was really sent by the host in the source address because all remote hosts are connected to the same interface. On the other hand it is easily detectable if a packet trying to leave our network has a source address that was not assigned to a host inside the network. Even if all firewalls would implement ingress filters, RFC 2827 [12] states that spoofing an address from a host residing in the same network as the attacker, and therefore being a valid address at the router, will still be possible. An ingress filter should ensure that a packet entering the firewall has an IP address that resides behind the interface it entered through, thereby "prohibiting an attack[er] from using 'invalid' source addresses which reside outside of this prefix range."[12] A general filtering rule for ingress filtering can't be given because it is highly dependent on the interfaces of a firewall and the subnets residing behind these, which makes ingress filtering a complex task mainly for large firewall rule sets. A good way of ensuring that a firewall prohibits spoofed packets from passing through is the algorithm described by Diekmann et al. [11] which checks spoofing protection for a finished rule set, or to use a reverse path forwarding mechanism described in RFC 3704 [3]. Such a mechanism ensures that packets arriving at a firewall will only be forwarded if the firewall is on the route from the source specified in the packet to the destination. This is done by checking if a new packet to the source of the received packet would be sent via the interface the received packet arrived at. If this is not the case, the packet is dropped at the firewall [3].

2.3 IP Header Options

The Internet Protocol specifies header options to extend the protocol functionality. Those options mainly fulfil an important task in the routing of IP packets, but some are the basis for attacks. For IPv4, RFC 7126 [18] lists possible threats related to header options and what impact dropping packets containing them would have. We will only deal with some of the options listed there, but it is recommended that all of the options are reviewed.

2.3.1 Unknown IPv6 Extension Headers

Especially the extension headers of IPv6 pose a threat to network security as the IPv6 specification RFC 2460 [9] does not specify how packets with unknown options should be dealt with at firewalls along the path of the packet. But it specifies that "extension headers are not examined or processed by any node along a packet's delivery path, until the packet reaches the node [...] identified in the Destination Address field of the IPv6 header." [9] Because this prohibits middleboxes from checking for unknown header options, RFC 4942

doi: 10.2313/NET-2016-09-1_01

[7] states that this rule may be ignored by firewall administrators. Even if a middlebox firewall ignores the processing rule named above by checking extension headers and detects an unknown header option, it may be problematic to drop this packet because the firewall can not know if this unknown option is implemented in the target. The best trade-off is selecting firewalls which allow filtering based on IP extension header types used in a packet. This allows the firewall to drop unknown options and easily whitelist new header types when they are introduced [7].

2.3.2 Source Routing

IPv6 source routing, header type 43, can be used to specify a list of intermediate hosts along the path of a packet. At each of these intermediate hosts the destination address is set to the address of the next intermediate host or, at the last intermediate host, to the destination [9]. This header option, especially routing type 0, can be used for several different attacks, e.g. for bypassing filtering devices and bandwidth exhaustion [20]. Therefore RFC 5095 [1] deprecates the routing type 0 routing header (RH0) for IPv6. Packets containing a RH0 can be dropped in the firewall, but firewalls must not drop all packets containing a routing header, and forwarding packets with a type 43 header of other routing types must be permitted [1]. In our example firewall we want to drop IPv6 packets containing a RH0. Similar to the IPv6 RH0, IPv4 options 131 and 137 implement source routing. These types could be used to bypass firewall rules, to learn about the networks topology and to exhaust the bandwidth of a network and dropping them only has a small impact on troubleshooting [18]. Because of the small benefit of source routing compared to the dangers we want to drop all IPv4 packets containing a option type 131 or 137 in our example firewall, and it is suggested to do this in all firewalls.

2.3.3 Fragmentation

Another threat posed by IP header options are fragmentation attacks. This kind of attacks utilizes the vague regulations on filtering in the definitions of IPv4 and IPv6.

2.3.3.1 Overlapping Fragments

A common attack scenario is bypassing firewall checks by overriding protocol headers, e.g. the TCP header, with overlapping fragments. A detailed attack on IPv4 can be found in RFC 1858 [34] and on IPv6 in RFC 5722 [24]. Although the attack is very similar for both IP versions, the prevention measures differ. To prevent overriding transport-layer headers in IPv4, it should always be checked that the fragment offset field in the IP header is larger or equal to the length of the transport-layer header [34]. In IPv6, routers are not allowed to fragment packets and therefore fragmentation is managed by the fragment extension header, defined in Section 4.5 of RFC 2460 [9]. Because of this different fragmentation approach, a more general filtering method has to be applied. To counter an overlapping fragment attack in IPv6, when two fragments overlap, the whole packet, including all fragment packets arriving later on, must be silently discarded [24]. To this rule there is one exception. Atomic fragments are packets containing a fragment header without being split into multiple packets. Such packets are sent when an IPv6 host received a ICMPv6 Code 2 messages stating that along the packet path a section has a MTU smaller then the guaranteed IPv6 minimum MTU of 1280 [17]. To allow IPv6 hosts to use this option, RFC 6946 [17] updates RFC 5722 [24] to allow certain packets through the firewall that would have been filtered by the original filtering policy stated above. Packets with no fragment offset and the "more flag" set to 0, atomic fragments, have to be processed independently of any other packets. This means packets with the same source and destination address and fragment identification are not to be seen as overlapping when they are atomic fragments and therefore must not be dropped [17].

2.3.3.2 Tiny Fragments

Another attack against IPv6 using fragmentation is a DoS attack using unnecessary tiny fragments without a terminating packet, a packet where the "More Fragments" flag is set to 0. This could lead to an overload of fragment buffers and drain resources of the target machine. To prevent such an attack a firewall should always check that packets containing non-final fragments are at least half the size of the protocols guaranteed minimum MTU, in the case of IPv6 640 octets [7]. This way no correctly fragmented packet is dropped while the number of packets an attacker can send through the firewall is minimized, thereby relieving the fragmentation buffer. Another attack utilizes the small minimum MTU in IPv4. This allows the fragment size to be chosen small enough to force some TCP header fields into the second fragment. This prohibits TCP filter rules from matching the header parts in the second fragment and thereby might allow the packet through the firewall [34]. In addition there is an attack combining both IPv4 attacks described here. This attack combines tiny and overlapping fragments to bypass security measures preventing the overlapping fragment attack or the tiny fragment attack alone [28]. The countermeasure for this attack can be used to eliminate all other fragmentation attacks against IPv4 listed here. It must be ensured that a firewall checks that all the header fields relevant for filtering must be in the first fragment and that, after the first packet, no fragment offset value small enough to override relevant header fields is allowed [28]. In the example firewall we want to have every TCP header field up to the "Window Size" (Bytes 14 and 15) for filtering. Therefore we request the first fragment to carry at least 16 Bytes of payload and that there is no packet after the first with a fragment offset smaller than 2.

2.4 Application Ports

In addition to the special IP address registries described in Subsection 2.1, there are some well-known application ports that need special consideration. Although all of these ports serve a well intended purpose, the applications associated with these ports have been the target of successful attacks and therefore should be blocked in the firewall. The ports and applications listed are the most common examples of such vulnerable ports but additional vulnerabilities might come up in the future and may be added to the block list.

2.4.1 Network Management Protocols

Some protocols for managing hosts over the network have vulnerabilities that attackers could utilize in attacks against victims running such a protocol. The most prominent network management protocols with weaknesses are listed in

doi: 10.2313/NET-2016-09-1_01

this section and it is explained how and why to block them. Microsoft's implementation for Remote Procedure Calls (RPC), described by Microsoft's TechNet [27], is an interprocess communication technique for client/server architectures. The implementation includes various vulnerabilities, some exploited by the MSBlaster worm. To prevent such attacks, the corresponding TCP and UDP port 135 should be blocked by a firewall.

The Simple Network Management Protocol (SNMP), as described in RFC 1157 [5], is an IETF standard protocol used to gather and manage information on network devices running on UDP ports 161 and 162 as defined in RFC 3417 [32]. For SNMP multiple vulnerabilities at least in the versions 1, 2 and 3 are known, some of which can lead to DoS attacks and are open to format string and buffer overflow attacks [22]. To prevent attacks on SNMP, it is suggested by Jiang [22] to block all ports related to SNMP in the firewall to prevent attacks from outside the network. When it is required to access SNMP from outside the network, the access to these ports should be restricted to a limited IP address range, e.g. by using a VPN and creating an exception only for VPN traffic.

The Intelligent Platform Management Interface (IPMI) is a widely used standard for extensible, scalable interoperable server hardware management architecture independent of the server's power status. But the architectures protocol has vulnerabilities that could be utilized in attacks against the network, e.g. probing for other devices in the network, as pointed out by Gasser et al. [14]. To counter these flaws, Gasser [14] suggests to drop UDP port 623 used in IPMI in a firewall or to restrict the access to IPMI devices to a VPN and drop all other packets in the firewall.

2.4.2 Filtering Recommendation

For our own firewall we also want to drop additional TCP/IP application layer protocols. For example Microsoft NetBIOS and SMB do not fulfil functionality we need to access from outside our firewall, so we want to drop the corresponding ports. In addition to these ports, we also want to block the telnet protocol used for remotely controlling hosts over the Internet. This protocol runs on TCP port 23 and allows to take full control over a host while transmitting everything unencrypted [31]. Therefore it would allow attackers to intercept communication and take over hosts in our network. To prevent these attacks we drop the corresponding port. Most of the vulnerabilities attributed to applications mentioned can be prevented by blocking the corresponding ports in the firewall. To simplify firewall configuration we want to whitelist only port 80 in our firewall. This way all other ports are dropped by the firewall and we don't have to deal with the potentially dangerous ports separately. If a firewall configuration using blacklisting is wished, Table 3 lists the ports discussed in this section. But because there might be other security issues in other applications not listed in this paper it is advised to use a whitelisting strategy, or at least to update the rule set accordingly when another flaw is detected.

Port	Protocol	Application	Recommendation
23	TCP	Telnet	DROP
135	TCP	Microsoft RPC	DROP
135	UDP	Microsoft RPC	DROP
137	UDP	NetBIOS	DROP
137	TCP	NetBIOS	DROP
138	UDP	NetBIOS	DROP
139	UDP	NetBIOS	DROP
161	UDP	SNMP	DROP
162	UDP	SNMP	DROP
445	TCP	SMB	DROP
623	UDP	IPMI	DROP

Table 3: Port filtering recommendations

3 ICMP and ICMPv6

The Internet Control Message Protocol (ICMP) plays an important part in upholding Internet communication. With the introduction of IPv6, the corresponding ICMPv6, gained even more importance in communication. It handles error notification in the Internet Protocol and therefore must be implemented on every IP module [30]. But some of these control messages can be used for malicious intent and therefore several ICMP messages should be filtered in a firewall while others must not be blocked. This section deals with ICMPv4 and ICMPv6 message filtering along with possible attacks against ICMP and suggests a concrete filtering policy.

3.1 ICMPv4

During the lifespan of IPv4, different control messages have been introduced and others have been deprecated again already. This section deals with summarizing the essential ICMP messages and which should be dropped because they do not fulfil a valid purpose any more. A blogpost by John Albin [2] summarizes all common ICMPv4 messages and gives recommendations in filtering them. This, of course, is no scientifically accurate source, but it does not violate any RFC that was released until the publishing of this post in 2005. Later on, one of the messages marked essential, ICMP source quench, was deprecated and will be dealt with later on, all other recommendations are still valid today. Table 4 summarizes the filtering recommendations for our example firewall. In the following part we want to take a look at a possible classification of some ICMPv4 message types and how they should be dealt with.

3.1.1 Essential Messages

There are messages which have to be passed and accepted by all IPv4 hosts to enable communication. Albin [2] lists message type 3, destination unreachable, as essential to communication. ICMPv4 type 3 messages should be allowed through the firewall, because without these ICMP messages, communication is impossible under certain circumstances. When a datagram must be fragmented to be forwarded, because the network MTU is too small and the packet is not allowed to be fragmented, the packet has to be discarded and a "destination unreachable" error message may be sent [30]. A host dropping all ICMPv4 type 3 messages will not notice unfragmentable packets sent are too big and therefore will not be able to reach the destination.

doi: 10.2313/NET-2016-09-1_01

3.1.2 Deprecated Messages

Some ICMPv4 messages are not relevant in today's network communication and therefore have been deprecated. The ICMPv4 source quench mentioned in Subsection 3.1 is one of these messages. It is not to be used for congestion control because it is known to be unfair and ineffective [16]. In addition to this message, RFC 6918 [19] deprecates other message types and explains why they are not relevant any longer. These messages should not only be dropped because they are deprecated, but also because they might be used in attacks. The source quench message, for example, can be used to permanently slow down IP connections in both IPv4 and IPv6 and therefore slowing down hosted services [15]. Table 4 summarizes all the deprecated messages in the drop recommendations.

3.1.3 Potentially Dangerous Messages

In addition to the types deprecated, there are other types that firewalls might want to filter. The ICMPv4 types 0 and 8 used for ping and type 11 used for traceroute are not essential to communication as stated by Albin [2] but are used widely for host location and troubleshooting. If one doesn't wish the hosts in his network to be located by others it is suggested to block those message types. The message types 13 to 16 give additional information about the addressed host that attackers could use and Albin [2] suggests to drop these messages, therefore our example firewall will drop them, as well as all other ICMPv4 types not listed. We will however allow ping and traceroute for troubleshooting purposes.

3.2 ICMPv6

As stated in the introduction to this section, ICMPv6 gained importance in the IP communication compared to the ICMPv4 protocol. It is essential for IPv6, taking over many functions in error handling and information distribution, and therefore includes many different message types and options. Because it is a lot more extensive than ICMPv4, filtering is more difficult and requires a more detailed approach. To enable more detailed filtering depending on the state of connections, RFC 4890 [8] suggests to enable stateful packet filtering where possible. This allows the firewall to determine whether an ICMP error message arriving at the firewall corresponds to a sent packet and therefore is legit. For this to be possible, RFC 4890 [8] states that the firewall must be able to perform deep packet inspection on these error messages. This section will discuss ICMPv6 traffic passing through the firewall, but packets addressed to the firewall itself should be filtered as well. Table 6 suggests filtering for this traffic, detailed information on why it should be filtered can be found in RFC 4890 [8]. In the Tables 5 and 6 the terms "consider dropping" and "policy dependent" refer to the corresponding lines in the table. The types listed there will be dropped in our example firewall when referred, but are not repeated for reasons of clarity. Furthermore RFC 4890 [8] includes an example script for configuring an iptables firewall according to most of the best practices in Appendix B of the RFC, useful for quick configuration according to these guidelines.

3.2.1 Essential Messages

There are message types essential to communication, like messages for error detection and notification when a host is unreachable. Dropping these messages will prevent or severely impact communication establishment and maintenance, therefore these messages must not be dropped [8]. RFC 4890 [8] states the types 1, 2 as essentials, aswell as the ICMP code 0 of type 3 and codes 1 and 2 for type 4. Furthermore the types 128 and 129 are essential to Teredo, discussed in Subsubsection 4.1.1, and should be allowed because they pose no imminent threat in IPv6 port scanning. In our firewall we therefore allow all these message types to pass.

Action	Types
ALLOW	3, 0, 8, 11
DROP	4, 6, 13-18, 30-37
DROP when not needed	other types

Table 4: ICMPv4 filtering recommendations

3.2.2 Important Messages

In addition to the essential messages there are ICMPv6 messages that fulfil an important task in IP connections. These messages should not be dropped unless there is a severe reason. The type 3 code 1, type 4 code 0 and types 144 - 147 for mobile IPv6 should be allowed to pass the firewall [8]. Because we do not want to support mobile IPv6 in our example network, we will not allow the latter through the firewall.

3.2.3 Policy Dependent Messages

The broad variety of ICMPv6 messages includes many that are only essential to certain protocols as well as unallocated types. Therefore a policy has to be defined on how to deal with these messages. Messages from unallocated areas that only transit the firewall and do not end on a host inside the protected network should be able to pass the firewall. These types might be implemented on other hosts and already in use and dropping them could cause damage to this connection. For the site the firewall is protecting, administrators might want to choose dropping these messages until they are allocated to prevent covert channels. A part of these messages might become essential in the future therefore the firewall policy should be updated regularly to include such allocations [8]. In addition to those solely dependent on policy, RFC 4890 [8] lists other messages that should be dropped in most cases. ICMPv6 includes messages meant only for local information exchange that should not be able to leave the local network and therefore should not occur outside a local scope. Table 5 includes all such messages in the drop section, a more detailed classification can be found in RFC 4890 [8]. Our firewall will drop all these policy dependent messages, as it is located on the outside of the local scope and we do not want to support any experimental protocols.

3.2.4 Possible Attacks

With the variety of ICMPv6 messages come several possible attacks. These include man in the middle attacks on protocols, probing, and different Denial of Service attacks. In addition ICMP error messages can be used to establish a covert channel through error type payloads [8]. To counter these attacks, very detailed filtering of ICMP traffic in the firewall is essential.

Action	Types (Codes)
ALLOW	1, 2, 3 (0), 4 (1,2), 128, 129
Consider allowing	3 (1), 4 (0), 144-147
Policy dependent	15, 5-99, 102-126 154-199, 202-254
Consider dropping	100, 101, 127 138-140, 200, 201, 255
DROP addressed to example network	5-99, 102-126, 144-147 150, 154-199, 202-254 & consider dropping

Table 5: ICMPv6 filtering recommendations passing the firewall

Action	Types (Codes)
ALLOW	1, 2, 3 (0), 4 (1,2), 128, 129 130-136, 141-143, 148, 149 151-153
Consider allowing	3 (1), 4 (0), 144-147, 150
Policy dependent	4-99, 102-126, 137, 139, 140
Consider dropping	100, 101, 127 154-199, 200-255
DROP in example	144-147, 150 policy dependent & consider dropping

Table 6: ICMPv6 filtering recommendations addressed to the firewall

4 Other Considerations

This document focuses on the most common and important firewall best practices, but considerations should be made in other fields of network traffic that are left out in this paper. Some of those network issues are introduced briefly in this section.

4.1 IP Tunnels

IP tunnels are often used to encrypt IP payload or to support protocols not transmittable over IPv4. These tunnels pose a potential threat for networks as firewalls are not always able to check all of the contained data [25]. Therefore firewall administrators should consider if they want to accept packets using tunnel mechanisms in their network.

4.1.1 IPSec and Teredo

The most prominent implementations of IP tunnels are IPSec [23] and Teredo [21]. IPSec enables encryption and integrity protection of IP packets, providing confidentiality of the payload during transport. But for firewalls, this means they can not know what is transported in the payload, therefore anything could be transported through the firewall in an Encapsulating Security Payload (ESP) packet. To ensure that IPSec connections can take place, middlebox firewalls should always allow IPSec traffic, for end point firewalls a more differentiated handling is required. Teredo tackles problems that occur when trying to access IPv6 networks residing behind NAT devices by tunnelling these packets via UDP. It is possible that packet filters do not recognise that there is another IP datagram contained in the packet and therefore might not check this inner packet [25]. More detailed considerations to these and other problems of IP Tunnelling can be found in RFC 6169 [25].

4.1.2 6to4

A special case of IP Tunnels are 6to4 tunnels that are used as a IPv6 interim mechanism to connect IPv6 networks over IPv4 clouds described in RFC 3056 [4]. The transmission protocol has weaknesses that can lead to DoS attacks and service theft, where an attacker uses a service he is not authorized to use [33]. RFC 3964 [33] discusses a variety of attacks against 6to4 communication and provides suggestions on fixing some of these flaws.

4.2 Multicast

A network administrator should always consider if and how they want their network to be reachable with multicast messages. A general recommendation whether to filter multicasts or not cannot be given because it highly depends on the policy of the local network. Depending on this policy, it might be feasible to drop some or all multicast packets at the firewall or allow communication with the inner network. Most intranets will not need external multicast at all, and others might only need some of these messages. RFC 2588 [13] gives recommendations how firewalls should and can control the traversal of multicast packets and what else firewall administrators should consider when dealing with multicast messages for a gateway firewall. Our example firewall will drop all multicast packets as they do not fulfil a purpose we need in our local network.

4.3 Other Protocols

Although this paper mainly focuses on TCP and UDP in the transport layer and IP and ICMP in the network layer, firewall administrators should be aware that there are other network protocols that a firewall can stumble upon. An example for a protocol not discussed here is the Routing Information Protocol (RIP) used for distance vector routing [26]. A whitelisting firewall would normally drop packets using unknown protocols because it does not have a rule allowing theses datagrams. But rule sets accepting all packets from a certain source address will allow these datagrams through the firewall, which might have impact on the networks security. Even though these protocols are not widely used and might not pose an actual threat, administrators should be aware that other protocols exist.

5 Conclusion

To give a detailed overview over the most important best practices, we reviewed dozens of RFCs and other sources and summarized the filtering recommendations they give for several aspects of different transport and network layer protocols. In addition we show what attacks can be prevented by implementing these best practices and what liabilities might come with these implementations. We learned that some aspects of these best practices are dependent on local policies and the type of firewall used. Furthermore we showed how these policy dependent implementations could be done for a example firewall we defined.
This paper lists the most important best practices available at the moment, but other aspects of firewall calibration have to be considered as well. The best practices listed here should be updated and extended accordingly if new vulnerabilities are found or a new protocol is introduced.

6 References

[1] J. Abley, P. Savola, and G. Neville-Neil. Deprecation of Type 0 Routing Headers in IPv6. RFC 5095 (Proposed Standard), Dec. 2007.

[2] J. Albin. Everything you ever wanted to know about ICMP, (but were afraid to ask rfc792), 2005 (accessed March 17, 2016). http://john.albin.net/essential-icmp.

[3] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), Mar. 2004.

[4] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), Feb. 2001.

[5] J. Case, M. Fedor, M. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[6] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman. Special-Purpose IP Address Registries. RFC 6890 (Best Current Practice), Apr. 2013.

[7] E. Davies, S. Krishnan, and P. Savola. IPv6 Transition/Co-existence Security Considerations. RFC 4942 (Informational), Sept. 2007.

[8] E. Davies and J. Mohacsi. Recommendations for Filtering ICMPv6 Messages in Firewalls. RFC 4890 (Informational), May 2007.

[9] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[10] C. Diekmann. net-network. GitHub repository https://github.com/diekmann/net-network, 2014 (accessed March 20, 2016; last commit March 18, 2016).

[11] C. Diekmann, L. Schwaighofer, and G. Carle. Certifying spoofing-protection of firewalls. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 168–172. IEEE, 2015.

[12] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[13] R. Finlayson. IP Multicast and Firewalls. RFC 2588 (Informational), May 1999.

[14] O. Gasser, F. Emmert, and G. Carle. Digging for dark IPMI devices: Advancing BMC detection and evaluating operational security. *Traffic Monitoring and Analysis*, 2016.

[15] F. Gont. ICMP Attacks against TCP. RFC 5927 (Informational), July 2010.

[16] F. Gont. Deprecation of ICMP Source Quench Messages. RFC 6633 (Proposed Standard), May 2012.

[17] F. Gont. Processing of IPv6 "Atomic" Fragments. RFC 6946 (Proposed Standard), May 2013.

[18] F. Gont, R. Atkinson, and C. Pignataro. Recommendations on Filtering of IPv4 Packets Containing IPv4 Options. RFC 7126 (Best Current Practice), Feb. 2014.

[19] F. Gont and C. Pignataro. Formally Deprecating Some ICMPv4 Message Types. RFC 6918 (Proposed Standard), Apr. 2013.

[20] J. Hui, J. Vasseur, D. Culler, and V. Manral. An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL). RFC 6554 (Proposed Standard), Mar. 2012.

[21] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), Feb. 2006. Updated by RFCs 5991, 6081.

[22] G. Jiang. Multiple vulnerabilities in SNMP. In *Computer*, volume 35, pages 2–4. IEEE, Apr 2002.

[23] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFCs 6040, 7619.

[24] S. Krishnan. Handling of Overlapping IPv6 Fragments. RFC 5722 (Proposed Standard), Dec. 2009. Updated by RFC 6946.

[25] S. Krishnan, D. Thaler, and J. Hoagland. Security Concerns with IP Tunneling. RFC 6169 (Informational), Apr. 2011.

[26] G. Malkin. RIP Version 2. RFC 2453 (Internet Standard), Nov. 1998. Updated by RFC 4822.

[27] Microsoft TechNet. What Is RPC?: Remote Procedure Call (RPC), Last updated 2016 (accessed March 17, 2016). https://technet.microsoft.com/en-us/library/cc787851(v=ws.10).aspx.

[28] I. Miller. Protection Against a Variant of the Tiny Fragment Attack (RFC 1858). RFC 3128 (Informational), June 2001.

[29] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), Sept. 2007. Updated by RFCs 5942, 6980, 7048, 7527, 7559.

[30] J. Postel. Internet Control Message Protocol. RFC 792 (Internet Standard), Sept. 1981. Updated by RFCs 950, 4884, 6633, 6918.

[31] J. Postel and J. Reynolds. Telnet Protocol Specification. RFC 854 (Internet Standard), May 1983. Updated by RFC 5198.

[32] R. Presuhn. Transport Mappings for the Simple Network Management Protocol (SNMP). RFC 3417 (Internet Standard), Dec. 2002. Updated by RFCs 4789, 5590.

[33] P. Savola and C. Patel. Security Considerations for 6to4. RFC 3964 (Informational), Dec. 2004.

[34] G. Ziemba, D. Reed, and P. Traina. Security Considerations for IP Fragment Filtering. RFC 1858 (Informational), Oct. 1995. Updated by RFC 3128.

Resilience Metrics

Valentin Zieglmeier
Advisor: Dr. Heiko Niedermayer
Seminar Future Internet SS2016
Chair of Network Architectures and Services
TUM Department of Informatics
Email: v.zieglmeier@tum.de

ABSTRACT

Computer networks have become an essential part of our world. To ensure their operational safety, measures need to be taken. Resilience defines how well a network can maintain operational safety in the face of various challenges to its operation. Metrics are needed for each aspect of resilience to enable quantification of the resilience of networks.
In this paper the different disciplines of resilience are named and explained. For each discipline the corresponding resilience metric is presented. To illustrate the need for resilience in practice, two case studies and possible solution strategies utilizing resilience metrics for each example are discussed.

Keywords

Resilience; Network security; Security measurement

1. INTRODUCTION AND MOTIVATION

Computer networks are ubiquitous today. Not only do we interact with them every day and rely on them in our personal life, we need them even more than we might sometimes think. Financial services can only be provided if we can rely on stable networks that are secure and safe. The military relies on networks to control troop movement, to plan attacks and to monitor drones. And many services we personally use are provided only through the internet.
An example for this are cloud services used to back up data and photos. Users of these services depend on the cloud service to work more reliable and to be more resilient than their personal computers. Another example is the Nextbit Robin, a smartphone that uploads unused apps and media from the smartphone to the cloud. When the user wants to access this uploaded content, it has to be downloaded first [17]. Lastly, services like Netflix aim to substitute the personal video collection with on-demand streaming. To watch a movie, the customer simply starts streaming it from the server [16].
Because it is important that these networks are reliable and work consistently, efforts are being made to ensure their resilience. To quantify the resilience of a network, we need to be able to measure it. In this paper a common definition of resilience is presented in section 2. Section 3 defines aspects of resilience and a corresponding metric for each. In section 4 case studies of real world examples are discussed. Finally, section 5 gives a summary and a conclusion.

2. DEFINITION: RESILIENCE

Resilience is no fixed concept. One definition can be found in [6, p. 6], [20, p. 1246] and [21, p. 12]:

> Resilience is the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation.

We assume that the network is under constant threat. As it is extremely important that we can rely on certain networks to function, the challenges that they face must not jeopardize their operations. Resilience is a measure to guarantee an acceptable level of service in the face of these challenges. Resilience metrics may be used to quantify how well a network can retain this level of service regarding different challenges.
As a prerequisite, common challenge categories need to be defined first. There are five basic categories of challenges for networks, listed below [6, p. 3].

- Environmental (e.g. node mobility),
- Malicious (attacks),
- Non-malicious (e.g. unusually high traffic load),
- Large scale disasters (e.g. a hurricane) and
- Lower level failures (e.g. path failures).

One strategy to achieve resilience is $D^2R^2 + DR$ (*Defend*, *Detect*, *Remediate*, *Recover* + *Diagnose*, *Refine*; see figure 1). It is at the core of the ResiliNets [20, p. 1253] and the ResumeNet projects [21, p. 24]. Both of these projects research resilience metrics and developed a framework to achieve resilience in networks.
This strategy is based on the idea that unforeseen events will always occur. After installing general defensive measures, these events have to be detected and the defensive measures strengthened to react to similar challenges more appropriately. An example for this might be failed links in a network. Remediation in this case would mean that traffic is rerouted. After the damage has been repaired or the challenge has been overcome, the recovery is initiated. In this stage the system returns to its normal state [20, p. 1254].
These measures are assumed to always have flaws, so they have to be constantly diagnosed and refined. In case the automatic measures were not enough to ward off the challenge, they may be developed further [20, pp. 1254-1255].

doi: 10.2313/NET-2016-09-1_02

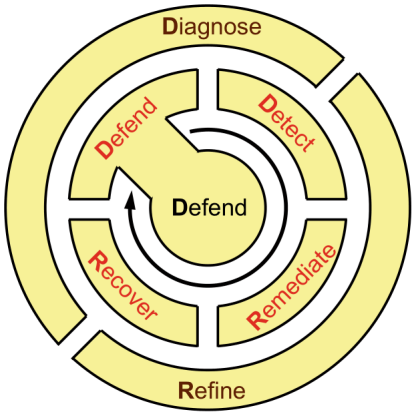


Figure 1: The $D^2R^2 + DR$ strategy of the ResiliNets project [20, p. 1253].

3. RESILIENCE METRICS

As we found in the last chapter, resilience includes tolerance of multiple different threat categories. Below we discuss the different disciplines of resilience and how each metric can be calculated or measured. The dimensions of resilience are adapted from [20, pp. 1247-1249] (see figure 2).

3.1 Challenge tolerance

Networks regularly face challenges. Their ability to tolerate them and to continue operating even when challenged defines one of the two main dimensions of their resilience.

3.1.1 Survivability

Survivability is defined as the "capability of a system to fulfill [sic] its mission, in a timely manner, in the presence of threats such as attacks or large-scale natural disasters" [20, p. 1247]. This refers to the general failure tolerance of a system. "Failure" refers to "correlated failures" [20, p. 1247] either because of an attack, or because large parts of the system fail.
As fault tolerance (see section 3.1.2) is defined as a subset of survivability, we can conclude that survivability is defined as a combination of failure tolerance and fault tolerance.
Survivability defines how well the system can maintain an acceptable level of service when parts of the system fail or are attacked. It does not include the ability of the system to prevent such attacks or failures.
The second part of survivability, fault tolerance, is discussed in the next section.

Metric. One definition of a method to assess survivability can be found in [10, pp. 8-9]:

> A system is survivable if it complies with its survivability specification. [...] A survivability specification is a six-tuple, {S, E, D, V, T, P}.

The six-tuple consists of the acceptable service specifications (S), all theoretical ways the system can degrade (E), all realistically reachable degradation states (D), an ordering of system specifications from S for certain degradation states from D from the perspective of the user (V), valid transitions between acceptable forms of service (T) and service probabilities for each member of S (P) [10, pp. 9-10].
By defining these different properties and analyzing them, the necessary survivability of a network can be assessed in consideration of the context.

An example might be a personal backup server. This server is connected to a RAID 1 with two hard drives that both contain the same data. Possible service states (S) might be:
s_1: Direct monitoring done at the server
s_2: Remote monitoring with email alerts for failures
s_3: No monitoring (e.g. when travelling)
Theoretical ways the system may degrade could be (reduced for simplicity):
E: {drive status(d) $\mapsto$ {good, one failed, both failed},
energy(e) $\mapsto$ {good, none}}
These are then combined into all possible combinations and realistic combinations are collected (D). In the example we use, all combinations are possible:
d_1: {d $\mapsto$ good, e $\mapsto$ good}
d_2: {d $\mapsto$ one failed, e $\mapsto$ good}
d_3: {d $\mapsto$ both failed, e $\mapsto$ good}
d_4: {d $\mapsto$ good, e $\mapsto$ none}
d_5: {d $\mapsto$ one failed, e $\mapsto$ none}
d_6: {d $\mapsto$ both failed, e $\mapsto$ none}
With this we can define priorities of service states for each degradation state (V), as seen exemplary in table 1.

Table 1: Example of V: Higher is better

	s_1	s_2	s_3
d_1	2	3	1
d_2	3	2	1
d_3	3	2	1
d_4	3	1	2
d_5	3	1	2
d_6	3	1	2

Continuing with T, possible transitions have to be defined. In our example, every state is accessible from every state. This can for example be marked in a graphic containing the service states, with arrows connecting them for every possible transition.
Lastly for P, probabilities of each service state are calculated. These are the percentages of time that the system will probably be in the respective state. In our case that might be:
$\Pr[s_1] = 0.5$; $\Pr[s_2] = 0.3$; $\Pr[s_3] = 0.2$
This concludes the exemplary survivability specification.

3.1.2 Fault tolerance

Fault tolerance is defined as a subset of survivability. It is "the ability of a system to tolerate faults such that service failures do not result" [20, p. 1247]. Contrary to failures, faults are "random uncorrelated failure[s] of components" [20, p. 1247].
This metric does not define if such faults can be prevented, but how well the system can handle them. The goal is to provide acceptable service to the users.

doi: 10.2313/NET-2016-09-1_02

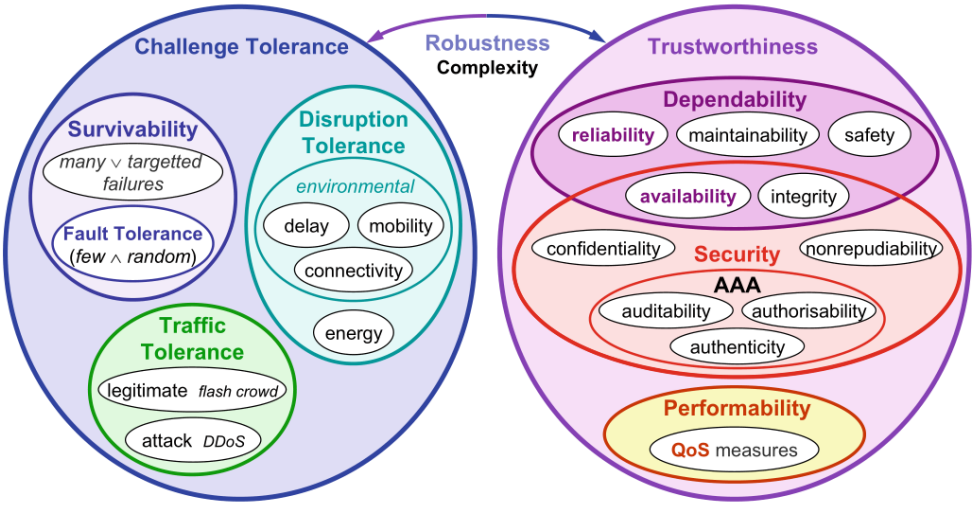


Figure 2: The different resilience disciplines [20, p. 1247]

Metric. As fault tolerance can be achieved with redundancy, a measurement for good fault tolerance might be the number of redundant instances available. Assessing how much is "good enough" is depending on the context. For military or financial applications this might be significantly higher than for private applications.

3.1.3 Disruption tolerance

Disruption tolerance is defined as the "ability of a system to tolerate disruptions in connectivity among its components" [20, p. 1248]. It consists of multiple possible disruptions, grouped as environmental and energy challenges. Environmental challenges can be delays, mobility and weak connectivity [20, p. 1248]. Node mobility refers to nodes that change their position, which influences connectivity and changes the wireless topology of the network. An example for this are mobile phones.
This metric is similar to survivability, but at a smaller scale. Disruption tolerance can refer to the system tolerating the failure of just a few connections, while survivability may refer to a power grid failing, resulting in the outage of a complete local network.

Metric. The disruption tolerance could be measured as the percentage of possible disruptions the system can tolerate or as the percentage of system parts that may be disrupted without the system failing. Which aspects are most important again depends on the context.
To handle different possible disruptions, redundancy of multiple types should be in place. This includes systems utilizing different technologies and hardware or being installed in multiple geographic locations.

3.1.4 Traffic tolerance

The last aspect of challenge tolerance, traffic tolerance, is defined as "the ability of a system to tolerate unpredictable offered load without a significant drop in carried load (including congestion collapse), as well as to isolate the effects from cross traffic, other flows, and other nodes" [20, p. 1248].
Traffic tolerance is a metric that can be observed when it is too low. If a higher-than-usual amount of users try to connect to a server at the same time and that server cannot be reached any more as a consequence, the traffic tolerance is inadequate. This is called an "unusual but legitimate traffic load" [20, 1249]. An example for that might be many students trying to register for a course at the same time. This problem can be especially severe if the network fails just when it is needed the most.

Metric. Traffic tolerance can be measured by testing how much traffic the system can handle while still offering acceptable service to legitimate users. Different tests can include traffic from an identical IP address as well as from different IP addresses (similar to a simple distributed denial-of-service attack).
To test various more advanced tolerance strategies the requests can be modeled after realistic and unrealistic client behavior. Possible patterns from legitimate flash events [7, pp. 3-5] or from distributed denial-of-service (referred to as DDoS) attacks [7, pp. 5-7] can be taken into consideration.

3.2 Trustworthiness

The second dimension of resilience is trustworthiness. This refers to the predictability of the system from the perspective of the consumers of its services.

3.2.1 Dependability

The first aspect of trustworthiness is defined as the dependability of the system, which "quantifies the reliance that can be placed on the service delivered by a system" [20, p. 1248]. The main parts of this are availability and reliability.
Availability can be defined as "readiness for correct service" [3, p. 6]. It is the percentage of time the system is available in contrast to being unavailable. This metric is important for servers that are accessed regularly and whose availability is important to the business of a company, like shopping sites. More than 80 % of mobile users will abandon a site if it loads longer than 20 seconds or not at all [8].

Reliability by contrast is "continuity of correct service" [3, p. 6]. This can be understood as how long the service is available without interruption. This metric is important e.g. for video streaming or voice over IP services, where a continuous connection is necessary to maintain an acceptable level of service for users.
The remaining parts of dependability are maintainability, i.e. the "ability to undergo modifications, [sic] and repairs" [3, p. 6], safety, i.e. the "absence of catastrophic consequences on the user(s) and the environment" [3, p. 6] (protection *from* the system in contrast to security, protection *of* the system) and integrity, i.e. the "absence of improper system alterations" [3, p. 6] ("improper" means "unauthorized" here [3, p. 6]).

Metric. The availability of a system is defined as
$A = MTTF/MTBF$ [20, p. 1248].
The mean time between failures (MTBF) is defined as
$MTBF = MTTF + MTTR$ [20, p. 1248].
The mean time to failure (MTTF) is the mean of continuous service uptime periods. The mean time to recovery (MTTR) is the mean of continuous service downtime periods. This means the availability is equivalent with the percentage of time that the system was available.
The reliability is depending on the period of time that the system should work continuously without failing. After defining this period, the reliability can be calculated as
$R(t) = \Pr[\text{no failure in } [0,t]]$ [20, p. 1248].
It is the probability that the system may fail during the specified period of time t.
Maintainability, safety and integrity are qualitative and binary properties, meaning that they are either true or false and this depends on the requirements. They need to be assessed and prioritized in consideration of the context.

3.2.2 Security

The second aspect of trustworthiness, security, is defined as "the property of a system, and the measures taken such that it protects itself from unauthorised access or change, subject to policy" [20, p. 1249]. Availability and integrity, which were defined as parts of dependability, are shared with security (see figure 2). Additional properties are authenticity, authorizability, auditability, confidentiality, and nonrepudiability [20, p. 1249].
In contrast to the other disciplines, security is not aimed at keeping the network running at any cost. The target is to prevent malicious access and modification. This can mean shutting down or cutting off parts of the network that might be compromised.

Metric. Metrics for availability and integrity are discussed in section 3.2.1. Authenticity, authorizability, auditability, confidentiality and nonrepudiability have to be assessed in a security audit. If that is possible, auditability is given and the other properties can be analyzed. These too are qualitative and binary properties, so this assessment is based on the requirements. If a security audit is not possible, they cannot be assessed. In general, it is difficult to measure and compare these properties.

3.2.3 Performability

The last aspect of trustworthiness, performability, is defined as the "property of a system such that it delivers performance required by the service specification" [20, p. 1249].
Similar to traffic tolerance, inadequate performability can be experienced directly by the users and affect their behavior. Ten seconds of loading time can make almost 50 % of mobile users abandon the page completely [8].

Metric. Performability is another requirement that is almost completely depending on the context. How much delay or throughput is necessary for acceptable service has to be decided on.

3.3 The connection between challenge tolerance and trustworthiness

To describe how the two dimensions challenge tolerance and trustworthiness are interconnected, the following aspects are named.

3.3.1 Robustness

The robustness of a system is defined as "the trustworthiness [...] of a system in the face of challenges that change its behaviour" [20, p. 1249]. This is exactly the interconnection of challenge tolerance and trustworthiness. Robustness therefore quantifies how trustworthy a system remains when challenged. It should ideally not change.
Other definitions of robustness are closer to survivability, like that "internet communication must continue despite loss of networks or gateways/routers" [23, p. 6].

Metric. Some metrics that robustness is based on are often not clearly quantifiable, like security or dependability, making the quantification of robustness also inexact. In general, a better challenge tolerance of a system means that its robustness is better as well.

3.3.2 Complexity

Complexity is named as the result of resilience mechanisms. If more of these mechanisms are deployed, the system gets more complex which "may result in greater network vulnerability" [20, p. 1249].
This means that work on resilience may also reduce the resilience regarding other aspects of the network.

Metric. Complexity increases "may be related to maximization of the information shared (transferred) within the system" [18, p. 4]. This does not mean that more shared information leads to higher complexity, but a lower amount of information that is shared may indicate a lower complexity of the network.

4. CASE STUDIES

Below we analyze two real world cases where resilience was a factor. This can help to understand the importance of different aspects of resilience.
First, we analyze Hurricane Katrina to see what impact it

doi: 10.2313/NET-2016-09-1_02

had on network infrastructure and to gain a better understanding of the importance of survivability and dependability.
Second, we analyze the 2007 cyber-attacks on Estonia directed from Russia. That example illustrates the need for good traffic tolerance, especially regarding networks that have an increased risk to be attacked.

4.1 Case study 1: Hurricane Katrina

Hurricane Katrina is one of the best-known large-scale natural disasters to hit the United States. This hurricane was "the costliest and one of the five deadliest hurricanes to ever strike the United States" [9, p. 1]. Between 23rd and 30th of August 2005 at least 1.245 people died as a consequence. The property damage is estimated to have been around $ 108 billion [9, p. 1].
Even though such large-scale natural disasters are not common, it is extremely important that people can still communicate with each other and learn about safety measures or whether they are affected in similar cases of emergency. Additionally, the regular network service in other states or countries should neither be affected by such disasters.

4.1.1 *Impact of Hurricane Katrina on network infrastructure*

A report published by Renesys found that over 35 % of networks in Mississippi, over 10 % in Louisiana and over 5 % in Alabama among others were outaged during Hurricane Katrina [4, p. 2] (see figure 3).

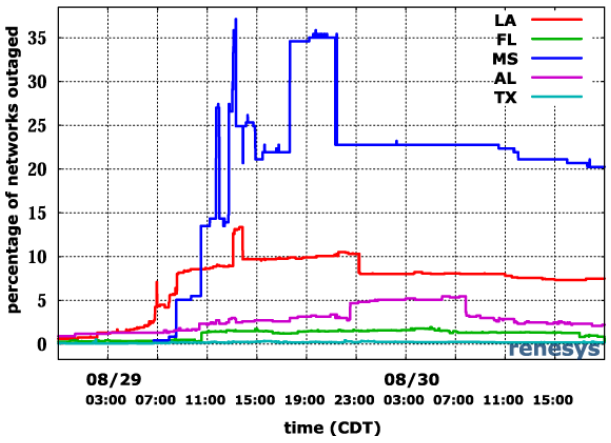


Figure 3: Percentage of globally routed networks outaged by Hurricane Katrina, as measured by Renesys [4, p. 3].

While the early outages were quickly fixed and the network operation recovered, the networks experienced extended outages in the two most affected regions, Mississippi and Louisiana, over the entire 10-day period (see figure 4).

4.1.2 *Survivability through path diversification*

There is no data about how these outages affected users of the network, but it can be assumed that the significant effect on the network infrastructure of the region affected them as well. Without necessary survivability measures, the adequate dependability for the users could not have been maintained. It is very important to be able to keep the network

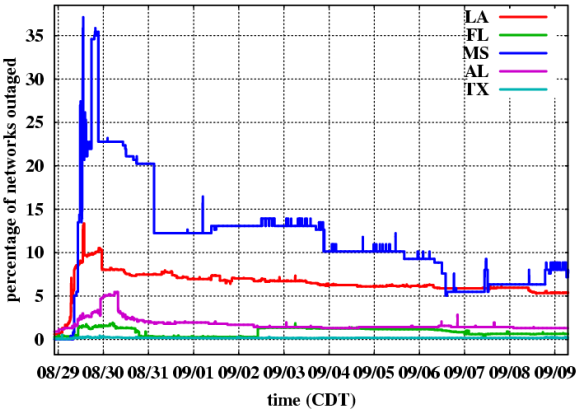


Figure 4: Percentage of globally routed networks outaged by Hurricane Katrina over the entire 10-day period since 29th of August, as measured by Renesys [4, p. 7].

stable even if significant parts of it fail.
To prevent such disasters from affecting the whole network, redundant systems should be in place. These systems should be located away from each other with enough distance between them that only one of them can be affected at once. Also the network should be able to dynamically reroute requests in case a part of it has failed. This approach is referred to as path diversification, as multiple possible paths for connections in the network are introduced by it. It is already common practice, but cannot be easily controlled and measured.

There has been research in the field of path diversification with the goal of achieving a high flow reliability [19]. Diversity of two paths P_a and P_b is defined as
$D(P_b, P_a) = 1 - \frac{|P_b \cap P_a|}{|P_a|}$ [19, p. 345].
For two completely disjoint paths this formula equates to 1, for equal paths to 0. (The endpoints of the paths are excluded as they are always identical.)
It is important to note that neither node nor link diversity alone is enough, as proven by [19, p. 345]. To be a valid fallback, the paths have to be disjoint regarding both links and nodes.
With this definition the path diversity of a network can be assessed and ensured. To make sure that a failing path cannot cause a network outage, the diversity of at least two paths has to be 1, meaning that they share no node or link. That means in case a node or link fails there is always at least one alternative path.
This can be further improved by taking past node or link failures into account and ensuring that fallback paths in different geographic locations or utilizing completely different technologies (e.g. optical and satellite) exist.

4.1.3 *Possible rerouting strategies*

To ensure that packages can be rerouted, a possible scenario is the following. Each routing table gets additional alternative routes for each node in the network. Should a link fail the node can propagate that network change and reroute packages accordingly [14, pp. 2-4].

doi: 10.2313/NET-2016-09-1_02

An alternative rerouting strategy is the proactive failure insensitive routing, proposed by [12]. The idea is to reroute the packages locally when a link fails. Instead of explicitly notifying other nodes of the failure, they can infer it from the packets they receive. If a packet is received on an unusual interface, it was probably rerouted. This means no changes to the link state propagation mechanism are necessary in this approach [12, p. 2].

4.2 Case study 2: Estonia under cyber-attack

In April 2007 Estonia provoked Russia by removing a war statue honoring soviet war dead from World War II from the city center of Tallinn. This led to protests from Russians in Estonia and Russia and violent riots in Tallinn. At the same time as the protests began, DDoS attacks started hitting Estonian servers [13]. After only affecting the website of the foreign minister of Estonia, they quickly spread. The main targets were the website of the Estonian police, the Ministry of Finance and other government websites [2]. Origin of the attack were Russian IP addresses which lead the Estonian government to claim involvement of the Russian government [13, p. 1]. This claim was questioned by security researcher Mikko Hyppönen of F-Secure in the *Helsingin Sanomat* [1]. While the load was not extraordinarily high, it hit Estonia badly because the country is small and its networks are not prepared for such load [13, p. 1].

4.2.1 Impact of the attacks on the network

The attacks were no steady stream and not of equal length, they were distributed over many days and different attack strengths (see table 2).

Table 2: Attack lengths and bandwidths [2]

Attacks	Length
17	$<$ 1 minute
78	1 min - 1 hour
16	1-5 hours
8	5-9 hours
7	$\geq$ 10 hours

Attacks	Bandwidth
42	$<$ 10 Mbps
52	10-30 Mbps
22	30-70 Mbps
12	70-95 Mbps

The impact of the attacks was enormous. Analyses by Mikko Hyppönen show that the homepage of the Estonian government was hardly accessible if at all (see figure 5).

Estonia tried to counter these attacks, but failed to do so. The countermeasures were not effective, so they had to cut their connection to the rest of the internet to restore services within Estonia for their population, many of whom relied on these services for their everyday life [13, p. 3].
This measure was the last resort and it helped. But it resulted in degraded service for the Estonian people. Internet access was effectively cut, meaning the only websites they could visit were Estonian. Accessing ATMs from other countries was now also almost impossible [13, p. 3].
Should the attacks have been launched from inside Estonia, this measure would have been of little use. In any case, if the attacks did not stop after some time, it would not have resolved the issue.

4.2.2 Traffic tolerance through resource accounting

DDoS attacks aim to prevent legitimate users from accessing a service. Either the users are directly attacked, or the server they try to access is targeted [24, p. 1], [22].
These attacks work because servers discard requests when they cannot handle them anymore. If a server follows this strategy, some legitimate requests will be discarded and some illegitimate requests handled which results in degraded service for legitimate users. One possible strategy to prevent this denial-of-service is resource accounting [15, p. 10]. In this scenario the requests are observed and discarded based on their properties and predefined rules. This aims to filter out more illegitimate requests and provide service to actual users.

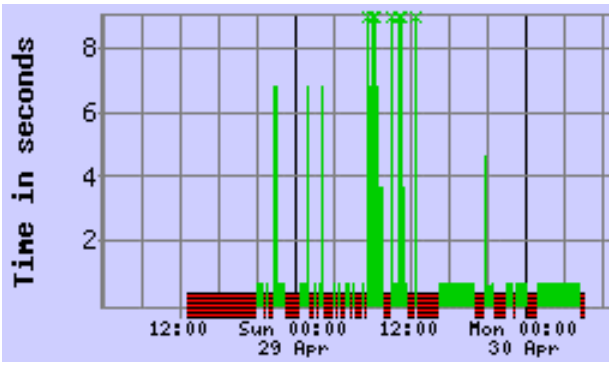


Figure 5: Availability statistics from Netcraft for the homepage of the Estonian government (cropped) [5]. Green (light) are successful connections, red (dark and horizontally striped) are failures.

4.2.3 A possible resource accounting strategy

Two properties seem to be very effective to detect illegitimate requests: The per-client request rate and cluster overlap before and during the event [7, p. 7].
The per-client request rate during DoS attacks was found to differentiate compared to legitimate flash events. During flash events it will slow down as the server responds slower to requests. In contrast to that the request rate will stay constant during DDoS attacks [7, p. 8].
To further refine which requests are discarded, the server can periodically cluster the legitimate requests during normal load. A possible clustering strategy is the network-aware clustering, for example by using the method proposed by Krishnamurthy and Wang [11]. When the server experiences unusually high load it can compare the requests to the clusters from past requests and discard unknown clients first. They are more likely to be attackers [7, pp. 7-8].

5. CONCLUSION

Networks have become the backbone of our society, and we depend on them working consistently and continuously. Challenges to networks occur constantly and come from various sources. To withstand these challenges while maintaining acceptable levels of service, networks need to be resilient. To ensure the resilience of a network, agreed on metrics that quantify the different aspects of resilience are necessary. These metrics can be used to assess the resilience of a network as well as to compare different approaches to achieve resilience and to understand the consequences of protective measures.

doi: 10.2313/NET-2016-09-1_02

In this paper a common definition of resilience was presented. A metric was found for every aspect of resilience. Two aspects have been analyzed more extensively in two case studies and possible solution strategies for both cases have been presented.
It is possible to achieve basic resilience in a network based on the work in this paper. The presented metrics provide a basic understanding of resilience and its aspects. The two discussed case studies and the presented solution strategies offer profound information regarding similar scenarios.
The remaining aspects of resilience may be studied in more detail and the corresponding metrics can be refined. To find out which countermeasures are most effective, some resilience disciplines need to be researched more extensively and experimented with.

6. REFERENCES

[1] N. Anderson. Massive ddos attacks target estonia; russia accused. `http://bit.ly/221ZnDj`, 2007. Accessed: 2016-04-03.

[2] ARBOR. Estonian ddos attacks - a summary to date. `http://bit.ly/1Rturu3`, 2007. Accessed: 2016-04-01.

[3] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[4] J. Cowie, A. Popescu, and T. Underwood. Impact of hurricane katrina on internet infrastructure. *Report, Renesys*, 2005.

[5] M. Hypponen. Update on the estonian ddos attacks. `http://bit.ly/1T2Zkbk`, 2007. Accessed: 2016-04-01.

[6] A. Jabbar. A framework to quantify network resilience and survivability. 2010.

[7] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of the 11th international conference on World Wide Web*, pages 293–304. ACM, 2002.

[8] KISSmetrics. How loading time affects your bottom line. `https://blog.kissmetrics.com/loading-time/`, 2011. Accessed: 2016-03-31.

[9] R. D. Knabb, J. R. Rhome, and D. P. Brown. *Tropical cyclone report: Hurricane katrina, 23-30 august 2005.* National Hurricane Center, 2005.

[10] J. C. Knight, E. A. Strunk, and K. J. Sullivan. Towards a rigorous definition of information system survivability. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 78–89. IEEE, 2003.

[11] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. 2000.

[12] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1. IEEE, 2004.

[13] M. Lesk. The new front line: Estonia under cyberassault. *Security & Privacy, IEEE*, 5(4):76–79, 2007.

[14] M. Menth and R. Martin. Network resilience through multi-topology routing. In *The 5th International Workshop on Design of Reliable Communication Networks*, pages 271–277, 2005.

[15] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[16] Netflix. How does netflix work? `https://help.netflix.com/en/node/412`, 2016. Accessed: 2016-04-01.

[17] Nextbit. Meet robin. `https://www.nextbit.com/pages/meet-robin`, 2016. Accessed: 2016-04-01.

[18] M. Prokopenko, F. Boschetti, and A. J. Ryan. An information-theoretic primer on complexity, self-organisation and emergence. *Advances in Complex Systems*, 2007.

[19] J. P. Rohrer, A. Jabbar, and J. P. Sterbenz. Path diversification: A multipath resilience mechanism. In *Proceedings of the IEEE 7th international workshop on the Design of Reliable Communication Networks (DRCN)*, pages 343–351, 2009.

[20] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[21] P. Trimintzios. Measurement frameworks and metrics for resilient networks and services: Technical report, european network and information security agency (enisa). Technical report, Tech. Rep., February, 2011.

[22] US-CERT. Understanding denial-of-service attacks. `https://www.us-cert.gov/ncas/tips/ST04-015`, 2009. Accessed: 2016-04-02.

[23] W. Willinger and J. Doyle. Robustness and the internet: Design and evolution. *Robust-Design: A Repertoire of Biological, Ecological, and Engineering Case Studies*, pages 231–272, 2002.

[24] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *Communications Surveys & Tutorials, IEEE*, 15(4):2046–2069, 2013.

Das Baltikum Testbed

Christian Feiler
Betreuer: Daniel Raumer
Seminar Future Internet SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: christian.feiler@tum.de

ABSTRACT
Das Durchführen von Experimenten ist ein wichtiges Hilfsmittel beim Testen neuer Services oder Ermitteln von Daten. Um Experimente reproduzierbar ausführen zu können, wird eine spezielle Umgebung benötigt. Das Baltikum Testbed der Technischen Universität München bietet eine solche Umgebung. Schwerpunkte dieser Arbeit sind vor allem auf den Aufbau des Testbeds und den Ablauf von Experimenten gelegt. Weiterhin werden die Möglichkeiten des Testbeds aufgezeigt und bisherige Einsatzgebiete erläutert. Zusätzlich werden verschiedene Nutzungs- und Nutzerstatistiken für das Testbed in Anlehnung an andere Testbeds vorgestellt.

Keywords
Testbed, Experiment, Statusseite, Statistiken

1. EINLEITUNG
Bei vielen neuen Services und Netzwerktechnologien sind Tests notwendig, um sämtliche Features zu analysieren. Bei diesen Tests ist es eine Erleichterung, wenn sie durch eine Umgebung kontrolliert ausgeführt werden können, welche es auch erlaubt, Experimente unter den gleichen Bedingungen beliebig oft auszuführen. Das Baltikum Testbed ist ein Beispiel für eine solche Umgebung, die auch komplizierte Experimente – wie sie zum Beispiel bei Multiprozessor-Anwendungen zustande kommen – realisieren kann. In über 15 Publikationen wurden Messwerte dieses Testbeds zitiert [6]. Im Gegensatz zu Simulatoren besteht ein Testbed aus realer Hardware und liefert somit authentische und realistische Ergebnisse. In Testbeds können eine Vielzahl an Messdaten erfasst werden, die zur Analyse von Services benötigt werden – unter anderem die Datenrate zwischen Netzwerkkomponenten oder die Ressourcen-Auslastung der Komponenten selbst durch den Service. Wie es im Baltikum Testbed ermöglicht wird, solche Daten zu erfassen, wird in Kapitel 2 erläutert, wobei sich über 30 studentische Arbeiten mit der Implementierung und Verbesserung der Funktionalität beschäftigen [6].

2. TESTBED ÜBERSICHT
Bevor der Ablauf von Experimenten und die verwendeten Technologien erläutert werden, ist es notwendig, sich mit dem Aufbau des Baltikum Testbeds vertraut zu machen.

2.1 Aufbau
Für die Ausführung von Tests stehen im Baltikum Testbed zehn Hosts mit verschiedenen Hardware Komponenten zur Verfügung. Wie dem passwortgeschütztem Wiki des Testbeds [3] entnommen werden kann, verfügen die Hosts jeweils über mindestens 16 GB Arbeitsspeicher und über eine CPU der Intel Xeon Reihe mit mindestens vier Prozessorkernen. Die Netzwerkkarten variieren je nach Host zwischen Kupfer- und Glasfaseranschluß und unterstützen mindestens 1 GBit/s – bei den meisten Hosts sind es 10 GBit/s. Somit kann eine Vielzahl an Experiment-Szenarien abgedeckt werden. Zusätzlich besteht für die Rechner die Möglichkeit, den Energieverbrauch während eines Experiments aufzuzeichnen. Daher können nicht nur Daten wie zum Beispiel CPU Auslastung, Paketrate oder Latency, sondern auch der Stromverbrauch einzelner Softwareabläufe gemessen werden. Die Messung des Energieverbrauchs erfolgt dabei über ein zentrales Messinstrument, das für jeden Host die Stromaufnahme misst [2]. Da Virtualisierung eine beliebte Rolle in der IT einnimmt [11], beschränken sich Experimente nicht nur auf die zehn physikalischen Hosts: Es können auf jedem Host Virtuelle Maschinen definiert werden. Außerdem stehen einem Benutzer des Testbeds verschiedene Betriebssysteme wie Linux, Windows oder FreeBSD für die Rechner zur Verfügung.
Neben der zehn Server und eines konfigurierbaren Switches für die flexible Verbindung der Hosts beinhaltet das Baltikum Testbed noch einen weiteren Host, der für Management Aufgaben zuständig ist. Auf diesem zentralen Host werden Experimente von Benutzern definiert, gestartet und im Anschluß die Ergebnisse eines Experiments abgelegt.
Die Hosts sind über mehrere Testnetzwerke untereinander und über ein Managementnetzwerk zentral verbunden.

2.2 Ablauf eines Experiments
Der Management-Rechner spielt eine zentrale Rolle bei Experimenten. Denn um ein neues Experiment zu definieren, muss auf diesem ein Konfiguration mit allen nötigen Informationen erstellt werden. Wichtige Inhalte könnten folgende sein:

- Konfiguration der (physikalischen) Hosts:
  - Netzwerkinterfaces
  - Betriebssystem Image
  - Skript
- Konfiguration der virtuellen Hosts/Maschinen: analog zu den physikalischen Hosts

doi: 10.2313/NET-2016-09-1_03

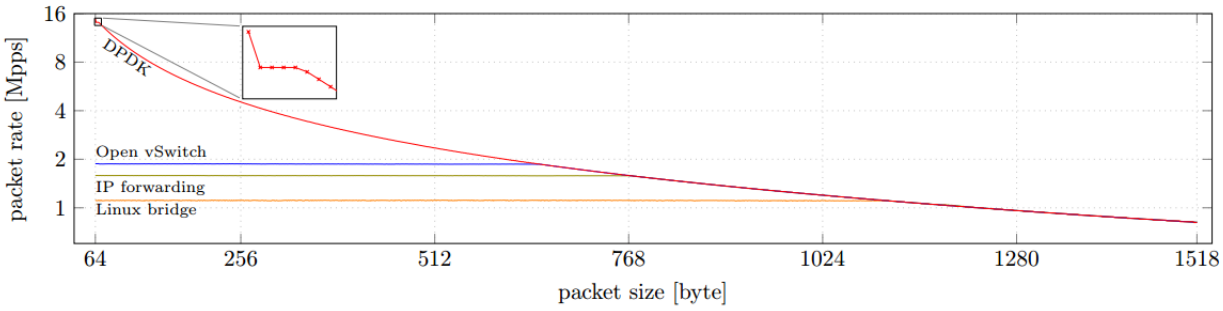


Figure 1: Vergleich der Datenrate verschiedener Frameworks. Aus [10]

- Konfiguration des Switches

Das Skript für die jeweiligen Hosts ist dabei ein Bash-Skript, das den Instruktionsablauf eines Hosts während eines Experiments beinhaltet. Für jeden verwendeten Rechner wird somit ein Skript benötigt. Um einen reproduzierbaren und synchronisierten Experimentablauf zu garantieren, gibt es die Möglichkeit, in diesen Experiment-Skripten Synchronisationspunkte einzubauen, welche durch Namen unterschieden werden. Wird ein Synchronisationspunkt erreicht, fährt der Host erst mit der Ausführung seines Skripts fort, wenn jeder andere Host den Synchronisationspunkt erreicht hat. Die Synchronisation erfolgt dabei über das Managementnetzwerk durch Anfragen an den zentralen Server.
Hat man nun die Konfiguration erstellt, kann man das Experiment starten, was folgende Schritte auf dem Managementhost als Resultat hat [3]:

- Die Konfigurationsdatei wird eingelesen.
- Es wird eine iPXE Konfiguration für die Testrechner angelegt. iPXE ist dabei ein Bootloader, der auf den Hosts installiert ist und die Möglichkeit schafft, das Betriebssystem als Image über ein Netzwerk zu beziehen.
- Es werden verschiedene Konfigurationsdateien und Skripte erzeugt, die die involvierten Hosts vor und während des Experiments über einen Apache Webserver vom Managementhost downloaden und für die Ausführung benötigen.

Nach diesen initialen Schritten erfolgt die Kommunikation über das Managementnetzwerk, wobei die Aktivitäten durch den Manangementserver angestoßen werden:

- Der Switch wird bei Bedarf mittels SNMP konfiguriert.
- Den Hosts wird mithilfe IPMI mitgeteilt, über das Netzwerk mittels iPXE zu booten. IPMI ist eine Schnittstelle, die das Fernsteuern von Rechnern auch im ausgeschalteten Zustand erlaubt.
- Die Hosts bekommen über DHCP eine IP Adresse für das Interface im Managementnetzwerk zugewiesen und laden anschließend mithilfe der vorher erstellten iPXE Konfiguration das Betriebssystem.
- Nachdem das Betriebssystem gestartet ist, laden die Test-Hosts die erzeugten Skripte – unter anderem das Experiment-Skript – und führen das Experiment-Skript aus.

Sobald ein Rechner mit der Ausführung seines Skripts fertig ist, meldet er dies dem Manangementserver mithilfe des Apache Webservers. Während des Experiments werden Dateien mit detailierten Informationen und Ergebnissen von den Hosts auf den zentralen Server hochgeladen. Wie viele Dateien geuploadet werden und was in den Dateien genau enthalten ist, muss im jeweiligen Skripte bestimmt werden.

3. EXPERIMENTSCHWERPUNKTE

Nachdem das Testbed an sich bekannt ist, stellt sich die Frage, für was das Testbed konkret verwendet wird. In der Einleitung dieser Arbeit wurde das Testen neuer Netzwerktechnologien als Beispiel genannt. Eine dieser Technologien sind Software Defined Networks (SDN's). Bei diesen Netzwerken sind die Control- und die Data-Ebene nicht mehr aneinander gebunden im Vergleich zu traditionellen Netzwerken. In diesem Fall schaffen Experimente eine geeignete Grundlage für den Performance-Vergleich der Netzwerke. Eine konkrete Implementierung von OpenFlow Switchen, die Einsatz bei SDN's findet und auch im Baltikum Testbed ausgiebig getestet wird [9], ist Open vSwitch. OpenFlow ist dabei ein Kommunikationsprotokoll, das eine Schnittstelle zwischen Control- und Data-Ebene ermöglicht.
Das Baltikum Testbed wird aber nicht nur zum Analysieren bereits entwickelter Technologien verwendet, sondern auch zum Erstellen neuer Services. Ein Beispiel hierfür ist der Packet Generator MoonGen [8]. Der Generator verwendet dabei das Network Framework DPDK, welches für schnelle Paket Verarbeitung optimiert ist.
Im Zusammenhang mit DPDK ist es auch interessant, dieses Framework mit anderen Frameworks für Paketweiterleitung zu vergleichen, da vermehrt auch handelsübliche Hardware statt Spezialhardware für den Einsatz als Netzwerkgerät verwendet wird [10], man aber möglichst ähnliche Performance erzielen will. Abbildung 1 zeigt als Beispiel das Ergebnis mehrerer Experimente, in denen Linux bridge, IP forwarding, Open vSwitch und Intel DPDK als Frameworks für Paket-Weiterleitung verglichen werden.
Für dieselben Experimente zeigt Abbildung 2 die zugehörige CPU Last Verteilung.
In Kapitel 2.1 wurde bereits angesprochen, dass auf den

doi: 10.2313/NET-2016-09-1_03

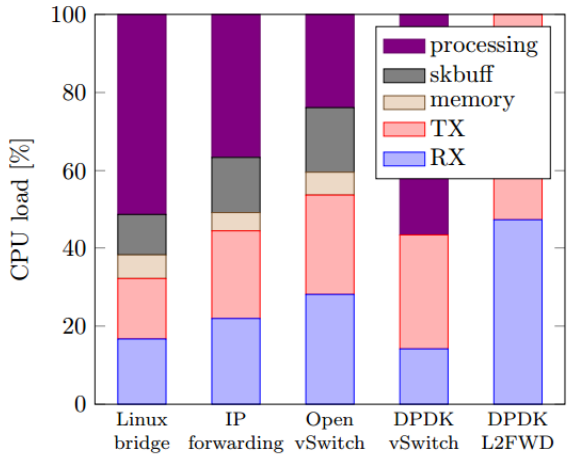


Figure 2: Vergleich der CPU Lastverteilung verschiedener Frameworks. Aus [10]

Hosts im Testbed verschieden Betriebssysteme verwendet werden können. Dadurch lassen sich viele Vergleiche der Betriebssysteme durch Experimente erstellen, wie zum Beispiel der Vergleich der generelle Netzwerkleistung auch in Hinblick auf Energieverbrauch [1].

4. NUTZUNGSSTATISTIKEN

Teil dieser Arbeit soll es auch sein, eine Statusseite für das Testbed mit aussagekräftigen Statistiken zu erstellen, mit denen sich die Nutzung des Testbeds analysieren lässt. Dazu ist es hilfreich, die Features der Statusseiten anderer Testbeds zu analysieren.

4.1 Statistiken anderer Testbeds

Die drei Testbeds, deren Statistikseiten im Weiteren erläutert werden, beinhalten das Emulab, das PlanetLab Europe und das FIT IoT-Lab. Diese Testbeds sind wesentlich größer dimensionert als das Baltikum Testbed, da sie zwischen 300 [7] und 2700 [5] Nodes beinhalten. Im Unterschied zum Emulab und zum PlanetLab Europe besteht das FIT Iot-Lab nicht aus normalen Servern, sondern aus Wireless Sensor Nodes, da es ein Internet of Things Testbed ist. Die unterschiedlichen Größen und Einsatzgebiete der Testbeds haben jedoch wenig Einfluß auf die Gestaltung der Statistiken, sodass die drei Statistikseiten ohne Bedenken als Orientierung für eine eigene Seite hergenommen werden können.

4.1.1 Emulab

Die Statusseite des Emulabs [4] beinhaltet sechs Verlaufsdiagramme, die einen Überblick über die Zahl der verfügbaren Nodes schaffen. Die Diagramme zeigen sowohl den kurzfristigen Verlauf als auch den langfristigen Trend der letzten zehn Jahre.
Zusätzlich gibt es auf der Emulab-Website noch eine Liste über aktuell laufende und kürzlich abgeschloßene Experimente. Detaillierte Informationen zu den Experimenten gibt es leider nicht.
Außerdem kann man auf der Emulab Website eine Weltkarte finden, auf der abgelesen werden kann, wie viele Nutzer des Testbeds es in welcher Stadt gibt. Zwar macht diese Statistik wegen der geringen Größe wenig Sinn für das Baltikum Testbed, dennoch schafft sie einen interessanten Einblick in das Emulab.

4.1.2 PlanetLab Europe

Im Vergleich zur Emulab-Statusseite beschränken sich die Statistiken nicht nur auf den Verlauf der verfügbaren Nodes, sondern geben einen breiten Einblick in die Nutzung des Labors. Zum einen gibt es unter anderem ein Verlaufsdiagramm zur Anzahl der Host-Standorte, der Nodes [7] und der User. Zum anderen gibt es auch Diagramme zum aktuellen Status der Nodes, der Verteilung der CPU-Architekturen oder der verwendeten Betriebssysteme.
Die Statistiken sind jedoch nur beschriftet und werden nicht beschrieben, sodass oftmals nicht ersichtlich ist, was ein Diagramm genau aussagt. Es gäbe nämlich auch Statistiken zur Auslastung der Nodes, jedoch ist es nicht erkennbar, ob es sich um einen Durchschnittswert handelt und welche Nodes für die Statistiken herangezogen wurden.

4.1.3 FIT IoT-Lab

Im Vergleich zum Emulab und PLanetLab besteht das FIT IoT-Lab aus wesentlich mehr Nodes – aus über 2700 Wireless Sensor Nodes [5].
Dennoch ist auf der Website [5] im Reiter „Activity“ unter anderem ein Gantt-Diagramm abrufbar, welches detaillierte Informationen zu jedem Gerät liefert: Wann war der Node mit welchem Experimenten beschäftigt? Außerdem sind Informationen über aktuell laufende Experimente und über den Status der Nodes verfügbar.
Es ist jedoch auffällig, dass der Aufruf von Statistiken teilweise mit langen Wartezeiten (über 20 Sekunden) verbunden ist. Zusätzlich sind die Informationen weniger übersichtlich aufbereitet als bei den anderen beiden Testbeds, da keine Verlaufsdiagramme verwendet werden.

4.2 Baltikum Testbed Statusseite

Ziel dieser Arbeit ist es, die Vorteile der oben beschriebenen Statusseiten zu verbinden und die negativen Auffälligkeiten zu umgehen. Dazu sollen auf der Statusseite sowohl Live-Informationen wie der Status der Nodes (siehe IoT-Lab) und Verlaufsstatistiken über die Nutzung (siehe Emulab) angezeigt werden. Zwar ist es schwierig Statistiken zur Auslastung der Nodes zu erstellen wie beim PlanetLab Europe, jedoch soll es auf der Baltikum Testbed Statusseite einen Graphen mit der aktuellen Netzwerktopologie geben. Aufgrund der geringen Größe des Testbeds kann das übersichtlich realisiert werden. Um die Statistiken zu realisieren, werden im Backend Benutzerdaten und Experimentdaten gesammelt, die durch das Webfrontend mithilfe einer REST-API erfragt und dann visualisiert werden.
Außerdem sollen die Statistiken selbsterklärend oder beschrieben sein und nicht lange (unter fünf Sekunden) zum Laden brauchen.

4.2.1 Webserver

Das Backend ist in Python mithilfe des Web Frameworks Pyramid realisiert und muss auf dem Managementhost ausgeführt werden. Der Server muss hierbei zum einen wie für einen Webserver typisch statische Dateien übertragen, damit der User das Webfrontend laden kann, zum anderen muss er spezielle HTTP-Requests erkennen, die mit Statusinforma-

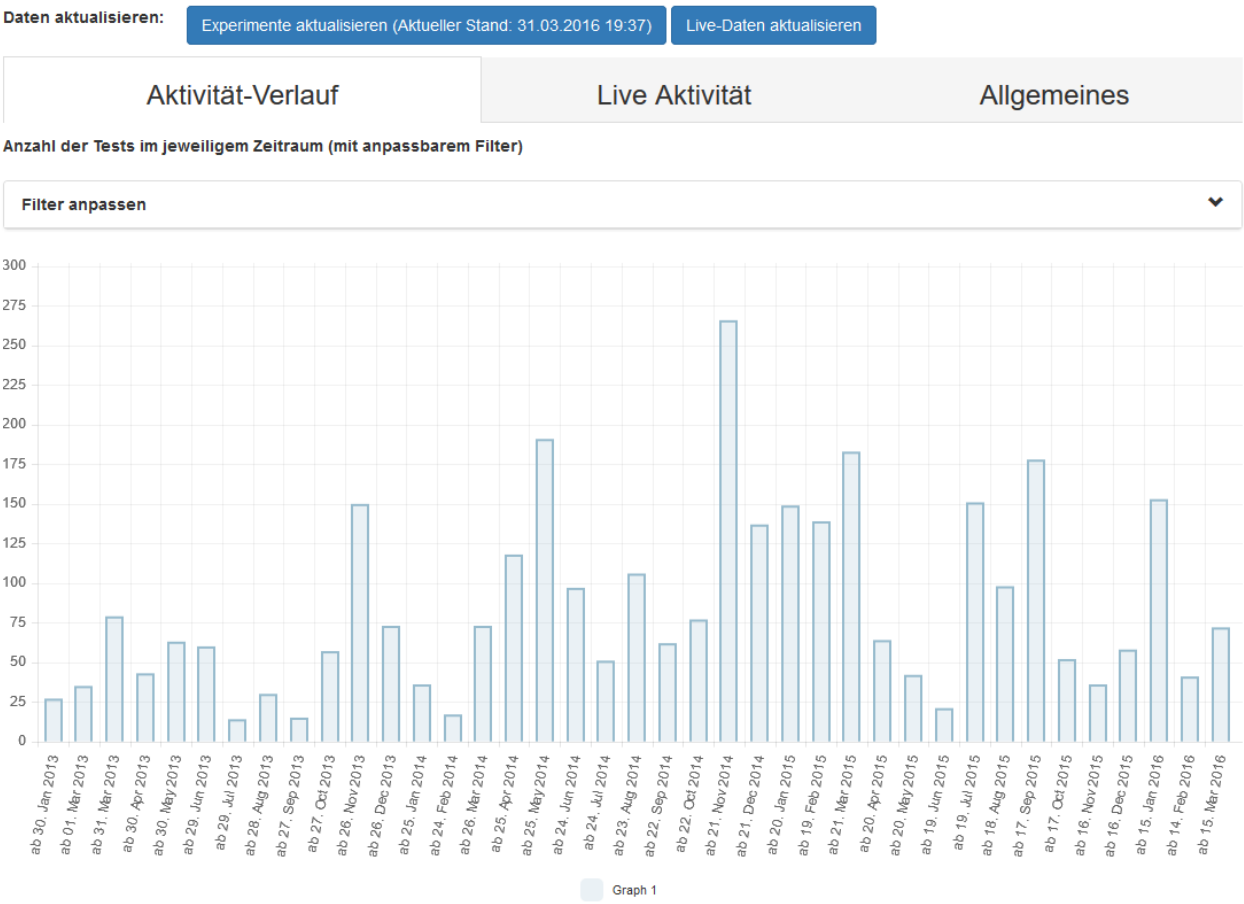


Figure 3: Die Startseite der Statusübersicht des Baltikum Testbeds. Die Daten können durch die blauen Buttons aktualisiert werden.

tionen beantwortet werden müssen. Diese HTTP-Requests sind folgende:

- `/api/last_logins`
  Der Output des Shell-Befehls `last` auf dem Managementserver wird zurückgesendet. Die Antwort beinhaltet also die letzten Logins auf dem Management-Host des Testbeds.
- `/api/logged_in_users`
  Als Antwort werden die Benutzernamen der User gesendet, die aktuell auf dem Testbed eingeloggt sind.
- `/api/host_status`
  Das Ergebnis dieser Anfrage beinhaltet, welche Hosts gerade eingeschaltet sind und ob sie im Moment an einem Experiment beteiligt sind.
- `/api/tests_sorted_by_date`
  Die Antwort beinhaltet alle Experimente als JSON-Objekt. Die Experimente sind dabei nach dem Datum und der Zeit sortiert, an dem sie abgeschlossen wurden. Zusätzlich werden zu jedem Experiment neben dem Datum und dem Namen noch die beteiligten Hosts und der Nutzer, der das Experiment ausgeführt hat, mitgesendet. Alle diese Informationen wurden zu einem vorherigen Zeitpunkt aus dem Ergebnisordner des zentralen Hosts ausgelesen.
- `/api/updated_tests_sorted_by_date`
  Das Resultat hat die gleiche Struktur wie Antworten auf `/api/tests_sorted_by_date`. Allerdings werden die Informationen zu den Experimenten neu ausgelesen und für spätere `/api/tests_sorted_by_date` Anfragen auf dem Server gespeichert. Die Antwort auf diesen Request kann mehrere Sekunden brauchen, da alle Experimentordner – insgesamt sind es mehrere tausend – im Ergebnisordner analysiert werden müssen.

Die statischen Dateien beinhalten zum Großteil den Quellcode für das Frontend. Jediglich die Datei `/connections.json` wird vom Frontend flexibel ähnlich zu den oben genannten Requests nachgeladen, da sie die aktuelle Netzwerktopologie des Testbeds beinhaltet.
Für die Umsetzung des Servers hätten auch andere Programmiersprachen wie Java verwendet werden können, Python bot sich aber wegen der einfachen Programmierung an.

4.2.2 *Webfrontend*

Mithilfe der oben genannten Informationen kann das Webfrontend visualisierte Ausgaben produzieren. Da AngularJS

doi: 10.2313/NET-2016-09-1_03

eine gute Strukturierung des Frontends erlaubt und über viele unterstützte Bibliotheken verfügt, wurde es im Frontend zusammen mit HTML verwendet. Zur Visualisierung werden die Bibliotheken vis.js und Chart.js benutzt.
Die Statusseite gliedert sich in drei Tabs. Im ersten Tab ist ein Verlaufdiagramm enthalten, dass die Anzahl der Experimente visualisiert. Dabei kann nach dem Benutzer, der die Experimente ausgeführt hat, und/oder nach den Hosts, die an den Experimenten beteiligt waren, gefiltert werden. Außerdem kann der Betrachtungszeitraum angepasst werden. Diese Seite gibt also einen guten Überblick über die Aktivität auf dem Testbed in der Vergangenheit, kann aber genauso hergenommen werden, um detaillierte Informationen zu erhalten – ähnlich zum Gantt Chart des Iot Lab. Abbildung 3 zeigt das erste Tab, welches zugleich die Startseite ist.
Im zweiten Tab können im Gegensatz dazu Live Aktivitäten betrachtet werden. Das beinhaltet aktuell eingeloggte Nutzer und den Status der einzelnen Hosts.
Das letzte Tab zeigt verschiedene Diagramme, die Daten wie die Anzahl der Hosts pro Test visualiseren und dabei keinen Verlauf berücksichtigen. Die Topologie des Netzwerks ist auch auf diesem Tab enthalten.
Da das Aktualisieren der Daten sehr lange (mehrere Sekunden) dauern kann, werden Aktualisierungen der Daten nur durch explizite Anfragen des Users durchgeführt. Die expliziten Anfragen werden durch Buttons ausgelöst, wie sie in Abbildung 3 sichtbar sind. Die Live-Daten werden zusätzlich beim Laden der Website geupdatet.

5. AUSBLICK

Wie der erstellten Statistikseite bzw. der Abbildung 3 entnommen werden kann, wurden bereits mehrere tausend Experimente im Baltikum Testbed erfolgreich durchgeführt. Jedoch ist die Entwicklung des Testbeds trotz der hohen Experimentezahl lange nicht beendet. Zum Beispiel wird das Erstellen und Konfigurieren von Experimenten durch eine REST-API erleichtert werden [12]. Zusätzlich sind der Seite des Lehrstuhls Netzarchitekturen und Netzdienste [6] einige Ausschreibungen für Studentenarbeiten zu entnehmen, die sich mit dem Testbed beschäftigen.

6. REFERENCES

[1] Ausschreibung Studentenarbeit: Comparison of Windows and Linux Networking Performance. `http://www.net.in.tum.de/fileadmin/bibtex/publications/theses/2015_9_16-Windows2.pdf`, April 2016.

[2] Ausschreibung Studentenarbeit: Smart Meter. `http://www.net.in.tum.de/fileadmin/bibtex/publications/theses/2015_06_12-MEMPHIS_SmartPDU.pdf`, April 2016.

[3] Baltikum Testbed Wiki. `https://projects.net.in.tum.de/projects/baltikum/wiki`, März 2016.

[4] Emulab Node Usage Statistics. `https://www.emulab.net/node_usage/`, März 2016.

[5] FIT IoT Lab. `https://www.iot-lab.info/`, März 2016.

[6] MEMPHIS / Baltikum Testbed. `http://www.net.in.tum.de/de/projekte/dfg-memphis/`, April 2016.

[7] PlanetLab Europe Statistics. `http://stats.planet-lab.eu/?page=1`, März 2016.

[8] P. Emmerich, S. Gallenmueller, D. Raumer, F. Wohlfahrt, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference*, 2015.

[9] P. Emmerich, D. Raumer, F. Wohlfahrt, and G. Carle. Performance Characteristics of Virtual Switching. In *International Conference on Cloud Networking*, 2014.

[10] P. Emmerich, D. Raumer, F. Wohlfahrt, and G. Carle. Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems. In *Fourteenth International Conference on Networks*, 2015.

[11] Stefan Kreuzer. Automation of Virtual Machine Setups for Network Experiments. Bachelor Arbeit, Technische Universität München, 2015.

[12] Tobias Betz. Optimierung des Ablaufs von testbedbasierten Netzwerkexperimenten. Bachelor Arbeit, Technische Universität München, 2015.

Virtual Switching in Windows

Maximilian Endraß
Betreuer: Daniel Raumer, Sebastian Gallenmüller
Seminar Future Internet SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: endrass@in.tum.de

ABSTRACT
In the field of virtualization, network virtualization tries to ensure mobility and flexibility of virtual machines. One of the key concepts of network virtualization is *virtual switching*, i.e. emulating the behavior of hardware switches in software for general purpose servers or computers; two of those virtual switches are Microsoft's Hyper-V virtual switch, used in the Hyper-V hypervisor and Open vSwitch which also supports Hyper-V (among other hypervisors). In this paper we are going to analyze and compare them with focus on aspects relevant for performance and evaluate the functionality and performance aspects of running Open vSwitch as a virtual switch for Hyper-V.

Keywords
Virtual switch, Open vSwitch, Hyper-V Virtual Switch, SDN, Network virtualization, OpenFlow

1. INTRODUCTION
Network virtualization is meant for datacenters to consolidate their networks to make them more manageable and enable "cloud" technologies. Virtual switches are a centerpiece of this approach as they begin to include more and more functionality that was previously reserved to higher-level devices or software and unify network hardware across entire datacenters into "one big switch" [9].

Both for network virtualization and "traditional" single machine virtualized environments, the feature set and the performance of virtual switches integrated into those physical hypervisors is a crucial point of the entire setup, especially for VMs communicating internally (e.g. as web server and database server). Two virtual switches that have undergone important changes during the last years to increase their virtual network and on-host functionality and performance, are Microsoft's Hyper-V virtual switch and the open-source Open vSwitch. They are arguably common software switches on Windows and both take seemingly different approaches on several aspects of their virtual switch functionality, so in the rest of the paper we are going to take closer a look at them

In Section 2.1, we are going to define the concepts of virtual switches in general and then look at them included in virtualized networks in Section 2.2. In Sections 2.3 and 2.4, we are going to analyze Open vSwitch and Hyper-V virtual switch respectively and conclude by comparing their features and performance in a small benchmark in Section 3 and have a look at Open vSwitch running on Hyper-V and (partly) replacing the integrated Hyper-V virtual switch in Section 3.3.

2. VIRTUAL SWITCHES
Before we compare the virtual switch solutions on Windows, we will explain what virtual switches are and in which particular environments they are used.

2.1 Concept
In virtualized environments, virtual machines created and managed by hypervisors (like VMware ESXi, Microsoft Hyper-V, Xen or KVM) share the same physical environment (i.e. they reside on one physical machine) [20]. To be able to communicate with each other (and with the network outside of the hypervisor) with the same protocols that are used in physical networks, the VMs usually have one or more virtual network interface cards (NICs) with their own MACs and IPs which are connected to a port on a *virtual switch* running on the hypervisor (usually on commodity hardware [9]) which provides connectivity to the outside physical network [20]. These integrated software switches were the first forms of virtual switches, which, in their early days, usually only provided very basic functionality and served no purpose other than extending the physical layer 2 network to the VMs [19].

However, with these early virtual switches, there was still a tight coupling between the traditional physical network, the virtual machines, and the hypervisor, which made the provisioning of new virtual machines cumbersome: In order to provide proper network connectivity to the VMs, the configuration had to be done both on the hypervisor virtual switch and the actual physical network the hypervisor was connected to [19]. Furthermore, coupling VMs with physical network segments renders some of the (theoretical) advantages of VMs, like easy scalability and mobility useless because of the effort required to reconfigure the network to redirect the traffic to a virtual machine's new physical location. From these circumstances the concept of *virtual networking* emerged and with it came more sophisticated virtual switches [19].

In virtual networks, virtual switches provide a greater part of the network connectivity for the VMs; actual physical networks are reduced to carrying tunneled packets between hypervisors or to the internet [19]. This allows for a decoupling of virtual networks from physical networks,

doi: 10.2313/NET-2016-09-1_04

while still being able to provide the same feature set as physical networks and a high level of flexibility and modularity [19]. By leveraging the capabilities of modern virtual switches, complex networks can be designed both inside a single hypervisor without needing dedicated network hardware at all, and across multiple hypervisors [21].

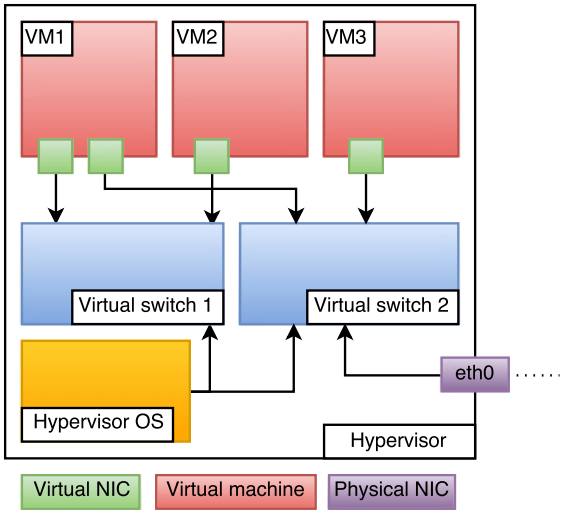


Figure 1: Illustration of a generic virtual switch.

The main task of virtual switches is still very similar to the main task of modern Ethernet switches, i.e. processing Ethernet frames, looking up the destination MAC address of each frame, forward the frames to one or more designated ports, all while learning and maintaining a MAC-to-port forwarding table and avoiding unnecessary transmissions (i.e. not acting as a hub) [21]. But modern virtual switches like Open vSwitch and Hyper-V Virtual Switch are capable of providing features way beyond simple layer 2 packet switching (e.g. router-like and firewall-like L3 and L4 packet filtering, network isolation through VLANs and more) [19]. Fine-grained centralized control over the virtual switches' rules even across different hypervisors is also possible for administrators [19], to be able to e.g. transparently move a virtual machine to a different physical host without network traffic getting lost, by simply assigning the VM to a different virtual port.

Figure 1 shows an example layout of a physical hypervisor running a hypervisor operating system and two virtual switches. Virtual switch 1 is an *internal* switch that connects VM1 and VM2 and the hypervisor OS, while virtual switch 2 is an *external* switch that connects VM1, VM3, the hypervisor OS and the physical NIC which serves as an uplink port to the external physical network. In this example, VM2 has no access to the physical network and VM3 cannot communicate with VM2.

2.2 Software Defined Networking (SDN)

The Open Networking Foundation describes *Software Defined Networking* as a network architecture where the "*control* and *data plane* are decoupled", effectively centralizing the network intelligence and abstracting the network infrastructure from the applications [16]. Virtual switches are a vital part of SDNs: Whereas in a conventional network there is a hierarchical tree structure of Ethernet switches, in an SDN the network "appears to applications [...] as a single, logical switch" [16], greatly simplifying even big network layouts.

The *OpenFlow* protocol "structures communication between the control and data plane" [16]. More specifically, the use of OpenFlow in an SDN allows "direct access to and manipulation of the forwarding (data) plane of network devices", like switches (both virtual and physical) and routers [16], therefore reducing a significant amount of protocols to a single one. Devices only have to understand and process OpenFlow instructions [16]. Therefore, OpenFlow can be compared to "the instruction set of a CPU" [16] and enables centralized and platform-agnostic programmability of networks. This way, OpenFlow can be used to configure simple L2 switching rules as well as more complex L3 or L4 routing and firewall rules.

This centralized networking approach enables a more scalable and flexible network design. For example, in order to transparently move a virtual machine from one physical host to another (as mentioned in Section 2.1), an administrator can simply assign the virtual machine to a different port on the virtual switch in the *control plane*, and the subsequent traffic (in the *data plane*) travelling through the actual physical network will reach its new destination without manually having to adjust different network hardware.

2.3 Open vSwitch

Open vSwitch (OVS) is a "multi-layer, open source virtual switch for all major hypervisor platforms" [19] that can be used both in commodity hardware and in switch hardware [13] and was designed from the ground up with virtual networking in mind. Its design departs from traditional hardware switches and early software switch implementations like the Linux bridge which only provided basic L2 connectivity between VMs and the physical network. Being an OpenFlow virtual switch, OVS was designed for "flexibility and general-purpose usage" [19] to work on many platforms and hypervisors while possibly having to share system resources with the hypervisor operating system and several VMs running on the physical machine. While OVS was originally designed for UNIX-like operating systems and hypervisors running on them (like Xen or KVM) [14], there have been efforts to port it to Microsoft Windows to run with the Microsoft Hyper-V hypervisor, which we cover in more detail in Section 3.3.

2.3.1 Architecture

The three main components of Open vSwitch are the *data-path* in the kernel, the userspace daemon `ovs-vswitchd` and the flow database `ovsdb-server`, also residing in userspace. The userspace daemon is mostly the same across all platforms, while the kernel module is specifically written for the host OS for performance and compatibility reasons [19]. Switch-level configuration, like bridge and interface layouts, are stored by `ovsdb-server` [13] (from which `ovs-vswitchd` pulls its configuration, cf. figure 2).

doi: 10.2313/NET-2016-09-1_04

A basic overview of the functionality of OVS can be seen in Figure 2: A packet from a physical or virtual NIC arrives at the kernel module, then the datapath module checks whether a flow for this packet has already been cached. If the userspace daemon has already instructed the kernel module what to do with the packet (i.e. the respective flow has been cached), the kernel module will simply carry out the specified action (i.e. mostly forwarding the packet to the specified port(s), but other actions like dropping or modifying the packet are also possible). If no flow has been cached, the packet will be forwarded to the userspace daemon, which will then either process the packet locally (e.g. by getting the flow from `ovsdb-server` and then processing it) or outsource the request to a remote OpenFlow controller. Subsequently, the userspace daemon will forward the resulting flow to the kernel datapath, where the specified action can now be carried out and the flow can be cached [19]. In Figure 2 the green line (i.e. the "fast" path) represents the traversal of a packet through OVS, where a corresponding flow has already been cached, which is faster than the traversal path of a packet represented by the red line (i.e. the "slow" path) where the corresponding flow has not yet been cached [19].

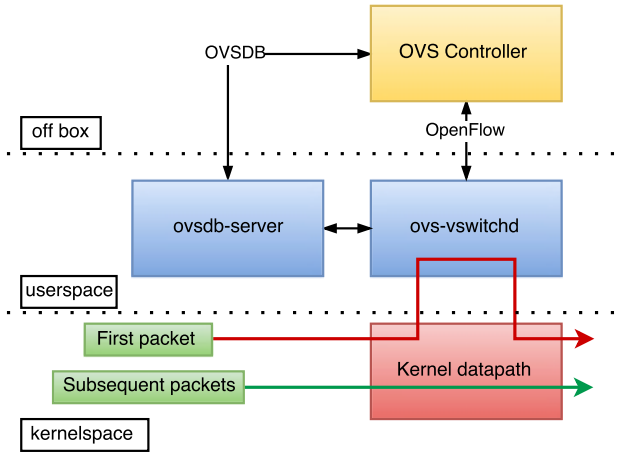


Figure 2: Open vSwitch architecture based on [19].

2.3.2 *Performance optimizations*

Traditional hardware switches are specified to achieve a certain worst-case line-rate performance with dedicated hardware. For OVS however, resource conservation is more critical, since the worst-case performance is secondary to the workload that is running on the user VMs on the hypervisor, and therefore OVS is optimized for common case usage [19] while still being able to gracefully handle worst-case scenarios (like port-scans) e.g. by caching rules.

Caching. Caching takes place in the kernel datapath module, which is effectively separated from the OpenFlow processing. This happens transparently for an OpenFlow controller, which just assumes the packet is going to be matched against the highest priority flow given in the flow table [19].

At first, OpenFlow only used so called *microflow caching*. This means the kernel cache is a simple hash table (without a packet classifier, which now resides in userspace) in which "a single entry exact[ly] matches all packet header fields supported by OpenFlow" [19]. Of course, this leads to cache entries being really specific and usually, with these specific entries only packets of a single transport connection were being efficiently forwarded.

Consequently, the microflow caching approach suffers from a loss of performance in scenarios with lots of short-lived connections, where many cache misses are produced, because for every cache miss, the packet has to be sent to the userspace daemon for classification, resulting in increased latency [19]. Therefore, the concept of *megaflow caching* has been introduced to solve this problem. A megaflow cache is a flow lookup (hash) table with *generic* (or "wildcarded") matching [19]. It mostly resembles an OpenFlow table without priorities. This way, the kernel can terminate the lookup as soon as one match has been found. However, to avoid ambiguity, the userspace daemon is only allowed to install disjoint (i.e. non-overlapping) megaflows in order not to apply the wrong actions because of wrong priorities [19]. To be able to match wildcard header fields while still maintaining a decent level of performance compared to a full packet classification in userspace, OVS performs a so called *staged lookup*: The hashes of four groups (stages) of the packet header (in decreasing order of granularity: metadata, like the switch ingress port, and L2, L3, L4 header data) are calculated and matched against the megaflow cache incrementally. The first hash is calculated only over metadata, the second hash over metadata and the L2 header and so forth [19]. This way, not all packets have to be scanned for L4 headers (e.g. TCP or UDP ports), even if some flows require it. In the end, this solution presents a trade-off, because it requires more hash table lookups than a microflow cache, but might avoid an expensive trip to userspace for packet classification. Microflow caching remains as a "first-level" cache to match a limited number of packets to their designated megaflow cache entry [19].

Packet classification. The wide variety of filtering possibilities offered by Open vSwitch in combination with OpenFlow [15] (packets can be checked against any combination, wildcard or range of MAC addresses, IPv4, IPv6, TCP/UDP header fields, etc.) make packet classification in userspace a rather expensive task on general-purpose processors [19]. OVS therefore introduced a concept called *tuple space search classifier* both for kernel and userspace packet classification [19], although in kernel space there are no priorites and several lookup tables are consolidated into a single table. With this approach, several hash tables are created for different combinations of header fields. For one packet classification (in userspace), every hash table has to be searched and the result with the highest priority will be selected as the resulting flow. Tuple space search classifiers offer constant-time updates (a single hash table operation) and their memory usage scales linearly [19].

Further improvements originate from *tuple priority searching*, where the lookup code is modified in such a way

doi: 10.2313/NET-2016-09-1_04

that tuples (i.e. rows of a flow table) are always sorted in descending order of their priorities. Thus, a lookup can always terminate once a matching flow has been found, since every other flow has to have a lower priority [19].

Lookup of IPs and IP ranges (for IPv4 and IPv6) is based on a trie lookup approach called *prefix tracking* which avoids having to scan all IPs. When traversing the trie (e.g. in Figure 3) starting from the root, the search can be terminated once the current matching node has no more leaves (e.g. in Figure 3 for an IP in `10.2.0.0/16`, the lookup can terminate once it reaches the node 2) [19].

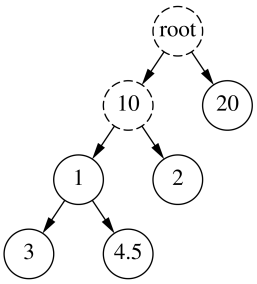


Figure 3: Example prefix tracking tree based on [19].

2.4 Microsoft Hyper-V Virtual Switch

The *Hyper-V role* built into the Windows Server operating system includes a hypervisor, VMbus (a mechanism to enable inter-VM and host-to-VM communication, also used by Hyper-V virtual switch), and, among other components, a "software-based layer-2 Ethernet network switch" [5] called *Hyper-V virtual switch* [10].

2.4.1 Design

Starting from Windows Server 2012, the Hyper-V virtual switch also supports third party extensions [17]. Hyper-V virtual switch (HVS) runs in the management OS (Windows Server or regular desktop editions of Windows starting from Windows 8 [10]) and provides three different kinds of virtual switches [17]:

- An *external switch* which supports a single physical network adapter and an arbitrary number of virtual network adapters, which makes it a comprehensive switch type, connecting all partitions of the host machine as well as the management OS itself (represented by its own virtual network adapter).
- An *internal switch* which supports the same features as an external switch except that it doesn't allow a phyiscal network adapter to be connected.
- A *private switch* which only connects virtual network adapters of Hyper-V child partitions (i.e. virtual machines). That means both physical adapters and the management OS (and applications running on it) can't connect to this switch type.

This design decision mostly serves isolation purposes and thus it is not possible to directly connect two physical network adapters to the same virtual switch. However, VMs can have multiple network adapters, possibly connecting them to multiple different virtual switches.

2.4.2 Packet flow through the virtual switch datapath

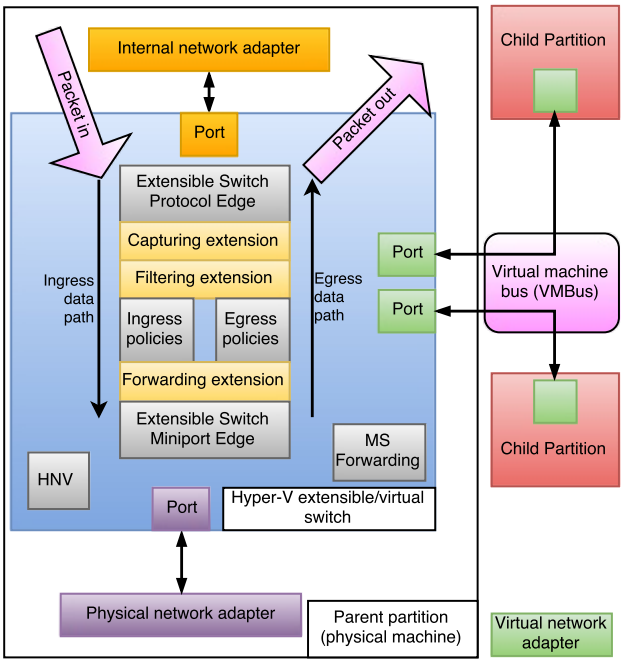


Figure 4: Hyper-V architecture layout based on [17].

The flow of a packet through the Hyper-V virtual switch and its extensions works as follows (cf. Figure 4 for a visual route of a packet) [17, 18]:

1. A packet is first registered at the *protocol edge* which prepares it for the ingress data path by allocating a *context area* that contains out-of-band (OOB) information about the packet, like the source switch port and the origin network adapter.
2. A *capturing extension* can capture and monitor the packet afterwards, but it cannot modify or drop it. It can, however, originate new traffic down the ingress path (e.g. to submit statistics to a monitoring application).
3. The following *filtering extension* is now able to inspect, modify and drop packets.
4. If there is no third-party filtering extension installed or enabled, Hyper-V virtual switch will now apply its built-in ingress policies to the packet before forwarding it to the forwarding extension: Filtering based on Access Control Lists (ACLs), DHCP Guard (to prevent rogue DHCP VM servers), Router Guard (to prevent VMs from pretending to be routers), and more.
5. The *forwarding extension* extends the capabilities of a filtering extension to the process of actually choosing a destination switch port for a packet. If

there is no forwarding extension installed, Hyper-V virtual switch will choose the destination port based on its standard (built-in) settings. This is also the position in the Hyper-V virtual switch data path, where the Windows implementation of Open vSwitch operates [12].

6. Arriving at the *miniport edge*, the Hyper-V virtual switch will apply its built in *port* policies (port based ACLs and Quality of Service, QoS) to the packet and possibly mirror it (if the respective setting has been enabled) by adding additional switch ports to the packet's OOB metadata information. If at this point the packet has not been dropped, it will be forwarded to the *egress* data path.
7. Along the egress data path, the destination switch ports are visible to all extensions. The packet now passes all stations of the ingress data path in reverse order: The forwarding extension can exclude destination switch ports from receiving a packet by removing them from the OOB metadata, then the built-in egress policies are applied to the packet if there are no respective extensions installed.
8. The filtering extension can now drop packets based on the switch destination port. If it modifies the packet, however, it has to clone it and inject it into the ingress data path without destination ports. The subsequent capturing extension can, once again, only analyze the packet and possibly inject new packets on the ingress data path.
9. Then the packet arrives at the overlying protocol edge again and is finally sent to its destination switch ports.

It is possible to install multiple capturing and filtering extensions per instance of Hyper-V virtual switch, but only one forwarding extension [17]. Following the ingress data path, the capturing and filtering extensions cannot see the packet's destination switch ports (because they have not been determined yet). These will later be added to the packet's OOB metadata either by a forwarding extension or by the virtual switch itself, and are visible to extensions along the egress data path.

Other than writing extensions as native NDIS (Network Driver Interface Specification, an API to communicate with network cards on Windows [11]) filter drivers, software vendors can use existing WFP (Windows Filtering Platform, a set of APIs that could previously be utilized by developers to write firewalls or intrusion detection systems for Windows [25]) filter applications to inspect, modify and drop packets along the HVS data path [17]. WFP filter applications operate between filtering and forwarding extensions [26] and usually run in userspace as opposed to NDIS drivers [8]. This possibly enables these kind of network security applications to be transferred from the VMs themselves to a VM-independent level on the underlying network architecture.

2.4.3 Hyper-V network virtualization

Hyper-V network virtualization offers a network virtualization or SDN approach centered around the Hyper-V platform. Hyper-V virtual networks (HVNs, identified by a single *Routing Domain ID*, RDID) "consist of one or more *virtual subnets*" [6] (identified by *Virtual Subnet IDs*, VSIDs). Virtual subnets represent isolated parts within a virtual network, in which VMs within a virtual subnet have their own client IP addresses (CAs) and use them to communicate with each other, while the traffic is transparently transported over the physical network infrastructure using physical addresses (PAs). This ensures that one virtual subnet can in fact be distributed among multiple hosts and subnets on the physical network infrastructure [6].

Virtual subnets allow inflexible applications, that might rely on specific IP and general network configurations, to be brought to the cloud, without having to heavily modify them, because they can be assigned their own virtual network that is isolated from the actual network infrastructure and can thus have IPs that would otherwise collide with IPs already existing on the datacenter network (customers can "bring their own IPs" [24]).

This approach is realized by using NVGRE (Network Virtualization using Generic Routing [3]) to encapsulate layer 2 packets into layer 3 packets (cf. Figure 5) and send them across multiple collision domains in the network [3]. Since this process happens inside HVS, it is transparent for the VMs.

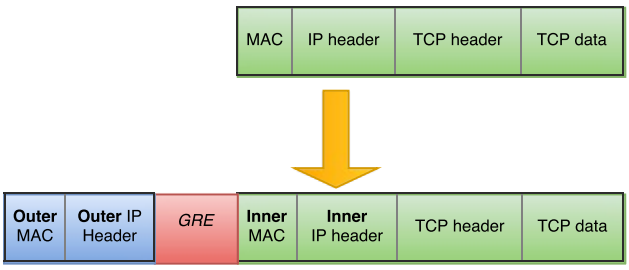


Figure 5: L2 packet encapsulated using NVGRE (based on [3]).

The HNV module inside the Hyper-V virtual switch (cf. Figure 4) now enables extensions to view both the PA and the CA of an encapsulated packet, as it encapsulates and decapsulates NVGRE traffic [6]. That means extensions will receive the encapsulated NVGRE packet on the ingress path and the decapsulated packet on the egress path (because the HNV module operates alongside the miniport edge, cf. Figure 4) or the other way around if the packet is not received from a tunneled connection, but is to be sent to one [18].

doi: 10.2313/NET-2016-09-1_04

3. COMPARISON

Now that we know details about the respective design and datapath layout of both switches, we will compare them in the following Sections.

3.1 Design

Open vSwitch follows a more generic design idea, focusing on the power and programmability of the standardized OpenFlow protocol acting as an "instruction set" [16] of the switch, while the Hyper-V virtual switch enforces a more traditional, pipelined approach of packet processing, although in recent versions it can be extended by filtering and even forwarding extensions (see Section 2.4.2), basically taking control of the switch forwading logic. The current built-in ACLs of Hyper-V virtual switch offer matching for basic header fields (protocol, source & destination MAC, IP, TCP/UDP ports) [23], while OpenFlow basically matches almost all possible packet header fields [15] and offers more actions than the standard "accept"/"deny" (e.g. forwarding to a remote controller). Though it is not as straight forward, functionality like this can be added to HVS through extensions. In fact, in Section 3.3 we are going to discuss an OpenFlow-enabled OVS implementation for HVS. Since OVS is Open Source [14], obviously it can also be extended as such, though it doesn't offer dedicated extension support.

Both virtual switches also offer their own take at the SDN approach, with OVS based around OpenFlow and a centralized OpenFlow controller (see Section 2.2), and HVS based on the concept of "Hyper-V virtual networks" using NVGRE-tunneled packets to communicate (see Section 2.4.3).

For a concise, tabular comparison of the features of both switches, refer to Table 1.

3.2 Performance

The respective design of both virtual switches also impacts their performance which we will analyze in the following Sections.

3.2.1 Theoretic considerations

While the caching approach of OVS has already undergone several changes, culminating in a generic megaflow cache with a "first-level" microflow cache (see Section 2.3.2), the caching policies of the Hyper-V virtual switch are not thoroughly documented, although there are approaches to optimize performance for it. The 2012 R2 version of Windows Server introduced virtual RSS (Receive Side Scaling) which "enables network adapters to distribute the kernel-mode network processing load across multiple cores" [7] while maintaining cache locality [9] since in the past there have been problems achieving the full speed of high-throughput network adapters ($\geq$ 10 Gbps) [22]. Another performance improvement presented is *IPsec task offloading* which moves the encryption of network packets from the VMs (i.e. the CPU) to physical NICs [22]. For multiple teamed (virtual) NICs, Hyper-V virtual switch offers load balancing to distribute the load among them as equally as possible [23]. Dynamic VM queue (dVMQ) is another feature introduced in Windows Server 2012, which aims to speed up the HVS performance. It queues up traffic for (virtual) NICs by hashing the destination MAC and putting the traffic "destined for a virtual NIC into a specific queue" [22] mostly tied to a single core to avoid unnecessary CPU interrupts by processing the traffic on random cores.

Overall, OVS has fewer steps in its entire pipeline (cache lookup, possible userspace packet classification, cache update, and applying the action specified through the matched flow, see Section 2.3.1) than HVS, where the packet has to possibly traverse three extensions twice (along the ingress and egress path, see Section 2.4.2) in addition to being analyzed by the switch itself at the miniport edge. OVS has a more generic caching approach (megaflow-cache) to ensure decent processing speed, while HVS introduced a lot of specific, often hardware-related techniques, like vRSS or dVMQ.

3.2.2 Practical benchmark

Setup. To compare the performance between Hyper-V virtual switch and Open vSwitch, we tested them to see how they perform under load. The test machine was equipped with an Intel Core i5-2500K CPU (4 cores clocked at 4.1 GHz with no Hyper-Threading). To test Hyper-V virtual switch we used Windows Server 2012 R2 with Hyper-V as a hypervisor, for Open vSwitch we used Debian 8.3 jessie (kernel 3.16) with KVM as a hypervisor (`libvirt` 1.2.9, QEMU 2.1.2). For both tests we used 2 VMs with Debian 8.3 jessie and kernel 3.16 running as guests on the hypervisors and both were assigned 1 CPU core and 1GB RAM each. The virtual network adapters were provided by the respective hypervisors. As a traffic generator we used `trafgen`, and to measure packets on the second virtual machine we used `ifpps`, both part of the `netsniff-ng` package [2]. Trafgen was used to send as many empty identical ICMP packets (42 Bytes per packet, 64 Bytes with padding on wire due to Ethernet frame size requirements) as possible from VM1 to VM2. The reason we used ICMP and not UDP packets was that those did not provoke ICMP "Port Unreachable" replies from VM2, possibly slowing it down.

Measurement. From the result (cf. Figure 6) we can observe that in a direct comparison of 10,000,000 ICMP packets sent with `trafgen` from VM1 to VM2 in each scenario, Open vSwitch achieves an average throughput of 764,696 packets per second (pps) or about 0.76 Mpps, while Hyper-V achieves an average of 554,470 pps or about 0.55 Mpps. Interestingly, HVS's initial throughput is about 0.1 Mpps higher than that of OVS, but the overall throughput of OVS is higher. This can probably be attributed to the caching strategy of OVS, where after about 1 second the megaflow cache has been set up and packets do not have to pass the userspace anymore. HVS's caching strategy is not documented, but it seems to employ one nevertheless, given that the initial performance rises, fluctuates and even slightly drops at 10s, but then settles at about 0.6 Mpps.

Table 1: Feature comparison

Feature	Open vSwitch	Hyper-V virtual switch
Pipeline	Short, generic, programmable with OpenFlow	Long, arbitrary, with 3 extensions
Extensions	Open source extensibility	Extension API with 3 types
UNIX integration	Designed for UNIX-based operating systems	n/a
Windows integration	Beta port with increased managerial complexity	Designed for Windows Server
Caching strategy	Megaflow and Microflow cache	n/a
SDN approach	OpenFlow integration	Hyper-V virtual networks with NVGRE

The performance comparison was meant to be carried out in a very similar environment, and while the hardware, the VM configuration and VM software were the same during both tests, the fact that different operating systems and hypervisors were used (because, unfortunately, as of now, we were not able to test the Windows version of OVS) should be taken into account.

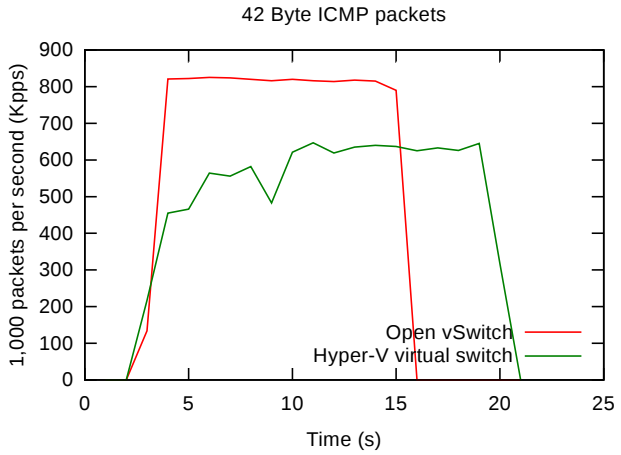


Figure 6: Switch performance comparison

3.3 Open vSwitch running on the Hyper-V hypervisor

Although OVS does not natively support the Hyper-V hypervisor, there is an OVS fork with Windows and Hyper-V support, developed by Cloudbase and VMware [4, 12]. The userspace code was, apart from minor tweaks, mostly left intact, but the kernel module had to be rewritten from scratch due to the fundamentally different architecture of Windows and UNIX-based operating systems. It was designed as an NDIS filter driver (`OVSEXT`) "implemented as a forwarding extension" [12] (cf. Section 2.4.2) so as to leverage the extension framework provided by Hyper-V and replace (almost) the entire built-in switching logic.

In order to make as few changes to the userspace daemon as possible, the developers opted for an emulation of Linux netlink sockets under Windows to let a pseudo device (representing the kernel) communicate with the userspace [12]. Part of the kernel module has also been optimized by using *zero-copy* to only transfer references to the memory location of a packet to userspace instead of deep copying the entire packet [12].

In terms of packet flow, the extension behaves like Open vSwitch: A packet is received by the OVS forwarding extension on the ingress data path, the kernel module of OVS will calculate its hash(es) and look for a cached flow. If there is no such flow, then the packet will be sent to userspace with an upcall, processed there and sent back to the kernel module where the new flow will be installed. Afterwards, the processed packet will be sent "back" onto the HVS egress data path [12]. To keep switch ports in sync between the OVS userspace components, the OVS kernel datapath and HVS, an additional field name has been added to HVS ports which is synchronized between the three components [12].

The inclusion of OVS into Hyper-V as a forwarding extension extends the HVS pipeline even further and thus increases the packet processing overhead and adds the necessity for two basically different virtual switches to stay synchronized. However, since a forwarding extension replaces a substantial amount of default functionality of HVS, we might also see slightly increased performance due to the megaflow cache. Furthermore, it offers datacenter operators a possibility to integrate the Hyper-V hypervisor into their lineup even if their virtual network is reliant on OpenFlow.

4. RELATED WORK

On the topic of virtual switch performance analysis, Emmerich et. al. [9] quantitatively analyze the performance of Open vSwitch compared against other Linux virtual switches or software bridges in a test environment, while Pfaff et. al. [19] look at the real-world performance of Open vSwitch deployed in a commercial Rackspace datacenter. Since we couldn't possibly cover all hardware-offloading features and they are also often subject to change, information on those can be found at the respective virtual switches' websites [5, 14]. A more in-depth look at the SDN features of OVS (also focusing on the integration into OpenStack - a comprehensive software platform for "cloud computing") and HVS is given in talks by Pettit [13] and Williams [24] respectively. The source code and integrated documentation of Open vSwitch [14] and the Windows implementation of OVS [4] is also available. There are, of course, completely different virtual switches with other internals than the two covered in this paper. Commercial solutions by Cisco [1] and VMware [21] might be of particular interest in this case.

5. CONCLUSION

Hyper-V virtual switch has long been the only notable virtual switch for the Hyper-V platform, but its extension

doi: 10.2313/NET-2016-09-1_04

support spawned alternatives, deeply integrated into HVS, and Open vSwitch is among them. While OVS (in its native UNIX-like environment) has an edge on HVS in terms of raw performance (as of 03/2016, cf. Section 3.2), both switches perform well even when faced with several hundreds of thousands of packets. Naturally, OVS is more prevalent in Linux-based environments and datacenters, while Hyper-V and HVS provide several Windows-specific capabilities. The OVS implementation for Windows discussed in Section 3.3 tries to bridge the gap between both worlds, albeit with a slight increase in managerial complexity (due to having to manage two different switches). Nevertheless, this approach can help the consolidation of traditional networks into virtual networks, since it extends the availability of a common protocol to another platform.

6. REFERENCES

[1] Cisco nexus 1000v switch for vmware vsphere. http://www.cisco.com/c/en/us/products/switches/nexus-1000v-switch-vmware-vsphere/index.html, last visited 2016/05/01.

[2] netsniff-ng homepage. http://netsniff-ng.org/, last visited 2016/03/26.

[3] Network virtualization using generic routing encapsulation (nvgre) task offload. https://msdn.microsoft.com/en-us/library/windows/hardware/dn144775(v=vs.85).aspx, last visited 2016/03/17.

[4] Open vswitch for hyper-v (fork). https://github.com/cloudbase/ovs, last visited 2016/03/21.

[5] Hyper-v virtual switch overview, September 2013. https://technet.microsoft.com/en-us/library/hh831823.aspx, last visited 2016/03/16.

[6] Hyper-v network virtualization technical details, June 2014. https://technet.microsoft.com/en-us/library/jj134174.aspx, last visited 2016/03/17.

[7] Virtual receive-side scaling in windows server 2012 r2, October 2014. https://technet.microsoft.com/en-us/library/dn383582.aspx, last visited 2016/03/18.

[8] B. Combs and L. Hernandez. Extending the hyper-v switch, November 2011. https://channel9.msdn.com/Events/BUILD/BUILD2011/SAC-559T, last visited 2016/03/18.

[9] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle. Throughput and latency of virtual switching: A quantitative analysis. 2016.

[10] Hyper-V Overview (Microsoft Technet), March 2015. https://technet.microsoft.com/en-us/library/hh831531.aspx, last visited 2016/03/15.

[11] NDIS Drivers. https://msdn.microsoft.com/en-us/library/windows/hardware/ff565448(v=vs.85).aspx, last visited 2016/03/18.

[12] Open vSwitch 2014 Fall Conference: Open vSwitch on Hyper-V: Design and Development Status, 2014. https://www.youtube.com/watch?v=T5gGD2YQqPc, last visited 2016/03/19.

[13] Open vSwitch Deep Dive - The Virtual Switch for OpenStack, November 2013. https://www.youtube.com/watch?v=x-F9bDRxjAM, last visited 2016/03/14.

[14] Open vSwitch website, March 2016. http://openvswitch.org/, last visited 2016/03/19.

[15] OpenFlow 1.4.0 specification, October 2013. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, last visited 2016/03/19.

[16] OpenFlow Whitepaper, April 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, last visited 2016/03/13.

[17] Overview of the Hyper-V Extensible Switch. https://msdn.microsoft.com/de-de/library/windows/hardware/hh582268(v=vs.85).aspx, last visited 2016/03/18.

[18] Packet Flow through the Extensible Switch Data Path. https://msdn.microsoft.com/en-us/library/windows/hardware/hh582269(v=vs.85).aspx, last visited 2016/03/18.

[19] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association.

[20] H. M. Tseng, H. L. Lee, J. W. Hu, T. L. Liu, J. G. Chang, and W. C. Huang. Network virtualization with cloud virtual switch. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 998–1003, Dec 2011.

[21] VMware Virtual Networking Concepts, 2007. https://www.vmware.com/files/pdf/virtual_networking_concepts.pdf, last visited 2016/03/11.

[22] What's New in Hyper-V Virtual Switch in Windows Server 2012, March 2014. https://technet.microsoft.com/en-us/library/jj679878.aspx, last visited 2016/03/16.

[23] What's New in Hyper-V Virtual Switch in Windows Server 2012 R2, August 2013. https://technet.microsoft.com/en-us/library/dn343757.aspx, last visited 2016/03/16.

[24] C. Williams. Deep dive on hyper-v network virtualization in windows server 2012 r2, June 2013. https://channel9.msdn.com/events/teched/northamerica/2013/mdc-b380, last visited 2016/03/17.

[25] Windows Filtering Platform. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366510(v=vs.85).aspx, last visited 2016/03/18.

[26] Windows Server 2012 Hyper-V Component Architecture Poster. https://www.microsoft.com/en-us/download/details.aspx?id=29189, last visited 2016/03/17.

Diving into Snabb

Dominik Scholz
Betreuer: Paul Emmerich, Daniel Raumer
Seminar Future Internet SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: scholzd@in.tum.de

ABSTRACT

Virtualization techniques are widely deployed in modern computing applications and in recent years became more interesting for the field of networking. Network Function Virtualization (NFV) is used to replace dedicated hardware solutions for networking appliances like switches, routers and packet filters, with software implementations based on virtualized commodity hardware. Today's network providers plan on replacing their backbone infrastructure with this approach to cope with the increasing demands for bandwidth and mobile communication, the Terastream project of Deutsche Telekom, aiming to deploy a cloud based network with NFV, being a lead example. While this approach results in reduced hardware cost and manifold simplifications achieved through the virtualization, both for developing and deploying services, it poses new challenges to comply with complex constraints of mobile networks, primarily achieving carrier grade performance for network appliances, even when running within virtual machines.

The Snabb virtual switch is a packet processing framework that exploits novel design approaches to fulfil and solve the mentioned requirements and problems, resulting in it being a prime solution for NFV.

Keywords

Snabb, Packet Processing, High-Performance, Virtualization, NFV, Ethernet I/O, Virtual Switch, LuaJIT, SIMD

1. INTRODUCTION

Network Function Virtualization (NFV) [4] has become an increasing trend in recent years, applying the concept of virtualization, the dynamic sharing and aggregating of resources [30], which is already deployed in various computing fields today, to the functionality of networking appliances. Specialized hardware solutions have manifold problems, requiring separate middle boxes for each networking task like routing, packet filtering or DoS protection which not only increases the cost but also the complexity of the system when purchasing technology from multiple vendors. Because of the advances of commodity hardware, allowing to perform networking operations on common x86 hardware with almost equal performance compared to proprietary solutions [14], this can be overcome conveniently by providing virtualized software solutions instead.

The NFV architecture is illustrated in Figure 1. The NFV Infrastructure (NFVI) is based on virtualized commodity hardware, while the different functionalities previously offered through dedicated hardware solutions are instead provided by Virtualized Network Functions (VNF) based on the underlying computing, storage and networking framework. The goals of this approach are to guarantee the same requirements in regard to performance, reliability, stability and security, while improving the manageability. This is achieved by completely separating the software from the underlying hardware and providing dynamic and scalable services which can be flexibly deployed to the network [27]. However, several technical challenges have to be overcome, primarily implementing high performance virtualized networking solutions that are portable to allow integration with hypervisors from different vendors [9]. Furthermore, network services can have complex requirements, for instance in regard to maximum delay.

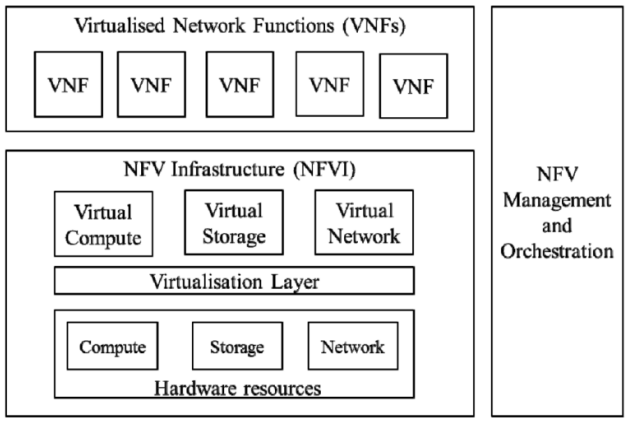


Figure 1: Overview of the Network Function Virtualization Architecture (image taken from [49])

The use cases for NFV technology are manifold, but are particularly interesting for today's telecom core networks [49, 27]. The market situation for network operators has changed in recent years: the demand for mobile communications and Voice over IP (VoIP) has increased, while fixed access lines loose importance [34]. Furthermore, the overall demand for mobile communication and wireless bandwidth is still increasing exponentially, predictions claiming a 10 to 50 fold increase until 2020 [8, 11]. This puts pressure on the network operators to supply scalable and flexible networks which can only be accomplished with a shift to IP-based networks. Deutsche Telekom announced plans to replace their backbone network in the future with a cloud enabled IP network

doi: 10.2313/NET-2016-09-1_05

based on NFV [11] with the Terastream project. By using virtual machines instead of specialized devices and following the "simple - lean - differentiated" [11] approach, the Terastream project aims to not only cope with the increasing demands, but also to reduce complexity and faster deployment of future applications to the customers [11].

Virtual switches are used to virtualize the underlying hardware, for instance, offering virtual Network Interface Cards (NIC) for each network application that make use of one actual NIC [41, 40]. This layer of virtualization has to guarantee the requirements mentioned previously for packet switching and manipulations, primarily carrier grade performance and interoperability for instance with different hypervisors. The Snabb [1] virtual switch [3] is a framework which deploys a novel architecture and exploits virtualization techniques to allow high performance packet processing in user space. Its functionality and goals resulted in a cooperation with the Terastream project to extend Snabb with a dedicated module for NFV, allowing to create VNFs within virtual machines [46]. Therefore, we summarize and explain the core architecture and implemented features of Snabb to understand its compatibility with NFV.

The remainder of this paper is structured as follows. We introduce Snabb, its novel approach in regard to design and architecture, and how to develop and use programs based on this framework in Section 2. The gained insights are compared to other proposals and existing solutions for processing frameworks and NFV in Section 3. In Section 4 the performance of Snabb is demonstrated and briefly analysed by evaluating two programs in a measurement environment. We conclude with a summary in Section 5.

2. SNABB

Snabb [3] is an open source, high-performance networking framework developed by L. Gorrie since 2012. The project's goal is to offer network administrators, Internet service providers and in general operators that have to process large traffic volumes beyond 1GbE software solutions for existing hardware networking appliances [26]. The virtual switch is implemented for modern Linux operating systems, leveraging commodity x86 hardware and Intel NICs [28] running solely in user space [40].

Therefore, Snabb combines new approaches in its core architecture to allow developing of efficient networked applications, which will be illustrated in Section 2.1. The intuitive and easy to use Application Programming Interface (API) as well as code excerpts of sample applications are detailed in Section 2.2.

2.1 Design and Core Architecture

Researchers have demonstrated several bottlenecks of conventional packet processing applications and network stacks in recent years [17, 7, 13], which are in general being avoided and replaced with new techniques by modern high-performance packet processing frameworks (cf. Section 3). Snabb combines these with new approaches to offer fast performing and easy to use network functionalities. The basic paradigm is to focus on simple, small and commodity-targeted solutions instead of complex, large and proprietary ones [5].

The following Sections illustrate the core conceptions that apply these guidelines. The general concept introduced in Section 2.1.1 is also used by other state of the art processing frameworks and not specific to Snabb. A selection of techniques that have not been widely used in networking applications before and are introduced with Snabb are illustrated in Sections 2.1.2-2.1.4.

2.1.1 Kernel Bypass: Ethernet I/O

Various research has shown that using the existing network stack of the kernel, or in general performing networking within the kernel, is not advantageous for a multitude of reasons, including interrupts and context switches for every packet when being passed from the hardware to the kernel and later on to the user space application [17, 7, 13]. Therefore, the trend is to move the complete application to user space, bypassing the kernel. While this achieves the raw performance of the Ethernet device, which can be significantly increased compared to legacy applications [19, 18], it also means that everything that is usually done by the kernel and its inherited features and security measures and guarantees have now to be taken care of in user space.

Kernel bypass networking, which allows to focus on implementing exactly the features beneficial for the application, is a core concept not only chosen by Snabb, but several high-performance packet processing frameworks [17, 16]. The primary idea is to write all low level functionality, the interaction with the hardware, from scratch. Snabb does so for modern Intel NICs. Instead of communicating with the device via the kernel and its drivers, the PCI device is bound directly to Snabb and configured by writing respective values according to the NICs specification using Memory-mapped I/O (MMIO), allowing direct I/O operations. Via Direct Memory Access (DMA) the NIC is allowed to directly access reserved memory regions to read and write packets. In particular, two ring-based FIFO structures for transmission and reception descriptors, storing the addresses of packets to be transmitted or received, respectively, and actual memory for packets are mapped. As the NIC can only operate with physical addresses, *huge pages*, which offer contiguous physical memory blocks of 2 MB or 1 GB, are used for these memory regions [25].

2.1.2 Implemented using LuaJIT

While it is a common approach for packet processing frameworks to focus on the hardware-near C or C++ as primary programming languages and respective established compilers, Snabb uses the lightweight scripting language Lua and the just-in-time LuaJIT [39] compiler instead. Lua is an easy to learn, high-level programming language that has been used in various fields including game development, image processing and robotics [29] for many years. The following illustrates reasons why the use of LuaJIT over another programming language is beneficial for a networking framework.

Firstly, the simple and fast Foreign Function Interface (FFI) allows to easily integrate and work with C libraries [16]. Therefore, Snabb can conveniently make use of low-level

[1] Formerly known as SnabbSwitch

doi: 10.2313/NET-2016-09-1_05

functions and data structures. Secondly, LuaJIT, which produces target specific machine code for instance for x86 and ARM architectures, is a trace based compiler that gathers profiling information during runtime to compile parts of the application in an optimized way [48]. The control-flow is analysed in detail by inspecting whole program paths "to capture the smallest set of execution traces that are representative of the dynamic behavior of the application. Doing so, a trace-based compiler can focus all of its optimization budget on a tiny, yet very important part of an application" [40]. Instead of an one time optimization during traditional static compilation, the just-in-time compiler can adapt the machine code for the current traffic flow or other events, even unexpected ones [40]. These concepts are well researched in the field of compilers [6] and are now applied to networking applications.

Because of these reasons, even low-level code like the driver is written in Lua. Performance evaluation has shown that the results are promising as LuaJIT can outperform applications written in C [35]. The long term goal of Snabb is to drop all references to C functions and structures despite the easy integration with the FFI module and replace them with system-call libraries and assembler libraries written in Lua [5].

2.1.3 Virtualization

Virtualization offers the possibility that real resources can be shared and aggregated amongst multiple virtual applications, optimizing their usage. Furthermore, multiple users using the same physical resource can be isolated to guarantee no interferences and provide security [30]. Applied to virtual machines, networking has to be performed in a fast but also flexible manner, for which only the former applies to hardware NIC virtualization. The Snabb NFV module aims to accomplish both by combining single root IO virtualization (SR-IOV) [32, 12, 36] and a novel virtio technology for the software layer, vhost-user [46].

As Paolino et al. [40] illustrate, Snabb used the pa-ravirtualized I/O framework virtio to connect the kernel-base virtual machines (KVM) with the network controller. This framework is composed of the front end driver virtio-net executed in the kernel of the guest, a backend driver and the emulator module vhost-net, running on the underlying kernel of the host, causing all traffic to go through the virtualized application and the virtual switch. As in this scenario the packets have to pass the kernel of the host, significant overhead is generated by inevitable costly context switches.

vhost-user was implemented to circumvent the host kernel and therefore allow direct communication with the virtual machine. By using Unix domain sockets, with this method in combination with QEMU/KVM, the virtual switch can directly access the shared data structures in the guest's memory space. This allows the transmission and reception of packets without copy operations or context switches, providing carrier grade performance, while also offering flexibility as this is accomplished without specialized drivers. Initially specifically designed for Snabb and its vhost-user App [2] for integration with NFV, Virtual Open Systems has included it in its QEMU release and is now adopted by other virtual switch implementations [40].

The Snabb NFV program exploits these techniques, allowing networking with QEMU/KVM, integrated with or without the OpenStack plugin Neutron at high packet rates [46].

2.1.4 SIMD Offloading

The calculation of, for instance, checksums as they occur in multiple protocols of the ISO/OSI model is a tedious and CPU consuming task, which is why modern NICs allow them to be offloaded to the NIC and calculated in hardware, significantly increasing the performance. Frameworks have started using this approach to improve their performance [16]. However, this comes with the downside that the offloading feature is dependent on the hardware, requiring differing configuration for every NIC, if it even is supported at all.

Snabb uses a different approach as it does not want to be restricted to proprietary hardware features. It makes use of Single Instruction Multiple Data (SIMD) instructions of modern computer architectures. These allow for instruction level parallelism as one instruction leads to the parallel execution of one instruction in multiple functional units. On x86 instruction set architectures the performance of the Advanced Vector Extension (AVX) SIMD instructions has been continuously increased in recent years [23] and is expected to be further improved in the future. Furthermore, SIMD offloading is not depending on the hardware of the NIC, significantly reducing the implementation effort, but also allowing it to be used for other tasks that are usually expensive, like copying of packets [21].

Generalized, the effort of Snabb is to prefer the CPU for tasks over the likes of proprietary NIC features like offloading or Zero-Copy (ZC). With technologies like Intel's Data Direct I/O (DDIO) [1] the data can be directly loaded into the L3 cache of the CPU from which it can easily be accessed and worked with. Even subsequent data copies are cheap, making it possible to use instruction level parallelism over specialized hardware features [5].

2.2 Frontend and Usage

The previous Section illustrated the design approaches of the underlying architecture. Following, the high level architecture and Application Programming Interface (API) which is used by an programmer to develop applications and complete programs are explained.

2.2.1 High-Level API

As already illustrated in the previous Section, Snabb is primarily written in LuaJIT. Scripting languages in general, and Lua in specific, are designed to be easy to learn and use. Using this language not only for the core functionalities but also the end user API allows to write scripts for new programs in an uncomplicated manner. Furthermore, existing features can be understood with less effort and time.

Snabb provides each functionality as a so called "App" in reference to modern App Stores [24]. Their intent is to be

[2] `https://github.com/SnabbCo/snabb/tree/master/src/apps/vhost`

doi: 10.2313/NET-2016-09-1_05

the software counterpart of typical hardware appliances for networking, like an I/O interface, repeater, but also more complex networking devices like switches or routers. Hence, one App represents a specific network function.

Each App has one or more input and outputs links. The typical workflow of an App is to pull packets from an input link, process them according to the scripted task of the App and then push them to an output link. The processing can be anything that translates to actual machine code. These small network functions can then in turn be combined via links, from one App's output to another one's input, to form more complex ones. A link can be compared to an Ethernet cable connecting real network devices. In software it is a ring-based structure used to store the packets between successive network functions [22]. Via this network of separate functions, existing Apps can be reused or, because of the modularized nature, exchanged with new ones [24]. This overseeable and easy to maintain approach allows for the creation of complex network functions based on a network of Apps, all doing a specific task, fitting the NFV approach.

Packets themselves are represented simply by an variable sized array of bytes. This allows to easily work with the data contained as more specific representations, for instance an Ethernet header can be built on top of it [22]. Furthermore, compared to more complex data structures like the *sk_buff* structure used by the Linux network stack, no overhead is created as no additional data, aside from the raw bytes, are stored.

2.2.2 App and App Network Samples

To better illustrate the concepts presented in the previous Section, selected code snippets[3] of the core API functionalities of Snabb Apps and App networks are shown and briefly explained in this Section.

Listing 1 shows the *pull()* function of the Repeater basic App[4]. The purpose of this function is to continuously repeat all packets in the same order received from the input link back to the output link. As described earlier, *push()* is responsible to transmit processed packets back into the network of Apps. In this example, packets are read from the input link and stored in a table (lines 3 sqq.). Then, all gathered packets are replayed back to the output link (lines 7 sqq.).

This general concept of receiving packets, performing arbitrary processing on them and finally transmitting them via outgoing links, like it is with real hardware network devices, is performed by every App of Snabb. Creating an actual program, consisting of a network of these functions, is demonstrated via the *packetblaster* program[5], shown in Listing 2.

The program is used to generate streams of packets, either based on a pcap file or by synthesizing packets with rudimentary configuration options on the fly. The main *run()* function illustrates the general structure of a complete network of functions. The primary task is to generate a configuration for Snabb, containing the definition of the App network, used PCI devices and more. As shown in lines 7 sqq., the program consists of a PcapReader, Repeater and Tee App. The output of the PcapReader is connected with the input of the Repeater (line 10), which in turn is connected to the Tee App (line 11). This App multiplexes the packets to multiple outgoing links, each connected to one of the discovered PCI devices (lines 15 sqq.) as LoadGen App. After the network is configured, the configuration is passed to the Snabb engine and started (lines 25 sqq.), executing the program.

Based on this concept various complex programs have been developed, ranging from the rumpkernel App[6], a complete NetBSD TCP/IP routing stack, to snabbnfv [7], a module for Network Function Virtualization.

```
1  function Repeater:push ()
2     local i, o = self.input.input, self.output.output
3     for _ = 1, link.nreadable(i) do
4        local p = receive(i)
5        table.insert(self.packets, p)
6     end
7     local npackets = #self.packets
8     if npackets > 0 then
9        for i = 1, link.nwritable(o) do
10          assert(self.packets[self.index])
11          transmit(o, packet.clone(self.packets[self.index]))
12          self.index = (self.index % npackets) + 1
13       end
14    end
15 end
```

Listing 1: push() function of the Repeater basic App.

```
1  function run (args)
2     local c = config.new()
3     [...]
4     if mode == 'replay' and #args > 1 then
5        args = lib.dogetopt(args, opt, "hD:", long_opts)
6        local filename = table.remove(args, 1)
7        config.app(c, "pcap", PcapReader, filename)
8        config.app(c, "loop", basic_apps.Repeater)
9        config.app(c, "source", basic_apps.Tee)
10       config.link(c, "pcap.output -> loop.input")
11       config.link(c, "loop.output -> source.input")
12       [...]
13    end
14    [...]
15    pci.scan_devices()
16    for _,device in ipairs(pci.devices) do
17       if is_device_suitable(device, patterns) then
18          nics = nics + 1
19          local name = "nic"..nics
20          config.app(c, name, LoadGen, device.pciaddress)
21          config.link(c, "source."..tostring(nics).."->"
22                         ..name..".input")
23       end
24    end
25    [...]
26    engine.configure(c)
27    if duration then engine.main({duration=duration})
28    else             engine.main() end
29 end
```

Listing 2: Configuration of the App network in the *packetblaster* program

[3] All code snippets are only excerpts from the mentioned sources, shortened and modified for the sake of clarity

[4] https://github.com/SnabbCo/snabb/blob/master/src/apps/basic/basic_apps.lua

[5] https://github.com/SnabbCo/snabb/blob/master/src/program/packetblaster/packetblaster.lua

[6] https://github.com/anttikantee/snabbswitch/tree/rumpkernel

[7] https://github.com/SnabbCo/snabb/tree/master/src/program/snabbnfv

doi: 10.2313/NET-2016-09-1_05

3. RELATED WORK

In addition to the problems induced with virtualizing the hardware, as explained in Section 2.1.3 for the NIC, various performance critical bottlenecks for packet processing frameworks have been assessed. Garcia et al. [17] identified that the allocation of resources on a per packet basis and serialized traffic access are major impediments. Furthermore, the differentiation of user and kernel space implicates multiple copy operations of the packet and costly context switches. They propose improvements to cope with these bottlenecks. Firstly, resources should not be allocated during runtime, but instead preallocated whenever possible. To make best use of the underlying hardware, the whole processing path from receive and transmit queues to applications should be parallelized, whereas each separate path processes a batch of packets at once, reducing per packet overhead. Modern memory mapping techniques in combination with DMA allow for packet processing with zero copy operations.

Multiple frameworks that exploit these features to speed up packet processing have been developed. The Intel Dataplane Development Kit (DPDK) packet processing framework [2] provides drivers and software libraries for Intel based architectures to bypass the data plane of the kernel. It uses optimized data structures implemented in a lockless manner in combination with techniques like memory preallocation and batch processing to speed up processing performance [45]. Furthermore, hardware features of modern Intel NICs are used to offload, for instance, the calculation of checksums to hardware, saving valuable CPU time. One application that is based on the DPDK is the packet generator MoonGen [15]. While focusing on crafting packets and utilizing novel techniques for sub-microsecond timestamping and precise rate control mechanisms, it shows similarities with Snabb in the usage of LuaJIT as front-end scripting language. Packets are generated with Lua scripts, whereby each packet can be separately modified in a flexible way to generate complex streams of traffic. Measurements have shown that MoonGen is capable of saturating a 10 GbE link with minimum sized packets using only one CPU core [16].

The PF_RING ZC (Zero Copy) [38] packet processing framework, developed by ntop, focuses on custom drivers and virtualization. The drivers can be used as replacement for existing kernel drivers, or to bypass the network stack entirely. The Zero Copy technique allows the NIC to directly access memory that the KVM can access, too, removing the overhead normally induced when copying data from the NIC to the kernel and then to user space. This allows for packet processing at 10 GbE even within virtual machines [38].

Specifically aimed at NFV, Martins et al. [33] introduce ClickOS, a platform for virtualized software middleboxes for high performance scenarios with the same goals in regard to flexibility, scalability and performance as NFV. ClickOS utilizes hypervisor virtualization, in particular para-virtualized VMs with the Xen hypervisor, because of its low delay and high throughput. The Click modular router [31] is an architecture intended for creating flexible and configurable software routers at runtime [42] and is used as programming abstraction for ClickOS as it already provides a broad range of networking elements that can be used to create middleboxes, while also being extensible. Instead of running Click in user space or as kernel module, whereas each middlebox is executed within a Linux virtual machine, ClickOS makes adjustments to the Xen hypervisor and implements its own virtual machines based on Click. Briefly explained, each "VM consists of the Click modular router software running on top of MiniOS" [33], whereas MiniOS is a minimized operating system provided by Xen project. Furthermore, utility functions to administer these ClickOS VMs are provided. Extensive and manifold performance analysis of the authors has shown that ClickOS yields the execution of hundreds of middleboxes executing VNFs on commodity hardware, while maintaining low delays. In certain scenarios, a throughput of up to 27.5 Gbps can be reached [33].

Newer versions allow ClickOS to run on top of the packet I/O framework netmap [43], which includes its own virtual switch. Pinczel et al. [42] analysed the performance of a prototype when mapping the proposed architecture of future 5G networks [37] to machines running ClickOS based on netmap. While their model demonstrates good flexibility, Click imposes some limitations in regards to isolation, imposing potential security issues. Furthermore, while the confined programming environment of Click guarantees high performance, integration of third party libraries for the implementation of more complex scenarios is not possible because of restrictions when handling packets, accessing memory or performing I/O operations [42].

4. PERFORMANCE EVALUATION

Snabb provides sample applications itself and even more contributions by the community are accessible in other development branches. In the following Sections two applications were tested in a testbed environment and the results are compared with similar tests of other processing frameworks found in the literature. The measurements were executed in the Baltikum testbed, using a server equipped with a Supermicro X9SCL/X9SCM[8] motherboard, an Intel Xeon E3-1230 v2 CPU clocked at a maximum of 3.3 GHz and an 8 MB L3 cache[9], Dual channel 16 GB ECC DDR3 SDRAM clocked at 1333 MHz and two 82599ES 10-Gigabit SFI/SFP+ NICs[10]. As operating system the Grml Live Linux 2013.02 image with Linux kernel version 3.7 was used. All measurements were performed with the newest release of Snabb, 2016.03 "Tamarillo"[11], unless stated otherwise.

4.1 Traffic Generation: packetblaster

While it is not the primary intent of Snabb, it is able to create and send traffic streams based on pcap files or completely synthesized packets via the *packetblaster* program introduced in Section 2.2.2. As this program is mainly used to test and benchmark other Snabb Apps and programs and therefore should not interfere with such, the packet creation process is different than for other packet generators (cf. Section 3). Instead of allocating and modifying every single packet that is sent, only a few packets are created and written into the transmission queue of the device. When

[8] http://www.supermicro.com/products/motherboard/Xeon/C202C204/X9SCM-F.cfm
[9] http://ark.intel.com/products/65732
[10] http://ark.intel.com/products/32207
[11] https://github.com/SnabbCo/snabb/releases

doi: 10.2313/NET-2016-09-1_05

transmitting a packet, the LoadGen App does not invalidate the buffer, which would free and remove it from the queue, but instead resets the pointer to the DMA memory, basically instructing the NIC to send these packets in an endless loop [20]. This allows the program to generate traffic of multitudes of 100Gbps while only utilizing a small percentage of the CPU. However, this comes at the cost of customizability of the traffic stream. Per default, *packetblaster*, when synthesizing packets, only allows to change Ethernet addresses and the size of IP packets.

The measurement was done with the *packetblaster* program running on the server while the CPU is clocked at 1.6 GHz. The application is instructed to generate as much traffic as possible, crafting minimum sized 60 byte packets[12] using default values and sending them via the two available PCI devices. The output shows that *packetblaster* does so with 10 GbE line rate, reaching the full 14.88Mpps that are theoretically possible for each of the links. Analysing the CPU utilization with the profiling tool *perf* reveals that the program used only one CPU core to do so, however, at a load of 100 %, which would contradict the statement that only a fraction of the CPU is used. This is because the program exploits busy waiting mechanisms[13], exhausting the CPU at all times to avoid the execution of other tasks which would cause costly context switches. Disabling this mechanism and repeating the experiment over a period of 10 seconds with the program pinned to only one CPU core using *taskset* results in approximately 90 million CPU cycles used per second, equalling a utilization of only 5.6%.

For comparison, the dedicated packet generator MoonGen (see Section 3) is able to saturate a 10GbE link using one fully utilized core with customized packets, whereas each successive packet can be uniquely modified [16]. However, it has to be noted that both applications have different purposes: MoonGen aims to generate as much traffic as possible with the given ressources, while Snabb' *packetblaster* is primarily used to test other applications during development and therefore should generate the traffic with the least effort possible.

4.2 Lightweight 4over6: lwAFTR

A more complex program is the implementation of the lightweight 4over6 (lw4o6) architecture [10] lwAFTR. Because the Deutsche Telekom plans to use IPv6-only for their future backbone infrastructure, better tunnelling architectures and protocols that cope with existing problems are necessary to allow efficient encapsulation of IPv4 traffic in IPv6-only networks. This particular solution utilizes 4over6 and NAT. Previously, the primary problem was that keeping the state of the NAT at a centralized point (AFTR) resulted in problems regarding the scalability of the system, because it has to be done per flow. lw4o6 solves this by moving the NAT functionality to the client-side network function (B4) responsible for tunnelling the traffic [10]. The AFTR part only has to maintain a binding of "the [B4's] IPv6 address, the allocated IPv4 address, and the restricted port set" [10]. Therefore, the implementation lwAFTR is a good example to demonstrate the application area of NFV development with Snabb.

Simplified, the *push()* function of the App encapsulates all IPv4 packets coming from the internet and decapsulates all IPv6 packets received from the B4 clients. Neglecting the tasks of handling ICMP packets in a special way or binding lookups in the case of hairpinning [47], the primary operations are memory accesses to copy data when prepending or removing the tunnelling headers, which is a common bottleneck in high-performance packet processing frameworks [17, 45]. The authors claim a performance of 4 Mpps for 550 byte packets resulting in a throughput of 17Gbps on two Intel 82599ES NICs and one core of an Intel XEON CPU clocked at 2.4GHz [5].

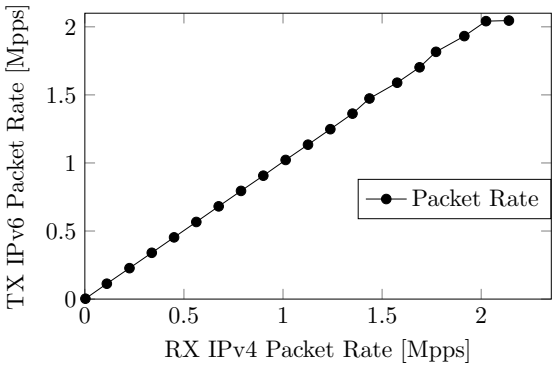


Figure 2: Performance of lwAFTR when encapsulating IPv4 packets

The measurements involved the most recent version of lwAFTR[14] on our test device, clocked at 3.3 GHz. Using another server running the packet generator MoonGen, IPv4 traffic of 550 byte packets matching the binding table with one respective entry was generated and sent to the server running the lwAFTR program. The resulting transmission IPv6 packet rate of the application is illustrated in Figure 2 for different reception IPv4 packet rates. It is clearly visible that the device is able to process all incoming packets until a reception rate of approximately 2.1 Mpps. At this point, the reception rate is approximately 8.4 Gbps with a respective transmission rate of 9.0 Gbps, matching the performance claims presented in [5] mentioned in the previous paragraph.

5. CONCLUSION

In this paper we have introduced the packet processing framework Snabb and its novel design approaches that follow the trends of network function virtualization. It utilizes main-streamed techniques to circumvent common bottlenecks when performing networking operations, for instance bypassing the existing network stack entirely. This can be achieved with DMA and customized NIC drivers. However, instead of using proprietary hardware-based features of modern NICs, like offloading as other frameworks choose to, Snabb uses AVX SIMD instructions to fully utilize the CPU instead, having the advantage of being independent from

[12]The 4 byte *Frame Check Sequence* of the Ethernet frame is appended by the NIC, resulting in a 64 byte frame

[13]See the busywait variable in the file core/app.lua

[14]Commit 2191c16 of `https://github.com/mwiget/snabb/tree/lwaftr_starfruit`

doi: 10.2313/NET-2016-09-1_05

the used NIC. Furthermore, Snabb uses the trace-based LuaJIT just-in-time compiled scripting language based on Lua as primary language, both for back- and front-end programming. Studies have shown that this approach generates optimized code for networking functions on x86 architectures, while the aspect of being an easy to learn and use scripting language eases the development. Because of optimized virtualization techniques, Snabb is eminently for NFV. The novel vhost-user virtio implementation allows packet processing with zero copy operations within virtual machines. These features of the underlying architecture guarantee high-performance processing of packets, while the front-end hides these aspects and provides an easy to use API.

While the project is rather young [15] and the code basis could therefore be considered to still be immature, Snabb enjoys an active community because it is completely independent of vendors, free of license fees and entirely open source. Its activity stretches from mailing lists over websites like reddit to regularly published development blogs, yielding in optimized communication and cooperation amongst projects and developers [26]. The main author publishes a new release every month, steadily extending and improving the core functionality by valuable contributions so that all other developers can incorporate these effortless into their programs [44].

This plays hand-in-hand with the concepts of Apps and programs as a network of Apps. Because of the unified and simple pull and push API of Apps, existing Apps can be reused, while new functionality can be implemented quickly. These small modules, created potentially by many different developers, can then be combined via links to form the complete network. This allows for high use- and reuseability, resulting in a large variety of existing programs.

6. REFERENCES

[1] Intel Data Direct I/O Technology. [Online]. Accessed: March 2016, Available: `http://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html`.

[2] Intel DPDK: Data Plane Development Kit. [Online]. Accessed: March 2016, Available: `http://dpdk.org/`.

[3] snabb-nfv project code base. [Online]. Accessed: March 2016, Available: `https://github.com/SnabbCo/snabb`.

[4] Network Functions Virtualization (NFV); Architectural Framework. [Online], October 2013. Accessed: March 2016, Available: `https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf`.

[5] K. Barone-Adesi. Snabb Switch: Riding the HPC wave to simpler, better network appliances. Talk at FOSDEM16 [Online], 2016. Accessed: March 2016, Available: `http://mirrors.dotsrc.org/fosdem/2016/ua2114/snabb-switch-riding-the-hpc-wave-to-simpler-better-network-appliances.mp4`.

[6] M. Bebenita, M. Chang, G. Wagner, A. Gal, C. Wimmer, and M. Franz. Trace-based compilation in execution environments without interpreters. In *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, pages 59–68. ACM, 2010.

[7] R. Bolla and R. Bruschi. Linux software router: data plane optimization and performance evaluation. *Journal of Networks*, 2(3):6–17, 2007.

[8] I. Chih-Lin, S. Han, Z. Xu, Q. Sun, and Z. Pan. 5g: rethink mobile communications for 2020+. *Phil. Trans. R. Soc. A*, 374(2062):20140432, 2016.

[9] C. Cui, H. Deng, D. Telekom, U. Michel, and H. Damker. Network functions virtualisation. *SDN and OpenFlow World Congress*, 2012.

[10] Y. Cui, Q. Sun, M. Boucadair, T. Tsou, Y. Lee, and I. Farrer. Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture. RFC 7596 (Proposed Standard), July 2015.

[11] Deutsche Telekom. Deutsche Telekom tests TeraStream, the network of the future, in Croatia. [Online], December 2012. Accessed: March 2016, Available: `http://www.telekom.com/media/company/168008`.

[12] I. L. A. Division. PCI-SIG SR-IOV Primer - An Introduction to SR-IOV Technology. [Online], January 2011. Accessed: March 2016, Available: `http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html`.

[13] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 15–28. ACM, 2009.

[14] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards high performance virtual routers on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 20. ACM, 2008.

[15] P. Emmerich. MoonGen. [Online]. Accessed: March 2016, Available: `https://github.com/emmericp/MoonGen`.

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

[17] J. L. Garcia-Dorado, F. Mata, J. Ramos, P. M. S. del Rio, V. Moreno, and J. Aracil. High-Performance Network Traffic Processing Systems Using Commodity Hardware. pages 3–27. Springer Verlag, 2013.

[18] L. Gorrie. Kernel-bypass Networking. [Online], January 2013. Accessed: March 2016, Available: `http://lukego.github.io/blog/2013/01/04/kernel-bypass-networking/`.

[19] L. Gorrie. Snabb Switch's Kernel-bypass Networking. [Online], January 2013. Accessed: March 2016, Available: `http://lukego.github.io/blog/2013/01/05/kernel-bypass-networking-in-snabb-switch/`.

[20] L. Gorrie. [dpdk-dev] TX performance regression caused by the mbuf cachline split. [Online], May 2015. Accessed: March 2016, Available: `http://dpdk.org/ml/archives/dev/2015-May/017506.html`.

[21] L. Gorrie. Packet copies: Expensive or cheap?

[15] Developed since 2012

[Online], October 2015. Accessed: March 2016, Available: https://github.com/SnabbCo/snabb/issues/648.

[22] L. Gorrie. Snabb data structures: packets, links, and apps. [Online], September 2015. Accessed: March 2016, Available: https://github.com/lukego/blog/issues/11.

[23] L. Gorrie. Snabb Switch development: SIMD FTW, or, Straightline redux. [Online], March 2015. Accessed: March 2016, Available: https://groups.google.com/forum/#!msg/snabb-devel/aez4pEnd4ow/WrXi5N7nxfkJ.

[24] L. Gorrie. Snabb Switch in a Nutshell. [Online], August 2015. Accessed: March 2016, Available: https://github.com/lukego/blog/issues/10.

[25] L. Gorrie. Snabb Switch: kernel-bypass networking illustrated. [Online], October 2015. Accessed: March 2016, Available: https://github.com/lukego/blog/issues/13.

[26] L. Gorrie. Why Snabb? [Online], August 2015. Accessed: March 2016, Available: https://github.com/lukego/blog/issues/7.

[27] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE*, 53(2):90–97, 2015.

[28] U. Hoodbhoy. Harnessing the RAW Performance of x86 – Snabb Switch. [Online], October 2014. Accessed: March 2016, Available: http://umairhoodbhoy.net/2014/10/09/harnessing-the-raw-performance-of-x86-snabb-switch/.

[29] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. The evolution of lua. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 2–1. ACM, 2007.

[30] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *Communications Magazine, IEEE*, 51(11):24–31, 2013.

[31] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[32] S. Lowe. What is SR-IOV? [Online], December 2009. Accessed: March 2016, Available: http://blog.scottlowe.org/2009/12/02/what-is-sr-iov/.

[33] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 459–473. USENIX Association, 2014.

[34] T. Mastrangelo. Market Dynamics Forcing Transformation to All-IP Network. [Online], February 2013. Accessed: March 2016, Available: http://blog.advaoptical.com/market-dynamics-forcing-transformation-to-all-ip-network.

[35] C. Mateo. Performance of several languages. [Online], July 2015. Accessed: March 2016, Available: http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/.

[36] Microsoft. Single Root I/O Virtualization (SR-IOV). [Online]. Accessed: March 2016, Available: https://msdn.microsoft.com/en-us/library/windows/hardware/hh440235(v=vs.85).aspx.

[37] Nokia. Network architecture for the 5g era. [Online]. Accessed: March 2016, Available: http://networks.nokia.com/file/40481/network-architecture-for-the-5g-era.

[38] Ntop. PF_RING ZC (Zero Copy). [Online]. Accessed: March 2016, Available: http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/.

[39] M. Pall. The LuaJIT Project. [Online]. Accessed: March 2016, Available: http://luajit.org/.

[40] M. Paolino, N. Nikolaev, J. Fanguede, and D. Raho. Snabbswitch user space virtual switch benchmark and performance optimization for nfv. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 86–92. IEEE, 2015.

[41] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[42] B. Pinczel, D. Gehberger, Z. Turanyi, and B. Formanek. Towards high performance packet processing for 5g. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 67–73. IEEE, 2015.

[43] L. Rizzo. Netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.

[44] M. Rottenkolber. Continuous Integration for Snabb Switch. [Online], November 2015. Accessed: March 2016, Available: http://mr.gy/blog/snabb-ci.html.

[45] D. Scholz, D. Raumer, and F. Wohlfart. A Look at Intel's Dataplane Development Kit. *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, NET-2014-03-1 of Network Architectures and Services (NET), August 2014.

[46] snabb.com. Snabb NFV. [Online]. Accessed: March 2016, Available: http://snabb.co/nfv.html.

[47] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). RFC 5128 (Informational), Mar. 2008.

[48] C. S. Steele and J. Bonn. Fast functional simulation with a dynamic language. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–3. IEEE, 2012.

[49] J. Yang. Network Function Virtualization (NFV). [Online], June 2014. Accessed: March 2016, Available: https://jipanyang.wordpress.com/2014/06/16/network-function-virtualization-nfv/.

QUIC - Quick UDP Internet Connections

Florian Gratzer
Betreuer: Sebastian Gallenmüller, Quirin Scheitle
Seminar Innovative Internet-Technologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: florian.gratzer@tum.de

ABSTRACT

QUIC is a transport protocol on top of UDP, developed by Google. It uses mechanisms of TCP and introduces new features not used by other transport protocols. QUIC is optimized to be used for HTTP/2 connections and aims to reduce the end-to-end latency. QUIC works best (compared to TCP) for slow connections with high latency. This paper addresses the problems of TCP and the mechanisms of QUIC are discussed. QUIC packet and frame types are examined. The performance of HTTP over QUIC is compared to HTTP over TCP and HTTP + SPDY over TCP. The results show that QUIC is not better than the other protocols in all scenarios, but it can outperform TCP under certain network conditions.

Keywords

QUIC, Quick UDP Internet Connections, TCP, UDP, Layer 4, Google, Transport Protocol, HTTP/2, congestion control

1. INTRODUCTION

These days, the Internet is used for read the latest news or watching videos on platforms like YouTube. When the page load time is high, the user experience can become very bad.

In the last years, many approaches were considered to make Internet surfing faster. Internet Service Providers (ISPs) try to reduce page load times by increasing the bandwidth and using faster networking devices. Developers of web browsers tempt to make their software more efficient. Google's recent approach is to reduce the latency in the middle of the OSI model, namely in Layer 4, called Transport Layer.

Today, TCP and UDP are the most used protocols in the Transport Layer. While TCP is connection oriented, UDP is connectionless. Both protocols have advantages and disadvantages. TCP provides a reliable connection, but the TCP three-way handshake increases the latency for establishing a connection. This can be problematic for short living connections. In contrast, UDP establishes no connection, which results in a fast, but unreliable data transfer. To unite the advantages of both protocols, Google is developing a new transport protocol called **QUIC - Quick UDP Internet Connections** [8].

According to Google, QUIC performs better than TCP in scenarios with high Round Trip Time (RTT) and at least as good as TCP in most other scenarios [4]. This claim will be evaluated in Section 10. The most important differences between TCP and QUIC connections are that QUIC connections are always encrypted and connection establishment takes 0 RTTs when a server is known by a client and 1 RTT for the first connection to an unknown server. Figure 1 shows the connection establishment of QUIC compared to the TCP three-way handshake [10].

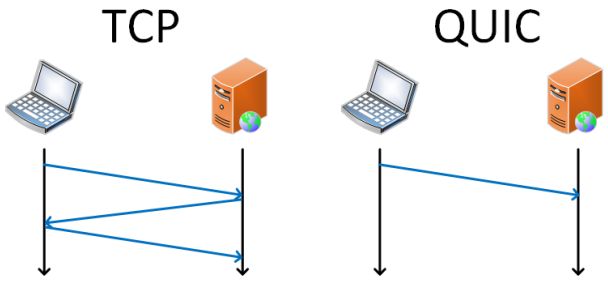


Figure 1: Connection Establishment of TCP and QUIC

TCP cannot be used to establish an encrypted connection within 0 RTT. Consequently, a different protocol is used as Transport Layer protocol. Therefore UDP is used, because it is already supported by most devices. Google's services like YouTube are already supporting QUIC , as well as Google's Browser Chrome [6]. In 2016, QUIC is still in development, so it is likely that the protocol changes in the near future. For this reason, Internet drafts of the IETF are used as sources instead of RFCs. These drafts can be updated or replaced by other documents at any time [9, 14].

This paper discusses the mechanisms of QUIC compared to TCP. Moreover, the QUIC packet and frame types are observed and the performance of QUIC is evaluated.

2. OVERVIEW OVER QUIC

QUIC is a transport protocol built upon UDP, designed to be used mainly for HTTP/2 [10], but it can be used by every application layer protocol. It is developed by Google and aims to reduce the connection establishment latency [10]. In contrast to TCP and UDP, QUIC connections are always encrypted and authenticated (see Section 5.7). The handshake is inspired by the handshake of TLS [5]. QUIC can be compared to TCP + TLS + HTTP/2. According to Google, QUIC has a better congestion control compared to TCP [10]. In Section 10, different scenarios are considered to evaluate this claim.

doi: 10.2313/NET-2016-09-1_06

3. PROBLEMS OF TCP

Today, TCP is widely used. It is more reliable than UDP but nevertheless it has several problems, especially when using it for HTTP/1.1. When designing a protocol based on TCP, these problems have to be addressed. To establish a TCP connection, the TCP three-way handshake is used. This handshake increases the connection establishment latency significantly. This is not problematic for long lasting connections, but can decrease the user experience for web surfing significantly. HTTP/1.1 uses a new TCP connection for each fetched URL [17].

Another problem is that a TCP segment can only carry a single HTTP/1.1 Request/Response. Consequently it is possible that a large number of small segments are sent within an HTTP/1.1 session. This can lead to a large overhead.

Also, HTTP/1.1 transfers are always initiated by the client [17]. This decreases the performance of HTTP/1.1 significantly when loading embedded files, because a server has to wait for a request from the client, even if the server knows that the client needs a specific resource [7].

The last problem discussed in this Section is Head-of-line (HOL) blocking. It occurs, when a packet is lost and consecutive packets arrive at the destination. The receiving host has then to wait for the retransmission of the lost packet until it can process consecutive packets, which arrived at the host without a loss. In scenarios like video streaming, a small number of lost packets do not have a notable influence on the user experience. Nevertheless, a TCP receiver has to wait for the lost packet before it is able to continue playing the video.

One way to overcome this problem with TCP is to open multiple connection between the same endpoints. This can work satisfyingly for a small number of connections, but when too many connections are opened, the connections tend to oscillate between very small and too large congestion windows when losses occur. This leads to a bad throughput and a bad user experience.

4. TCP MECHANISMS IN QUIC

Although QUIC uses UDP, many mechanisms are inspired by TCP. QUIC uses acknowledgments like TCP to inform the sender, that segments arrived at the receiver. The congestion control and loss recovery of QUIC is a reimplementation of TCP cubic with additional mechanisms [10]. TCP Cubic is optimized for high bandwidth networks with high latency [2]. In this section, the most important mechanisms of TCP used in QUIC are discussed. More can be found in the according Internet draft [14]. The mechanisms not used in TCP are focused in Section 5. QUIC uses a retransmission timer. Each segment, which is not acknowledged within this timer is considered to be lost (exceptions are discussed in Section 4.3).

QUIC also distinguishes between two phases: Slow Start and Congestion Avoidance. In the Slow Start phase, the congestion window grows exponentially, while it grows linearly in the Congestion Avoidance phase. A new connection always starts in the Slow Start phase until a loss occurs. After a loss, usually the congestion window the Fast Retransmit mechanism triggers and the connection changes to/stays in the Congestion Avoidance phase (more details in Section 4.1) [16].

4.1 Fast Retransmit

Fast Retransmit is a mechanism to avoid retransmission timeouts (RTOs). It triggers, when the sender receives three duplicate acknowledgments (ACKs) (this threshold is used in TCP as well as in QUIC) [4, 16]. A duplicate ACK is an acknowledgment for a segment, which has already been acknowledged and indicates a packet loss. When a RTO occurs, the congestion window is set to one maximum segment size (MSS), while the Fast Retransmit mechanism sets the congestion windows (and the Slow Start threshold) to a value dependent on the value it was before the loss. Also, the connection stays in the Congestion Avoidance phase and does not start with a new Slow Start as it is done after a RTO. According to evaluations by Google, over 99% of the packet losses are recognized by duplicate ACKs and so the Fast Retransmit mechanism kicks in in most cases [10].

4.2 Tail Loss Probe (TLP)

When a receiver does not receive the last segment of a transmission, Fast Retransmit cannot be triggered, because the receiver needs to receive (any) segments to identify the loss and therefore to send duplicate ACKs. To overcome this problem, QUIC uses Tail Loss Probes (TLPs). Before a RTO, the sender sends two TLPs containing the last unacknowledged segment [4]. The receiver then triggers the fast recovery mechanism [21].

4.3 Forward RTO-Recovery (F-RTO)

F-RTO aims to avoid unnecessary retransmissions to improve the performance of TCP/QUIC. TCP and QUIC use two mechanisms to trigger retransmission. As described in Section 4.1, Fast Retransmit kicks in, when the sender receives three duplicate ACKs. The second reason for retransmissions are RTOs. After RTOs, a sender sets the congestion window to one MSS and continues with a new Slow Start phase.

It is possible that a RTO occurs, even without a packet loss [18]. There are several reasons for spurious retransmissions, including delay spikes in mobile networks and different priorities of the sent data from the sender and the acknowledgments of the receiver [18]. After reducing the congestion window caused by delayed (but not lost) ACKs, the delayed ACKs reach the sender. As a result of the reduced congestion window, the ACKs are not inside the congestion window and trigger additional spurious retransmissions [18]. QUIC avoids this problem by not reducing the congestion window (and the Slow Start threshold) until the sender receives a subsequent ACK [4].

5. QUIC IMPROVEMENTS

As discussed in Section 4, QUIC's congestion control is inspired by TCP and therefore uses mechanisms of TCP. Additionally, QUIC uses mechanisms not used in TCP. Not all new mechanisms of QUIC are discussed in this paper and can be found in the Internet draft about QUIC's congestion control [4]. All mechanisms are pluggable, so QUIC can be configured to fit best in different scenarios.

5.1 Faster Connection establishment

When a QUIC client connects to a QUIC server for the first time (currently a server is identified by its IP address and UDP port [6]) it sends an empty, so-called inchoate, Client-HELO (CHLO) [10]. The server then responds with a rejection (REJ) including the server configuration and certificates [10]. The client uses this information to send another CHLO (can already contain application data) which is then accepted by the server (provided the versions of client and server are compatible) and all consecutively sent data are encrypted and authenticated [10]. When the server is known by the client, this inchoate CHLO is not needed, resulting in a 0 RTT handshake instead of the 1 RTT handshake for unknown server. More details about the handshake can be found in the QUIC Internet draft [9].

5.2 Multiplexing

TCP uses TCP ports and IP addresses (of both endpoints) to identify a connection, when Multipath TCP is not used. So it is not possible for a client to communicate with a server over multiple ports via a single connection. In contrast, QUIC uses a 64 bit connection identifier (which is randomly selected by the client) [5]. Within these connections, multiple streams are used to transport segments. This allows clients to establish connection mobility across IP addresses and UDP ports [14]. It is also possible to use multiple ports for an application, but the application has then to listen to all these UDP ports, because QUIC uses UDP as Layer 4 protocol. Connection IDs also allow connection migration. Thus, a connection can stay established, even when the IP address of one of the endpoints changes.

The support of multiple streams within a single connection also addresses Head-of -line blocking by sending independent data via different streams. All streams are identified by a stream identifier (stream ID) and can be established by the client or the server. To avoid collisions, the stream ID has to be even, when the client initiates the stream and odd when initiated by the server. Each participant has to increase the stream ID monotonically for new streams [14]. The stream IDs 0-3 are reserved. QUIC provides flow control on stream- and connection level.

5.3 Monotonically Increasing Sequence Numbers

TCP uses the same sequence numbers for retransmitted segments. This leads to the problem, that a host cannot distinguish between original and retransmitted segments. Contrary, QUIC uses a monotonically increasing sequence number for every segment, also for retransmitted ones, resulting in unique sequence numbers [14]. This helps to estimate the RTT more accurately, because the RTT can also be calculated for delayed ACKs. TCP can use an extension called Timestamps to distinguish between original and retransmitted segments too [22].

User data is transferred within stream frames (see Section 7). This frames also contain sequence numbers, which stay the same for retransmitted data [6].

5.4 Better Signaling

QUIC has a more verbose signaling than TCP. As focused in Section 5.3, all packets get a new sequence number, even retransmitted ones. Therefore every sequence number is unique, and a receiver can distinguish between an original packet and its retransmitted equivalent. As a result, a sender can calculate the RTT more accurately, because it can recognize the difference between a delayed ACK and the ACK of the retransmitted packet.

TCP uses cumulative ACKs. This means that all segments with a sequence number smaller than an acknowledged packet are also acknowledged. To reduce the amount of ACKs, only the packet with the highest sequence number in the receive window is acknowledged. As a result, the sender has to wait a whole RTT to notice the loss or to unnecessarily retransmit received packets [15].

To overcome this problem, TCP has the option to use Selective Acknowledgments (SACKs). SACKs are used by the receiver to inform the sender, which packets are received, so a sender can retransmit lost segments. More details can be found in RFC 2018 [15].

QUIC uses Negative Acknowledgments (NACKs) instead of SACKs with a bigger range (up to 255 instead of 3) [9]. Negative Acknowledgments report lost packets directly instead of implicating it by not getting acknowledged. Although NACKs and SACKs are different approaches to report lost packets, the result is comparable. The larger range of QUIC NACKs compared to TCP SACKs is advantageous for large receive windows.

5.5 Forward Error Correction (FEC)

QUIC can use Forward Error Correction (FEC) to reconstruct lost packets [14]. A scheme similar to RAID systems using XOR operations is used for this purpose. This approach for reconstructing packets is simple and therefore effective, but it cannot reconstruct lost packets when multiple packets are lost within a group. As it will be discussed in Section 10 this feature can have a negative effect when the loss rate is low or too high [13].

5.6 Packet Pacing

TCP tries to send data as fast as possible. When data is sent too fast, losses are very likely. When a loss occurs, and Fast Retransmit kicks in, the congestion window is decreased. Then the congestion windows grows again until the next loss occurs. This can result in a very bursty transmission [23].

Packet Pacing is an approach to make the transmission less bursty by not sending at full rate. This has usually a positive effect in scenarios with low bandwidth, but it can decrease the overall throughput for fast connections (see Section 10). Packet Pacing is also used for TCP by some Linux kernels.

5.7 Authentication and Encryption

TCP Headers (and the payload, as long as no other protocols are used) are neither encrypted nor authenticated. Conversely, QUIC packets are always encrypted (except for the public header, see Section 6) and authenticated (including the public header) after the connection is established.

This also includes IP spoofing protection. The handshake is inspired by TLS. More details can be found in the QUIC crypto documentation [11].

6. PACKET TYPES AND HEADER FORMAT

QUIC differentiates between two types of packets: Regular Packets and Special Packets. Special Packets can either be Version Negotiation Packets or Public Reset Packets. Regular Packets are divided into Frame Packets and FEC Packets (see Section 5.5). QUIC packets have two different headers: an unencrypted Public Header and an encrypted Private Header. More information about these headers can be found in the according Internet draft [9].

6.1 Public Header

Figure 2 shows the Public Header of Regular Packets. Special Packets also have a Public Header with a slightly different design. The flag field, which is the same for all packet types is shown more in detail in Figure 3:

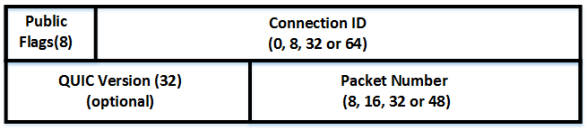


Figure 2: QUIC Public Header (Numbers in bits) [9]

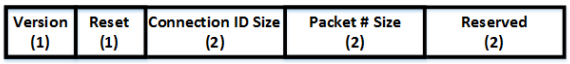


Figure 3: Flag Field in QUIC's Public Header (Numbers in bits) [9]

Public Flags: The *Version* flag is interpreted differently, depending on the sender of the packet (client or server). When set by the client, the packet includes the *QUIC Version* field. A client has to set this flag until it receives a packet from this server without setting this flag. When *Version* is set by the server, it indicates that the packet is a *Version Negotiation Packet*.

The *Reset* flag is set for Public Reset Packets. When both, *Version* and *Reset* are set, the packet is treated as *Public Reset* Packet.

The *Connection ID Size* and *Packet Number Size* determine the length of the according header field. The exact interpretation can be found in the according Internet draft.

The last two bits are unused and reserved for future use. More details can be found in the QUIC Internet draft.

Connection ID: A randomly generated identifier, used to identify the connection instead of the four-tuple (source IP address, source port, destination IP address, destination port) used by TCP.

QUIC Version: This field is only present, when the *Version* flag is set and is used by the client to propose a QUIC version to be used when establishing a connection.

Packet Number: Used to identify packets. As described in Section 5.3, each packets gets a unique identifier, even retransmitted ones.

6.2 Regular Packets

Regular Packets are always encrypted (except for the public header) and authenticated (including the public header). All regular packets have a common Private Header format. This header starts directly after the Public Header and is followed by the payload, which has a different format for Frame Packets and FEC Packets. The Private Header is shown in Figure 4:

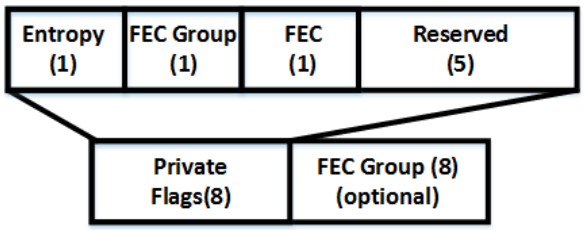


Figure 4: Private Header Format (Numbers in bits) [9]

Private Flags: The *Entropy* flag indicates that this packet contains the 1 bit of entropy in frame packets or the result of the XOR-Operation of the entropy flag of the other packets in the FEC group when it is an FEC Packet. The *FEC Group* flag determines whether FEC is used and therefore if the FEC field is present. The last flag (*FEC*) is set for FEC packets.

FEC: This field is used to determine, which packets belong to the same FEC group. The value in this field is the offset from the first packet in the FEC group to this packet.

6.2.1 Frame Packet

Frame Packets carry the actual application data within a connection. The payload is located directly after the Private Header and formatted as shown in Figure 5:

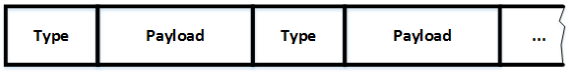


Figure 5: Frame Packet Format [9]

Type: Specifies the Type of the Frame. The Frame types are focused in Section 7.

Payload: Payload of the frame, is dependent on the type of the frame.

6.2.2 FEC Packet

Regular Packets with the *FEC* flag set are FEC Packets and contain the result of the XOR-Operation of the Frame Packets of the same FEC group.

7. FRAME TYPES

QUIC differentiates between Regular Frame Types and Special Frame Types. The frame types are not discussed in detail in this paper, a detailed list of all frame types and the values of the field type for the according frame type can be found in the QUIC Internet draft. Not all frame types are used in the current implementation (see description of according frames) [9].

7.1 Regular Frame Types

Regular Frames can either be Stream, Acknowledge or Congestion Feedback Frames.

Stream Frames: Used to send data over a stream and to (implicitly) create a stream.

Acknowledge Frames: Used to signal that a packet (not necessarily a Frame Packet) has been received or is missing. The exact format can be found in the QUIC Internet draft.

Congestion Feedback Frame: Currently not used. Designed to provide additional congestion feedback in future implementations.

7.2 Special Frame Types

In Addition to the Regular Frame Types, the following Special Frame Types are specified.

Padding Frame: Used to pad a packet with binary zeros.

Connection Close Frame: Used to notify the communication partner that a connection is closing. All unclosed streams within a connection are also closed, when a connection is closed.

Reset Stream Frame: Used to irregularly terminate a stream. When sent by the stream creator, the creator indicates that he wants to close a stream. When the receiver of a stream sends a Reset Stream Packet, he either does not accept the stream or an error occurred.

Go Away Frame: Used to inform the opposing endpoint that the connection will be closed in the near future and should not be used any more. Usually sent shortly before a Connection Close Frame.

Windows Update Frame: Used to tell the opposing peer to update the receive window.

Blocked Frame: Used for debugging purposes. Inform the receiver of a frame that the sender is ready to transmit data, but is blocked by flow control mechanisms.

Stop Waiting Frame: Used to tell the communication partner not to wait for packets any more.

Ping Frame: Used to verify, that the opposing endpoint is still available. A ping frame contains no payload and the receiver has to answer with an Acknowledgment Frame.

8. QUIC DISCOVERY IN CHROME

Chrome uses TCP to send requests to unknown server. When a server supports QUIC, it uses the Alternate Protocol Header in HTTP Responses to inform a client, that it supports QUIC [12]. For the next requests, Chrome tests which protocol is faster and uses the faster protocol. It also stores a list of all server, which support QUIC for subsequent connection [12]. When a QUIC connection fails (e.g. incompatible versions of client/server, Firewalls, ...) the host is marked as broken for a certain time span (currently five minutes), where QUIC will not be tested again. After this time span, QUIC is tested again and is marked as broken for twice as long as before, when QUIC is still unable to establish a connection [12]. When a user does not want to use QUIC (for any reason), it is possible to disable it manually.

9. SPDY

The performance of QUIC is usually evaluated by comparing the page load time of HTTP/1.1 over TCP, HTTP/1.1 + SPDY over TCP and HTTP/1.1 over QUIC. SPDY is discussed shortly in this section.

SPDY ("speedy") was developed by Google and enhances the HTTP/1.1 Protocol. In Contrast to pure HTTP/1.1, SPDY allows hosts to send multiple HTTP/1.1 Requests/Responses within a single TCP segment. With SPDY it is also possible to prioritize HTTP/1.1 Requests/Responses. So SPDY solves one of the problems, discussed in Section 3, but the other problems can't be solved by SPDY, because it is still using TCP.

HTTP/2 is based on SPDY and today it is used more often than SPDY. As a consequence, the use of SPDY is not longer supported by Google [3]. As SPDY is not in the focus of this paper and just used for comparison, it is not discussed here further. More informations can be found in the SPDY Internet draft [20] and on the SPDY Chromium project page [19].

10. EVALUATION

QUIC was evaluated by Google and independent testers. This Section summarizes their results. The experiments of Google set the focus on the effect of different mechanisms. On the one hand, they are made from the developing company, so they have to be treated with care. On the other hand, Google can interpret the results best, as they know the scenario, which QUIC is intended for better than independent testers. Moreover they show the impact of different mechanisms more in detail than other testers. When HTTP, QUIC and SPDY are compared, HTTP refers to HTTP/1.1 over TCP, QUIC refers to HTTP/1.1 over QUIC and SPDY is short for HTTP/1.1 + SPDY over TCP.

10.1 Experiments by Google

According to Google's experiments in 2015, 75% of the connections are connections to known hosts and therefore it takes *0 RTT* to establish this connections. Google assumes that this is the reason for 50 - 80% of the overall median latency improvements when comparing TCP and QUIC [4].

Packet Pacing reduces retransmission by 25%. To do this, the sender does not send at the maxiumum speed. As a

result the page load latency is reduced for slow connection, but fast connections suffer from this mechanism. All in all it does not change the median page load latency [4].

TLP has no effect on the median latency, but it improves the 95% - quantile of the latency and the YouTube rebuffer rate by almost 1% [4].

All in all, Google comes to the conclusion, that QUIC performs significantly better for slow connections with high latency and performs as good as TCP for fast connections with a low latency [6].

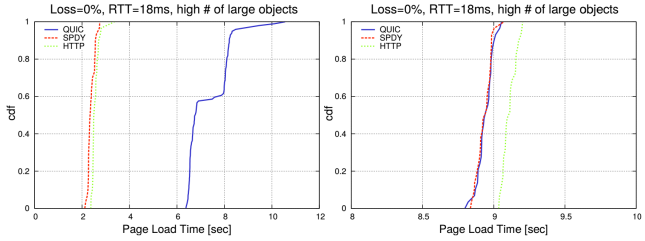
10.2 Measurements from the Budapest University of Technology and Economics

A group from the Department of Telecommunications and Media Informatics of the Budapest University of Technology and Economics tested QUIC compared to HTTP and SPDY [7]. Their measurement environment and results are shown here. The group did a higher number of test, only the most interesting ones are discussed in this paper.

Measurement Environment: A regular laptop with Google Chrome and additional tools for measurement purposes were used on the client side. On the server side, they used four sample pages, hosted on Google Sites. In their scenario, the demo pages contained either small (400 B - 8 kB) or large (128 kB) objects. For each scenario, there was a page with a small (5) and a page with a large (50) amount of the according objects. Between the client and the server, a shaper server was used in order to change the network conditions [7]. In contrast to the experiments of Google, they do not test the influence of enabling/disabling different mechanisms.

The following Figures show the Cumulative Distribution Function (CDF) (y-axis) of the Page Load Times (PLT) (x-axis) under different conditions [7].

In one of their tests, they showed that QUIC performs significantly worse in a scenario with, 50 Mb/s bandwidth, low RTT (18 ms), a packet loss rate of 0% and 50 objects with a size of 128 kB. When TCP is used (with and without SPDY) for HTTP connections, the average page load time is about 2 seconds, while it it higher than 7 seconds, if QUIC is used. After reducing the bandwidth to 10 Mb/s, the performance of QUIC is again comparable to the other ones and the average page load time is about 9 seconds. The result is shown in Figure 6.

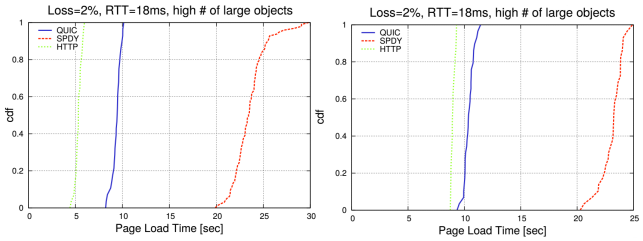


a) Bandwidth = 50Mbps b) Bandwidth = 10Mbps

Figure 6: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

The reason for the weak performance of QUIC is packet pacing [7]. This mechanisms tries to reduce retransmissions by sending below the maximum rate possible. In their scenario, the loss rate is 0%, so packet pacing tries to avoid packet losses, which wouldn't be there even when the host sends at full speed. The researchers argue, that QUIC underestimates the maximum bandwidth and sends at a far too slow rate [7]. When using a lower bandwidth of 10 Mb/s this behavior cannot be observed.

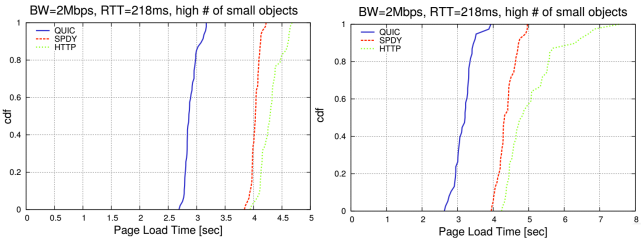
A completely different picture is shown, when the loss rate in the former scenario is changed. Figure 7 shows the effect of setting the loss rate to 2% in an apart from that unchanged scanario [7]. HTTP still performs better than QUIC, but the difference got smaller. In contrast, SPDY performs very bad. The reason for this is Head-of-line blocking, when sending multiple HTTP responses via a single TCP segment (see Section 3).



a) Bandwidth = 50Mbps b) Bandwidth = 10Mbps

Figure 7: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

Google claims that QUIC performs significantly better than other protocols when QUIC is used for slow connections with high RTTs, especially when requesting a large number of small objects. The researchers used a scenario with 2 Mb/s bandwidth, a high RTT of 218ms and 50 objects between 400B and 8kB to verify this claim [7]. The results shown in Figure 8 support Google's claim. QUIC performs better in both scenarios (0% and 2% loss rate). The benefits are mostly gained by the multiplexing mechanism of QUIC, which reduce the overhead significantly when sending a high number of small objects [7]. The group also argue that the 0 RTT connection establishment has a positive effect on page load time, which has to be mentioned [7].



a) Loss = 0% b) Loss = 2%

Figure 8: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

The group comes to the conclusion, that QUIC performs badly under high bandwidth, but outperforms the other protocols when the RTT is high, especially under low bandwidth

doi: 10.2313/NET-2016-09-1_06

[7].

10.3 Measurements of the Politecnico di Bari

A group from Gaetano Carlucci Politecnico di Bari & Qualvlive in Italy also performed measurements to evaluate QUIC [1]. Their test environment was similar to the environment of the Hungarian group, but they used different scenarios. Details can be found in their paper [1].

In contrast to the testers of the Budapest University of Technology and Economics, the Italian group tested the influence of FEC on the loss rate and the channel utilization. They used scenarios with 3, 6 and 10 Mb/s and a loss rate of 0 and 2% The results are shown in Figure 9.It can be seen that the loss rate is higher when FEC is enabled.

These results may come as a surprise at first glance. FEC cannot reconstruction lost packets, when more than one packet of a group is lost. When using FEC, the sender of the packet is not informed about the loss after the receiver was able to reconstruct the packet and so it continues to increase the congestion window until a packet loss occurs, where FEC cannot reconstruct the packet. This results in the observed behavior. Also, the channel utilization is higher, when using FEC because of the required redundancy, which also has to be sent via the same channel.

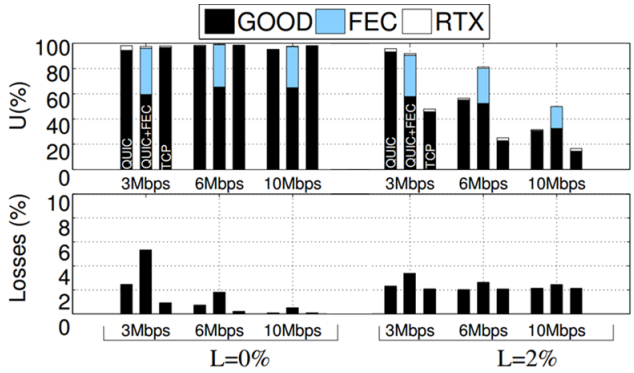


Figure 9: QUIC with enabled/disabled FEC compared to TCP [1]

The group also tested a similar scenario with low bandwidth and a high round trip time to verify Google's claims, that QUIC performs significantly better than TCP under this circumstances. This study confirms the results of Section 10.2. They used a bandwidth of 3 Mb/s, a (high) loss rate of 2% and an RTT of 50ms [1]. The result is shown in Figure 10. The y-axis show the throughput in kb/s and the x-axis shows the time in seconds.

The last interesting part of the evaluation, discussed in this paper, is the performance of QUIC when it is used in parallel with TCP with limited buffers. The research group used a bandwidth of 5 Mb/s and a RTT of 50ms for this scenarios. Then the used buffers of 13, 30 and 60 kB as bottleneck (with Tail Drop policy) [1]. Figure 11 show the throughput of the two protocols when using different buffer sizes. When QUIC and TCP are used in parallel, QUIC uses more of the buffer size unless the network is over-buffered (60kB under their conditions). This is because QUIC uses a smaller congestion window reduction factor than TCP [1].

Figure 12 shows that QUIC does not only use more of the buffer, but also has a higher throughput. When the buffer is large enough, both protocols have a comparable throughput.

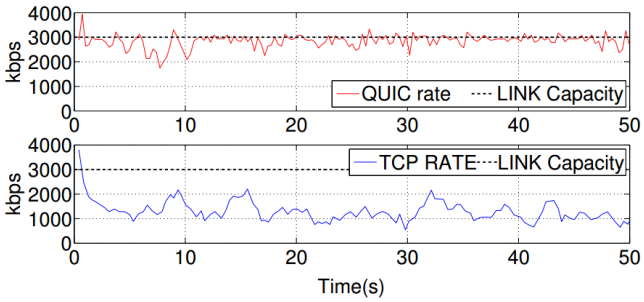


Figure 10: Comparison of TCP and QUIC (without FEC) [1]

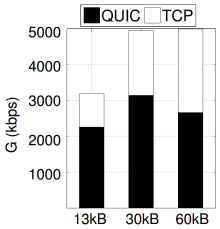


Figure 11: Impact of buffer size on QUIC/TCP [1]

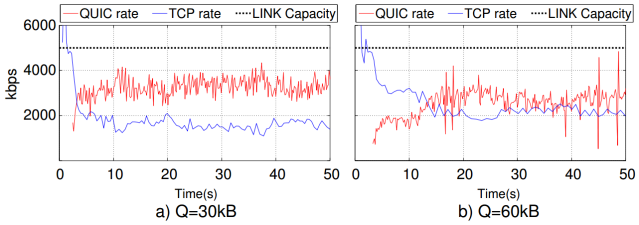


Figure 12: Parallel use of TCP and QUIC with limited buffers [1]

11. CONCLUSION

QUIC is a transport protocol, developed by Google, meant to be used by HTTP/2 [10]. It is still under development, so not all mechanisms discussed in this paper are implemented yet [9, 14]. QUIC uses UDP as Layer 4 protocol, but several mechanisms used are reimplementations of TCP best practices. QUIC also has features, which are not possible for TCP [4]. In contrast to TCP, encrypted QUIC connections can be established within 0 RTTs.

Google claims that QUIC works at least as good as TCP in all scenarios [6]. Measurements showed, that this is not entirely true. QUIC performs better than TCP in networks with high latency, especially when the bandwidth is low. The gains come from the 0 RTT handshake and QUIC's multiplexing mechanism. However, TCP performs better, when the bandwidth is high and the latency is low [1, 7].

doi: 10.2313/NET-2016-09-1_06

Also, QUIC introduces Forward Error Correction, which is used to reconstruct lost packets instead of requesting it again. Therefore, redundant data has to be sent. The current implementation does not work as well as Google intends, because it increases the amount of retransmissions (see Section 10.3). The problem with the FEC mechanism of QUIC is that the sender is not informed about a loss, when the packet was successfully reconstructed. Also it cannot reconstruct packets, when multiple packets of a FEC Group are lost. All in all the performance of QUIC is worse, when FEC is enabled [1].

In 2016, QUIC is used by Google services like YouTube and Google's Browser Chrome, but QUIC is still under development [1, 7]. When Google wants to establish QUIC on more server in the future, QUIC has to become better and the unimplemented mechanisms must be implemented. Some mechanisms of QUIC are innovative, but the actual implementation is not sufficient to replace TCP. QUIC has a lot of potential,but it has to be observed if QUIC can coexist with TCP in the future or if it can even replace the currently most used transport protocol.

12. REFERENCES

[1] G. Carlucci, L. De Cicco, and S. Mascolo. HTTP over UDP: An Experimental Investigation of QUIC. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 609–614. ACM, 2015. http://c3lab.poliba.it/images/3/3b/QUIC_SAC15.pdf.

[2] CUBIC for Fast Long-Distance Networks. https://tools.ietf.org/html/draft-rhee-tcpm-cubic-02.

[3] Hello HTTP/2, Goodbye SPDY. https://blog.chromium.org/2015/02/hello-http2-goodbye-spdy.html.

[4] IETF93 QUIC BarBoF: Congestion Control and Loss Recovery. https://docs.google.com/presentation/d/1T9GtMz1CvPpZtmF8g-W7j9XHZBOCp9cu1fW0sMsmpoo/edit#slide=id.gb7abf88bb_0_28.

[5] IETF93 QUIC BarBoF: Protocol Overview. https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs/edit#slide=id.g99041b54d_0_124.

[6] IETF93 QUIC video (BAR BOF). http://recordings.conf.meetecho.com/Playout/watch.jsp?recording=IETF93_QUIC&chapter=BAR_BOF.

[7] P. Megyesi, Z. Krämer, and S. Molnár. Comparison of web transfer protocols. http://proprogressio.hu/wp-content/uploads/2016/01/MolnarSandor_2015.pdf.

[8] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[9] QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 IETF Internet Draft. https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02.

[10] QUIC at 10,000 feet. https://docs.google.com/document/d/1gY9-YNDNAB1eip-RTPbqphgySwSNSDHLq9D5Bty4FSU/edit.

[11] QUIC crypto design doc. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit.

[12] QUIC Discovery. https://docs.google.com/document/d/1i4m7DbrWGgXafHxwl8SwIusY2ELUe8WX258xt2LFxPM/edit.

[13] QUIC FEC v1. https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit.

[14] QUIC Loss Recovery And Congestion Control IETF Internet Draft. https://tools.ietf.org/html/draft-tsvwg-quic-loss-recovery-01.

[15] RFC2018 - TCP Selective Acknowledgment Options. https://tools.ietf.org/html/rfc2018.

[16] RFC2581 - TCP Congestion Control. https://tools.ietf.org/html/rfc2581.

[17] RFC2616 - Hypertext Transfer Protocol – HTTP/1.1. https://www.ietf.org/rfc/rfc2616.txt.

[18] RFC5682 - Forward RTO-Recovery (F-RTO): An Algorithm for Detecting. https://tools.ietf.org/html/rfc5682.

[19] SPDY Chromium Project Page. http://dev.chromium.org/spdy/.

[20] SPDY IETF Internet Draft. https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00.

[21] Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses. https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01.

[22] TCP Extensions for High Performance. https://www.ietf.org/rfc/rfc1323.txt.

[23] TCP Performance Implications of Network Path Asymmetry. https://tools.ietf.org/html/rfc3449.

Using Looking Glasses to Understand the Internet's Structure

Jagjit Singh
Betreuer: Quirin Scheitle
Seminar Innovative Internettechnologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: jagjit.singh@tum.de

KURZFASSUNG

Looking Glasses (LGs) spielen eine sehr wichtige Rolle, um die Struktur des Internets besser darstellen zu können. Sie bieten sowohl Kontrollschicht- als auch Datenschichtmessungen von verschiedenen Blickwinkeln aus dem Internet. Da es jedoch keinen Standard für die Ein- und Ausgabe der Daten gibt, wird die Zusammenarbeit verschiedener LG Server erschwert, die wichtig ist, um das Potenzial auszunutzen. Es gibt mehrere Frameworks, die die Zusammenarbeit verschiedener LGs erlauben. Eines dieser Frameworks, Periscope, wird in diesem Paper genauer untersucht.

Schlüsselworte

Looking Glass (LG) Server, Border Gateway Protocol (BGP), Inter-Domain Routing, Autonomous Systems (ASes)

1. EINLEITUNG

Messungs- und Monitoring-Tools sind sehr wichtig und notwendig, denn sie bieten Lösungen für mehrere Problemdomänen: Von wissenschaftlicher Forschung bis hin zur Ermittlung von Topologien, Anomalien und Sicherheitsmängeln in den Netzwerken.

Viele Internetdienstleister verwenden eigene Diagnose- und Monitorings-Tools, unter anderem auch Looking Glass Server, um Störungen im eigenen Netz schnell finden zu können und die Leistung des Netzes zu verbessern. LG Server bieten eine Reihe von Befehlen wie *Ping, Traceroute* und *Border Gateway Protocol (BGP)*-Befehle als Webschnittstelle an. Diese Befehle können verwendet werden, um Informationen über Übertragungswege zu erhalten, die sehr hilfreich sind, um z.B die Ursache einer Störung, wie falsch konfigurierte BGP Route Advertisement, ausfindig zu machen [12]. Traceroute- und BGP-Anfragen liefern Informationen über die logischen Verbindungen der Border-Router und wie die Datenpakete tatsächlich weitergeleitet werden. Das Problem allerdings ist, dass verschiedene LG Server unterschiedlich implementiert sind. Jeder LG Server wird unabhängig von anderen LG Servern betrieben und es gibt keine zentrale Datenbank, die alle LG Server indiziert. Es gibt auch keinen Standard für das Format der Anfragen und Ausgabe der angefragten Daten. LG Server sind für die manuelle Nutzung gedacht. Die Anzahl der Anfragen ist auch limitiert, um DoS-Attacken zu vermeiden.

Ein Schwerpunkt dieses Papers ist das Framework Periscope. Dieses Framework vereinigt die unterschiedlichen, voneinander unabhängig betriebenen LG Server und bietet eine standardisierte öffentlich verfügbare API, die on-demand Messungen erlaubt [7].

Der weitere Aufbau dieser Ausarbeitung ist wie folgt: Abschnitt zwei beschreibt kurz eine abstrakte Sicht für den Aufbau des Internets, im Abschnitt drei werden verschiedene Möglichkeiten der Ermittlung von AS-Topologien beschrieben und Studien vorgestellt, die auch mit LGs arbeiten. Im Abschnitt vier werden die Periscope Architektur und dessen Komponenten beschrieben. Abschnitt fünf fasst die behandelten Themen noch einmal zusammen und gibt weitere Verweise zur Nutzung von Periscope.

2. GRUNDLAGEN

Das Internet besteht aus Autonomen Systemen (eng. Autonomous Systems ASes), die Inter-Domain Routing Informationen mit Hilfe des Border Gateway Protocols (BGP) untereinander austauschen [23]. Das Internet kann als ein Topologie-Graph auf AS-Ebene betrachtet werden, wobei jedes Autonome System einen Knoten darstellt und eine logische Verbindung zwischen zwei Autonomen Systemen eine Kante. Diese logische Verbindung zwischen zwei ASes wird auch Link genannt. LG Server können sowohl als Webschnittstelle, als auch via Telnet zur Verfügung gestellt werden und sind öffentlich zugänglich. Einige LG Server erlauben das Auslesen von Routingtabellen.

3. MESSMÖGLICHKEITEN

Es gibt unterschiedliche Möglichkeiten Topologien auf AS-Ebene zu erstellen. Dafür können Datenquellen verwendet werden, die unterschiedliche Methoden verwenden, um die Topologien zu erstellen: BGP basiert (IRL [24], RouteView [20], RIPE-RIS [19]), Traceroute-basiert (Ark [3], iPlane [13]) und Internet Routing Registry (IRR) [9]. Jedoch können die unterschiedlichen Datenbanken veraltet sein. Ein Paper von Khan *et al.*[12] beschreibt eine AS-Topologie mit Hilfe der LG Server. Dafür kommen BGP-Befehle wie *show ip bgp summary* zum Einsatz. Diese liefern die Routingtabellen von ASes vom jeweiligen Server, in dem sich der LG Server befindet. Eine typische Ausgabe zum Beispiel von einem Cisco Router sieht wie folgt aus:

BGP router identifier 10.0.0.1, local AS number 123
...

Neighbor	V	AS	others
192.168.0.1	4	123	...
192.168.1.1	4	456	...

doi: 10.2313/NET-2016-09-1_07

Das Ergebnis liefert die Information, dass der angefragte Router in AS123 liegt und zwei Nachbarrouter besitzt, wobei einer im gleichen AS ist und der andere sich in AS456 befindet. Die AS-Topologie enthält dann zwei Knoten (123, 456) und eine Kante (123-456).

3.1 Verschiedene AS-Topologien

In diesem Abschnitt werden einige AS Topologien kurz beschrieben, die in den Studien von Khan *et al.* [12], Augustin *et al.* [1] und Zhang *et al.* [26] verwendet wurden.

BGP Trace Collectors: BGP Trace Collectors [26] sind mit kommerziellen ISP Netzwerken über BGP Sitzungen verbunden. Ein Collector empfängt BGP Nachrichten von anderen Routern, die damit verbunden sind, schickt jedoch keine Prefix-Advertisements zu anderen Routern. So sammelt ein Collector die Routingtabellen und Updates, die er von anderen Routern empfangen hat.
Routingtabellen enthalten Informationen über bevorzugte Pfade, um ein Zielpräfix von IP Adressen zu erreichen. Die Routingupdates liefern Informationen über alternative Pfade. RouteViews [20] und RIPE RIS [19] sind die Hauptprojekte, die BGP Trace Collectors aufstellen und die gesammelten Daten veröffentlichen.
Route Servers: Route Servers sind Router, die von einigen Internetdienstanbietern öffentlich gemacht werden. Benutzer können sich über Telnet mit diesen Routern verbinden und verschiedene Router-Befehle wie *show ip bgp* ausführen, um die ganze Routingtabelle zu sehen. Allerdings speichern die Route Server keine Routing-Updates, wie dies bei BGP-Collectors der Fall ist.
IXP Datenbanken: Packet Clearing House [15] und PeeringDB[16] sind zwei Hauptquellen, die Daten über IXPs liefern. Sie enthalten Informationen wie IXP Namen, deren Standorte, IXP Präfixe, Liste der Mitglieder und Hyperlinks zu IXP Webseiten. Beide Datenbanken basieren auf freiwilligen Beiträgen. Packet Clearing House löscht nie IXPs, die einmal in der Datenbank sind. Sie werden als "stillgelegt" markiert, sobald die nötigen Beweise dafür verfügbar sind.
IRL: UCLA IRL [24] veröffentlicht regelmäßig die AS Topologie, welche von unterschiedlichen Datenquellen wie RouteViews[20], RIPE-RIS [19], Packet Clearing House (PCH) [15] und Internet2 [10] extrahiert wird [12].
Ark: CAIDA Archipelago (Ark) [3] bietet eine AS Topologie an, die auf Traceroute-Messungen basiert. Im März 2013 waren 71 Ark-Monitors über das Internet verteilt. Es wurden 116K direkte AS-Verbindungen von dem IPv4 Route /24 AS Datensatz verwendet [12].
iPlane: Madhyasta *et al.* [13] haben iPlane Dienst vorgeschlagen, welches Traceroute-Befehle von über 300 PlanetLab Standorten schickt, um die Internet Topologie zu mappen. Es wurden 81 K AS-Links von iPlane AS Topologie im März 2013 gefunden [12].
Internet Routing Registry (IRR): IRR [9] ist eine Datenbank, in der Routing- und Adresseninformationen registriert sein können. Selbst wenn einige Informationen nicht mehr aktuell sind [11], enthält die Datenbank Informationen über AS-Links, die nicht in BGP und Traceroute-basierten Topologien enthalten sind[8] [12].

3.2 AS Topologie Ermittlung mithilfe von LG Servern

Dieser Abschnitt beschreibt, wie Khan *et al.*[12] eine AS-Topologie mit Hilfe der Daten von verschiedene LG Servern erstellen.
Ein automatisiertes Tool verarbeitet Anfragen an 388 verschiedene LG Servern. Das Tool schickt gleichzeitig 30 Anfragen an die LG Server und legt 15 Sekunden Pause ein, falls mehrere aufeinanderfolgende Anfragen an den gleichen LG Server gerichtet sind. Es sind fünf Schritte notwendig, um die Daten von den LG Servern zu sammeln:

- Das Tool parst die Website vom LG Server, um die unterstützten LG Befehle und LG Router zu finden, damit später Abfragen an sie geschickt werden können.
- An jeden LG Router wird ein *show ip bgp summary* Befehl über den LG Server gesendet.
- Aus dem Ergebnis extrahiert das Tool die IP-Adressen von den Nachbarroutern des LG Servers.
- An die gefundenen IP-Adressen der Nachbarrouter wird ein *BGP neighbor ip advertised routes*-Befehl gesendet, um die vorgeschlagenen Routen von dem ursprünglichen LG Server an die Nachbarn zu finden.
- Zum Schluss werden die Ergebnisse der beiden Befehle *show ip bgp summary* und *BGP neighbor ip advertised routes* in einer Textdatei gespeichert, um daraus die AS-Topologie konstruieren zu können.

245 von 388 LG Servern unterstützen den *show ip bgp summary*-Befehl von 1.900 Standorten, die über 110 Länder verteilt sind. Mit den 245 LG Servern wurden im März 2013 zweimal in der Woche Messungen durchgeführt. Das Ergebnis der insgesamt 8 Schnappschüsse stellt kombiniert einen AS-Link Datensatz dar. Die Intra-AS Links, deren Ziel- und Quelladressen sich im selben AS befinden, werden dabei ausgefiltert und es bleiben 16.000 Inter-AS Links übrig.
Von den 245 LG Servern unterstützen 59 LG Server den *BGP neighbor ip advertised routes*-Befehl, welcher einmal wöchentlich im März 2013 ausgeführt wurde. Deren LG Router sind an 250 Standorten in über 40 Ländern verteilt und schicken an 5.000 Routern in den Nachbar-ASes die Route-Advertisements. Die Studie hat 686 Autonome Systeme und 11.000 AS-Links entdeckt, die nicht in anderen Datenbanken, wie IRL, IRR, Ark und iPlane vorkommen.

3.3 IXPs mithilfe von AS-Topologie mappen

„IXPs: Mapped?" ist eine Studie von Augustin *et al.* [1] die einen Ansatz vorstellt, um IXPs zu mappen, damit mehr Informationen über Peering-Matrizen (z.B. wer verbindet sich mit wem über welchen IXP) und den dadurch entstehenden Verkehr im Netz (z. B wie viele Daten tauschen die verschieden ASes über einen IXP aus) verfügbar sind.
Mehrere Datenquellen spielen eine entscheidende Rolle. Als erste Datenquelle kommt eine Liste von 278 IXPs zum Einsatz, die insgesamt 393 Präfixe, deren geografischen Standorte und Mitglieder enthält. Eine entscheidende Rolle spielt eine weitere Datenquelle, die 2300 LG Server auflistet. Diese erlauben das Ausführen von Traceroute-Befehlen und sind in

66 Ländern verteilt. Auch die AS Nummern (ASN) der zugehörigen Netzwerken sind bekannt. Diese Liste wurde mithilfe von *traceroute.org* Datenbank [22] erstellt und 486 weitere LGs wurden mithilfe von PeeringDB[16] hinzugefügt und 20 LGs wurde mithilfe von Suchmaschinen gefunden. Die dritte wichtige Komponente ist die gefolgerte AS-Karte, die beim Zusammenfügen von BGP Routingtabellen von RIPE RIS und RouteViews Projekte erstellt wurde und es wurde die aktuellste Version des Cyclops Projekts [6] heruntergeladen. Diese AS-Karte wird um von der Studie neu entdeckte Peerings ergänzt. Eine typische Cyclops-basierte AS-Karte enthält ungefähr 110.000 AS Links. Die erweiterte Karte von Augustin *et al.* [1] enthält ungefähr 10-20% mehr AS Links. Es wurden gezielte Traceroute-Experimente im Juli 2008, Dezember 2008 und April 2009 durchgeführt. Zusätzlich dazu wurden von der PlanetLab [17] Datenbank 254 aktive Knoten gewählt, um die Ergebnisse verschiedener Plattformen zu vergleichen. Anhand dieser Listen wurde von jedem aktiven Knoten und für jedes AS, ein Traceroute-Befehl zu einer IP Adresse durchgeführt, die auf den Ping-Befehl reagierte. Die Abbildung 2 listet für jede Studie die Anzahl der verschiedenen ASes, Länder und Regionen, in der sich die Traceroute-Quellen befinden. Die Abbildung 1 zeigt, wie die 2,3K LGs auf der Welt verteilt sind. Die Abbildung 3 zeigt, wie viele LGs pro AS verfügbar sind. Einige ASes bieten eine große Anzahl an LGs an, die weltweit verteilt sind. Viele ASes haben jedoch nur ein LG. Die meisten LGs befinden sich in Europa und Nordamerika.

Region	# LGs	Region	# LGs
Europe	1,361	South America	84
North America	718	Australia & New Z.	58
Asia	104	Africa	4

Abbildung 1: Anzahl LGs nach Standorten [1]

	CAIDA	PlanetLab	DIMES	LG
Sources	26	254	18K	2.3K
AS	26	223	n.a.	406
Countries	18	31	113	66

Abbildung 2: Reichweite verschiedener Datensätze [1]

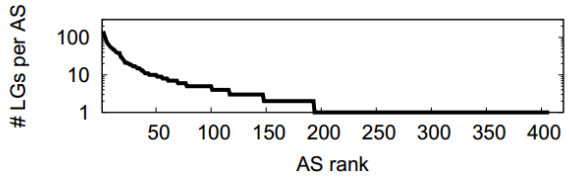


Abbildung 3: Verteilung der Anzahl der LGs pro AS[1]

4. PERISCOPE ARCHITEKTUR

Es gibt vier Hauptprobleme, die Periscope berücksichtigen muss [7]:

- Da es keine Standards für die Eingabe der Anfragen und Ausgabe der Ergebnisse gibt, muss Periscope die Anfragen für jeden LG entsprechend anpassen. Ebenso müssen die Ergebnisse in einem benutzerfreundlichen Format dargestellt werden [7].
- Es gibt keine zuverlässige Liste, die vorhandene, gültige LG Server beinhaltet. Periscope muss in der Lage sein, automatisch LG Spezifikationen zu ermitteln und validieren, die von verschiedenen Quellen stammen [7].
- Die Verfügbarkeit und Spezifikationen der LG Server ändert sich häufig. Periscope muss automatisch die Änderungen ermitteln und Spezifikationen anpassen [7].
- LGs sind für kleine Anfragen gedacht und blockieren somit die Benutzer, die eine bestimmte Anzahl an Anfragen überschreiten. Periscope muss in der Lage sein, gleichzeitig Anfragen von verschiedenen Benutzer auszuführen ohne die Grenzen von LGs zu verletzen [7].

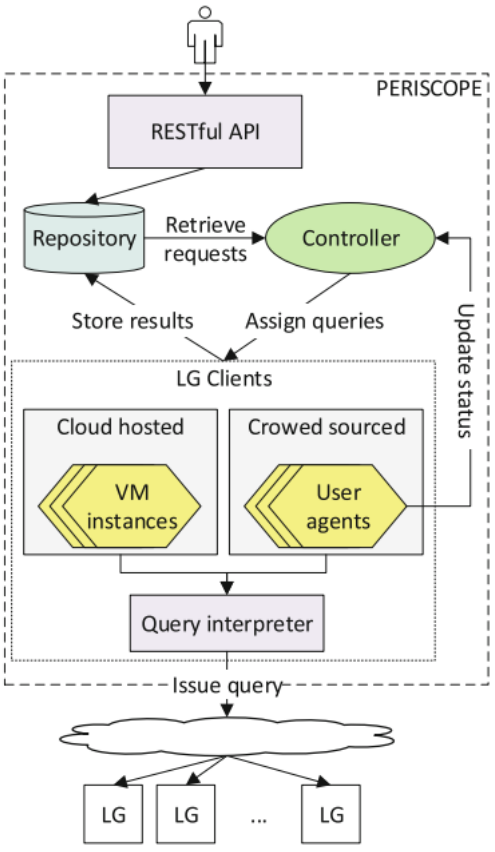


Abbildung 4: Pericope Architektur [7]

4.1 Arbeitsablauf vom Periscope

Als erstes holt Periscope die Listen der verfügbaren LG Server, die noch aktiv sind. Dafür werden öffentlich verfügbare Ressourcen, wie PeeringDB[16] und *traceroute.org*, die LG URLs beinhalten, verwendet. Diese Liste kann jedoch veraltet sein und auch nicht mehr aktive LGs auflisten. Deshalb besucht ein Web Crawler die URLs der LG Server und filtert dabei nicht erreichbare LGs heraus.

doi: 10.2313/NET-2016-09-1_07

Um festzustellen, ob die URLs überhaupt auf LGs verweisen, wird im HTML Quelltext überprüft, ob dieser ein Element der Form eines LG beinhaltet. Die meisten LGs basieren auf Open-Source Projekte basieren.[2] Es wurde für sieben bekannte Open-Source Projekte jeweils eine Vorlage der Implementierung erstellt, welches die HTTP Elemente und HTML Parameter beschreibt.

Ein Web Scraper extrahiert die <*Form*> Elemente aus dem HTML Quelltext und vergleicht die Eingabefelder mit den vorhandenen Vorlagen. Wenn mindestens eine Vorlage alle Eingabefelder aus dem HTML Quelltext extrahierten Felder enthält, gilt diese als passende Vorlagen. Dabei können die extrahierten Elemente eine Teilmenge der Elemente der Vorlage sein.
Sobald eine passende Vorlage gefunden ist, erstellt Periscope eine JSON Konfigurationsdatei, die das LG-Interface beschreibt. Des weiteren enthält die Konfigurationsdatei die HTTP-Anfrage und die Eingabeparameter mit zulässigen Eingabewerten. Diese JSON Konfigurationsdatei wird vom *Query Parser* verwendet, um die Messanfragen in ein Format zu übersetzten, das vom jeweiligen LG unterstützt wird. Findet der *Web Scraper* jedoch keine passende Vorlage, so sucht er nach LG-spezifischen Schlüsselwörtern wie z.B Netzwerkbefehle. Wird ein Schlüsselwort gefunden, so muss das Formelement manuell geparst werden. Im Anschluss wird eine aktualisierte Vorlage erstellt, um LGs ähnlicher Form in Zukunft automatisiert erkennen zu können. Abbildung 5 zeigt eine Beispielvorlage für die Eingabeparameter des Version6 [25] LGs. Der letzte Schritt ist es die Richtigkeit der automatisch generierten Konfigurationsdateien sicherzustellen. Ein *Health Checker* verwendet den *Query Parser*, um eine Messung anzufragen und verarbeitet das Ergebnis, das der Parser zurückliefert. Wird ein HTTP-Fehler zurückgeworfen oder handelt es sich um ein leeres Ergebnis, so markiert der *Health Checker* den verwendeten LG für eine manuelle Untersuchung. Diese Tests werden vom *Health Checker* regelmäßig durchgeführt, um die Änderungen in LG Vorlagen, Eingabeparametern oder in HTTP-Rückmeldungen festzustellen [7]. Abbildung 6 zeigt den Arbeitsablauf, um LGs zu entdecken und deren Spezifikationen von Webquellen zu extrahieren.

Input name	Input type	Expected values	Meaning
query	radio	[bgp, trace, ping]	[sh ip bgp, traceroute, ping]
addr	text	*	Query target
router	select	*	Router identifier
protocol	select	[IPv4,IPv6]	IP version

Abbildung 5: Beispielvorlage der Eingabeparameter von Version6 LG[25]

4.2 Komponenten der Periscope-Architektur

Periscope stellt *RESTful* API bereit, die verwendet wird, um Anfragen an vorhandenen LGs zu schicken, neue Messungen durchzuführen und Ergebnisse abzurufen. Abbildung 4 zeigt die verschiedenen Komponente von der Periscope Architektur. Alle Anfragen werden in einem *Repository* gespeichert. Das dient der Vermittlung zwischen Periscope API und den restlichen Komponenten von Periscope.
Ein *LG Client* fragt die in der Repository gespeicherten Anfragen an und übersetzt diese in LG-Abfragen. LG Clients

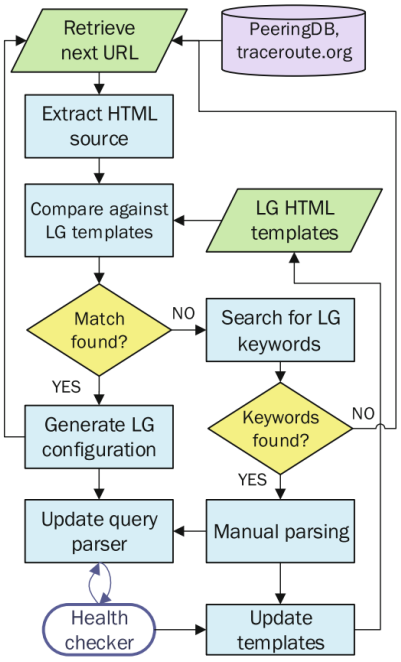


Abbildung 6: Arbeitsablauf um LGs zu entdecken und Spezifikationen zu extrahieren[7]

verwenden Selenium [21], um die Anfragen auszuführen. Selenium ist eine Web Browser Automation Suite welche mit LGs anhand der JSON Konfigurationsdateien interagiert.
LGs sind nicht für eine großen Anzahl an Anfragen gedacht. Überschreitet ein Client eine bestimmte Anzahl an Anfragen in einem bestimmten Zeitintervall, so wird er vorläufig blockiert. Es wird zum Beispiel die IP-Adresse vom Client und die Zeit der Anfrage gespeichert, damit der LG Server entscheidet, ob er die nächste Anfrage vom selben Client zulässt oder nicht. Aus diesem Grund kann Periscope nicht gleichzeitig Anfragen von unterschiedlichen Benutzern, die an den gleichen LG Server gerichtet sind, von einer öffentlichen IP-Adresse verarbeiten. Dieses Problem wird gelöst, indem jedem LG Client eine andere öffentliche IP-Adresse zugeordnet wird und somit die entsprechenden Anfragen von unterschiedlichen IP-Adressen an den LG Server gesendet werden.
Der erste Ansatz, um unterschiedliche öffentliche IP-Adressen für die LG Clients zu verwenden, ist Crowd-Sourced *User Agents* zu verwenden. Der zweite Ansatz ist Cloud-basierte Virtuelle Maschinen (VMs) zu verwenden. Jedem LG Client wird eine Instanz der virtuellen Maschine zugewiesen. Periscope schaltet die VMs bei Bedarf ein und aus. Es sind mindestens so viele LG Clients notwendig wie die Anzahl der unterschiedlichen Periscope-Benutzer, die gleichzeitig Anfragen an einen LG Server schicken. Periscope versucht erst mit dem ersten Ansatz die Nachfrage zu bedienen und schaltet

doi: 10.2313/NET-2016-09-1_07

bei Bedarf virtuelle Maschinen ein.
Ein zentraler *Controller* kommt zum Einsatz, um jeder Messungsanfrage einen LG Client zuzuweisen. Da der Controller einen globalen Überblick vom System hat, werden die Anfragen so bearbeitet, dass die auf einzelnen LG Server eingestellte Abfragegrenzen nicht überschritten werden. Der Controller verwaltet dabei sowohl Cloud-basierten VM-Instanzen als auch die Crowd-Sourced-Instanzen. Letztere schicken alle fünf Minuten eine Keep-Alive-Nachricht, um den Controller mitzuteilen, dass sie bereit sind, weitere Abfragen zu verarbeiten. Sobald eine neue Messanfrage eingeht, entscheidet der Controller wann und welche Instanz die Anfrage verarbeiten soll. Die Logik des Controllers basiert auf zwei LG-spezifischen Variablen, die die maximale Anzahl gleichzeitiger Abfragen an einem LG beschränken[1]:
1. *Timeout* ist die minimale Zeit zwischen zwei aufeinanderfolgenden LG Abfragen von dem selben Benutzer.
2. *Query slots* ist die maximale Anzahl der Abfragen, die Periscope zu einem bestimmten Zeitpunkt für einen LG akzeptieren kann.
Damit ein Benutzer überhaupt eine Anfrage an einem LG schicken kann, muss mindestens ein Query Slot übrig sein. Anderenfalls wird die Anfrage zu einer Schlange hinzugefügt, damit sie später verarbeitet werden kann. Wann die Anfrage zur Verarbeitung bereit ist, entscheidet der Controller. Für jede Anfrage berechnet der Controller den Zeitpunkt, zu dem er die Anfrage durchführen kann. Dieser Zeitpunkt basiert auf dem Zeitpunkt der letzten Anfrage des gleichen Benutzers zum gleichen LG und auf der Timeout-Variable des LGs. Die Anfrage wird so lange in der Schlange gespeichert, bis mindestens ein Query Slot frei wird und die Timeout-Zeit abgelaufen ist. Wenn alle Anforderungen zur Verarbeitung erfüllt sind, wird die Anfrage von der Schlange genommen und einer geeigneten Client-Instanz zur Ausführung zugeordnet. Eine Instanz ist für die Ausführung der Anfrage geeignet, wenn diese für die Dauer der Timeout-Zeit keine Anfragen an der entsprechenden LG geschickt hat. Steht keine geeignete Client-Instanz zur Ausführung bereit, beantragt der Controller eine neue Cloud-Instanz. Die Anzahl der aktiven Client-Instanzen hängt lediglich von der Anzahl der Anfragen ab, die von verschiedenen Benutzer gleichzeitig an einem LG geschickt werden und nicht von der Anzahl der aktiven Benutzer oder angefragten LGs.

4.3 Evaluierung

Periscope hat für 262 LGs automatisch die Konfiguration erstellt. 35 weitere wurden manuell geparst, da es am Anfang keine passende Vorlagen für sie gab. Insgesamt wurden bis Dezember 2015 297 LGs von Autonomen Systemen extrahiert. Es ist nicht immer leicht LGs zu AS Nummern (ASN) zu mappen. Deshalb wird die IP des LG Hosts verwendet, um diese mit Hilfe der Longest Prefix Match Methode zu einer ASN zuzuordnen. Um die IP Adresse von dem Router, die die einzelnen LGs hostet, zu ermitteln, wird ein Traceroute-Befehl zum Router ausgeführt. Mit *tcpdump* wird im Anschluss die Quelladresse von den ankommenden Paketen extrahiert. Dieses Verfahren wird auch verwendet um die Traceroute Protokolle der verschiedenen LGs zu entdecken. 266 LGs verwenden UDP-basiertes Traceroute und 31 LGs ICMP-basiertes Traceroute.

[1]Die Werte der timeout und query slot Variablen wurden dabei empirisch konservativ abgeleitet[7].

Viele LGs unterstützen das Ausführen der Befehle aus verschieden Blickwinkeln (eng. Vantage Points (VPs)) innerhalb eines Autonomen Systems. Wenn das LG-Interface keine Informationen über den Standort liefert, wird die gleiche Methode, wie bei Ermittelung von ASN vom jeweiligen LG, verwendet, um den jeweiligen LG zu einer Stadt zuzuordnen. Dabei wird die IP Adresse des LGs mit Hilfe der NetAcuity's geolocation Datenbank [14] zu einer Stadt zugeordnet. Periscope hat automatisch aus 262 LGs 1.691 VPs geparst. Diese sind über 501 Städten in 76 Ländern verteilt. 40% der LGs haben mehr als einen Blickwinkel auf Stadtebene und 20% der LGs haben 10 oder mehr VPs. Abbildung 7 zeigt, wie viele VPs die jeweiligen Befehle unterstützen. Mehr als 75% der VPs unterstützen Kontrollschicht- als auch Datenschichtmessungen und 60% unterstützen IPv6-Befehle zusätzlich zu IPv4-Befehlen.
Um die Topologien von Periscope, Atlas und Ark VPs zu vergleichen, wurden in Oktober 2015 Traceroute-Befehle zu 2.000 Zielen von jeder Plattform ausgeführt. Zu diesem Zeitpunkt hatte Atlas 7.292 öffentliche Ressourcen in 2.779 verschiedenen ASes, verteilt in 160 Ländern. Ark hatte 107 Ressourcen in 71 ASes, die in 41 Ländern verteilt waren.
Um eine wertneutrale Menge von Zielen zu erstellen, wurden zuerst die IP Adressen gesammelt, die in iPlane Datenbank [13] waren und im Anschluss wurde ein *ZMap* Scan ausgeführt, um nur IP Adressen zu behalten, die auf UDP und ICMP Befehlen geantwortet haben. Die IP Adressen wurden zu deren zugehörigen ASes gemappt und für jedes AS wurde zufällig eine IP Adresse ausgewählt, bis eine Zielmenge von 2.000 IP Adressen erreicht wurde. Eine kleine Zielmenge ist nötig, da die LGs und Atlas Infrastrukturen nur eine beschränkte Anzahl an Anfragen zulassen. Es wurden Messungen von allen Atlas Netzwerk Monitoren ausgeführt. Mehr als 6 Millionen Traceroute-Daten wurden dabei innerhalb von zwei Monaten gesammelt. Dies war nur mit einem Account mit höheren Rechten möglich. Mit einem Standardaccount hätte es ungefähr fünf Jahre gedauert [18].
Traces von LG VPs überquerten 3.109 ASes, 29.525 AS-Links und 167 IXPs. Die Traces von der Atlas Plattform überquerten 3.369 ASes, 55.936 AS-Links und 171 IXPs zu gleichen Zielen. Bei Ark waren es 1.608 ASes, 10.237 AS-Links und 136 IXPs. In der Tabelle1 ist die Anzahl der ASes, AS-Links und IXPs von jeweiligen Datensätzen aufgelistet, auch die, die eindeutig sind. 47% (13.969 von 29.525) AS-Links waren nicht in den Tracedaten von Atlas oder Ark enthalten und 26% (809 von 3.109) ASes von LG Traces waren nicht in Atlas oder Ark Datensätzen enthalten. 16 IXPs die in LG Trace-Datensätzen gesichtet wurden, waren nicht in den Datensätzen von Atlas oder Ark Traces.

Die neu entdeckten ASes und AS-Links erweitern bisherige AS-Topologien. Mithilfe von Periscope können viele weitere, bisher unentdeckte, ASes und Links ermittelt werden. Das Konzept zeigt, wie man verschiedene LGs effizient verwenden kann, ohne dabei deren Grenzen zu verletzten.

4.4 Test-Messungen

Die API von Periscope ist auf der offiziellen Website von CAIDA [4] dokumentiert. Hier [5] ist ein Beispielbefehl für eine neue Messung aufgelistet. Um eine neue Messung durchzuführen, braucht man Zugangsdaten, die von CAIDA auf Nachfrage zur Verfügung gestellt werden. Da kein Kontakt mit CAIDA hergestellt werden konnte, war es auch nicht

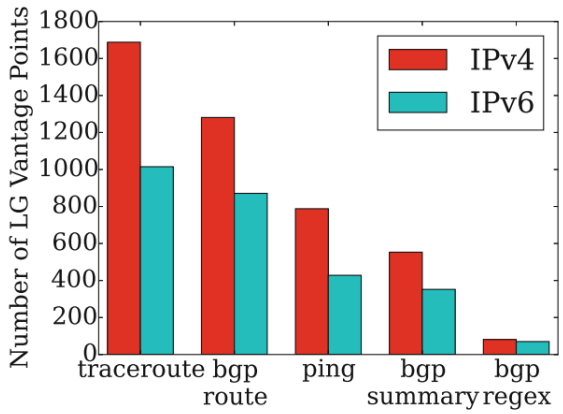


Abbildung 7: Anzahl der VPs verschiedener Befehle[7]

Tabelle 1: Anzahl ASes und Verbindungen dazwischen im LG, Atlas und Ark Datensätzen[7]

Dataset	ASes		AS Links	
	Gesamt	Eindeutig	Gesamt	Eindeutig
LG-Periscope	3.109	809	29.525	13.969
Atlas	3.369	1.464	55.936	40.620
Ark	1.608	59	10.237	1.625
LG Khan [12]	45.400	686	116.000	11.000
IXP: Mapped	3.300		36.000	29.600

möglich, Messungen durchzuführen. Aus dem selben Grund konnten auch keine neue LGs hinzugefügt werden.

5. ZUSAMMENFASSUNG

Messungen im Internet und dazugehörige Monitoring-Tools sind wichtig, um die aktuelle Netzwerkstruktur besser zu verstehen, Störungen in Netzwerken zu finden und um diese schneller beheben zu können und die Struktur des Internets in der Zukunft verbessern zu können. LGs spielen dabei eine sehr wichtige Rolle, denn diese erlauben sowohl Traceroute-Befehle, Ping-Befehle als auch BGP-Befehle und bieten somit Kontrollschicht- als auch Datenschichtmessungen an. Da die verschieden LGs unterschiedlich implementiert sein können, wird die Zusammenarbeit dieser verschiedenen LGs erschwert. Mehrere Studien stelle Konzepte vor, die die Zusammenarbeit unterschiedlich implementierter LGs ermöglichen. Periscope ist eines dieser Konzepte, das die LGs verschiedener Provider zusammenfasst. Es ermöglicht den Benutzern Anfragen an beliebigen LGs zu schicken, ohne direkt mit den entsprechenden LGs zu interagieren und sich Gedanken über Timeout-Limits und Anfragen-Limits zu machen oder selber Code zu entwickeln, um automatisiert Anfragen an LGs verschicken zu können. Periscope erweitert die AS-Topologie, die anhand der Daten von Atlas und Ark erstellt werden, um weitere wichtigen Daten. Periscope berücksichtigt die Begrenzung von LGs und erlaubt den Benutzern nicht, mehr Anfragen an LGs zu schicken, als sie in einem gegeben Moment verarbeiten können. Es werden dabei von Periscope mehrere Cloud-basierte und Crowd-Sourced-Instanzen verwendet. Es ist auch wichtig, dass die von LGs festgelegten Grenzen nicht überschritten werden, denn sonst würde die Verfügbarkeit der LGs für öffentlichen Zwecke reduziert werden[7]. CAIDA's Archipelago [3] Infrastruktur bietet laut dem Paper [7] bereits 132 VPs, die als LG Client-Instanzen verwendet werden können.
Die Ermittlung unentdeckter AS-Links aber ein unvollständig gelöstes Problem. Es wird weiterhin in diesem Bereich geforscht, um bisher nicht entdeckte ASes, AS-Links und IXPs ausfindig zu machen.

6. LITERATUR

[1] B. Augustin, B. Krishnamurthy, and W. Willinger. IXPs: Mapped? In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 336–349, New York, NY, USA, 2009. ACM.

[2] Bruno L., Graziano M., Balzarotti D., and Francillon A. Through the looking-glass and what eve found there. In *WOOT (2004)*.

[3] CAIDA ARK. `http://www.caida.org/projects/ark`.

[4] CAIDA Periscope API Dokumentation. `http://www.caida.org/tools/utilities/looking-glass-api/`.

[5] CAIDA Periscope API neue Messung. `http://www.caida.org/tools/utilities/looking-glass-api/#create`.

[6] Y.-J. Chi, R. Oliveira, and L. Zhang. Cyclops: The AS-level connectivity observatory. *SIGCOMM Comput. Commun. Rev.*, 38(5):5–16, Sept. 2008.

[7] V. Giotsas, A. Dhamdhere, and k. claffy. Periscope: Unifying Looking Glass Querying. In *Passive and Active Network Measurement Conference (PAM)*, Mar 2016.

[8] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy. Lord of the links: A framework for discovering missing links in the internet topology. *IEEE/ACM Trans. Netw.*, 17(2):391–404, Apr. 2009.

[9] Internet Routing Registry. `http://www.irr.net`.

[10] Internet2. `http://ndb7.net.internet2.edu/bgp`.

[11] P. M. Kevin Butler, Toni Farley and J. Rexford. A survey of BGP security issues and solutions. In *Proceedings of the IEEE, vol. 98, no. 1*, January 2010.

[12] A. Khan, T. Kwon, H.-c. Kim, and Y. Choi. AS-level topology collection through looking glass servers. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 235–242, New York, NY, USA, 2013. ACM.

[13] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 367–380, Berkeley, CA, USA, 2006. USENIX Association.

[14] Netacuity. `http://www.digitalelement.com/solutions/`.

[15] Packet Clearing House. `http://www.pch.net/resources/data.php`.

[16] PeeringDB. `http://www.peeringdb.com`.

[17] PlanetLab. `http://www.planetlab.org`.

[18] RIPE Atlas rate. `http://atlas.ripe.net/docs/udm/#rate-limits`.

[19] RIPE-RIS. `http://www.ripe.net/ris`.

[20] RouteViews. http://www.routeviews.org.
[21] Selenium browser automation suite. http://www.seleniumhq.org/.
[22] T. Kernen, "traceroute.org" 2008. www.traceroute.org.
[23] Y. T.Li and S. Hares". A border gateway protocol. *RFC 4271*, January 2006.
[24] UCLA IRL. http://irl.cs.ucla.edu/topology.
[25] Version6 Loooking Glass. https://github.com/Cougar/lg.
[26] B. Zhang, R. Liu, D. Massey, and L. Zhang. Collecting the internet AS-level topology. *SIGCOMM Comput. Commun. Rev.*, 35(1):53–61, Jan. 2005.

doi: 10.2313/NET-2016-09-1_07

NFV and OPNFV

Stanislav Guerassimov
Betreuer: Edwin Cordeiro, Lukas Schwaighofer
Seminar Innovative Internettechnologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: s.guerassimov@in.tum.de

ABSTRACT

The demand for flexible and economically viable cloud computing and virtualization technologies is continuously increasing. In this paper we discuss how Network Function Virtualization (NFV) enables simplified network function management and other benefits and challenges that it brings. We also note major NFV standardization efforts, while describing OPNFV in depth. Further, this paper introduces a couple of use-cases for NFV, such as deployment alongside SDN in a datacenter and even deployment in industrial automation systems. We also provide a collection of analyses of existing pilot implementations of NFV. As a result, this paper should provide the reader with an up-to-date state of network function virtualization technologies.

Keywords

NFV OPNFV network function virtualization

1. INTRODUCTION

Virtualization technologies have become a big subject over the recent years. Commodity hardware has become more powerful, thus allowing flexibility in IT operations as well as significant economical benefits. The growing demand to data size and processing speed, as well as the ability to scale in order to accommodate new demands for those parameters has been treated with virtualization technologies. The result was scalable cloud architectures. Network inflexibility has become a bottleneck for those growing demands. So it simply made sense to apply the same approaches to the network.

In order to avoid confusion we will address several terminological questions. From a formal perspective, virtualization can have a lot of different meanings. In the context of network and datacenter operations virtualization means the ability to run several full operating system on a software platform as if they were running on several distinct hardware platforms. Virtualization in networking historically meant multiplexing several network flows over a shared physical link, something often referred as a tunnel. Technically, this means the virtualization of layers 1-3 of the OSI model. NFV on the other hand virtualizes layer 4-7.

The overall concept of NFV should become more clear as the next section explores it in depth, together with technical aspects and challenges. Section 2.2 presents various attempts at standardization of NFV implementations. A particular standardization effort, OPNFV, is analyzed in section 3, along with existing OPNFV versions, projects and architecture. The reader will become familiarized with theoretical concept designs for NFV, including NFV in the datacenter and even in industrial automation systems in section 4, as well as with existing industry implementations of NFV in section 5.

2. NFV

Unlike server virtualization, network virtualization is not a mature technology that is widely adapted.

Network virtualization has the same goals as server virtualization. Analogous to those technologies, NFV should allow the network and it's functions to appear to hosts as a real physical ones. This should allow network engineers to deploy previously physical networking devices as software applications. Examples include firewalls, routers, switches, load-balancers, WAN optimizers, IP Multimedia Subsystems (IMS), Evolved Packet Cores (EPC) and Deep Packet Inspection (DPI). The network architecture itself will change, as all devices are now running on a single hardware environment. Figure 1 provides a basic overview, which shows how hardware functions are transformed into virtualized network functions (VNF). In the next section we will discuss various advantages and challenges of NFV.

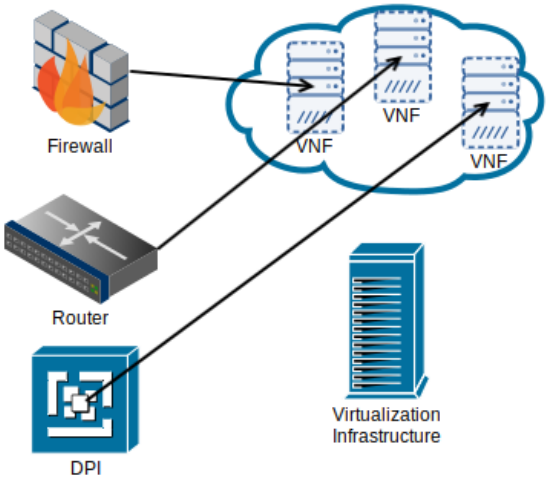


Figure 1: NFV concept

doi: 10.2313/NET-2016-09-1_08

2.1 Technical aspects

The concept of NFV introduces numerous benefits to network operations. Arguably, only by utilizing NFV we can achieve the full benefits of OS virtualization, since physical networks bottleneck the capabilities of the former.

The hardware liberation that NFV brings changes several aspects about network operations. The first is the cost of hardware. Function consolidation reduces the amount of required running hardware for high availability. Previously a common proper practice was to purchase spare hardware units for each working unit. The amount of devices standing by can now be reduced. This will also reduce energy consumption, a benefit inherited from virtualization of operating systems. It could be reduced by minimizing the amount of online hardware thanks to consolidation.

Hardware costs are also reduced because specialized devices are replaced by commodity of the shelf (COTS) hardware. This also means that network engineers become decoupled from vendor solutions. There is no need to relearn new hardware, no need to wait for a vendor to implement a new feature or suffer price extortion from network equipment producers.

NFV also lowers the network industry barrier to smaller companies and open source communities, so that they can also develop network solutions. Since network technologies are now software based the amount of competition among vendors increases. Smaller vendors that were unable to enter the market, due to the impossibility of launching their own hardware production lines, can now have a competitive edge against large network technology giants, for example by offering customers specialized solutions.

NFV can reduce the complexity of testing new network functions. Building a test environment becomes easier than before, since deploying a new virtual machine is significantly easier than purchasing new hardware. Another aspect is the benefits functions provided by virtualization bring, for example snapshotting makes it easier to test new settings with the ability to quickly revert to the last stable configuration. When it comes to launching those functions, the process will no longer entail purchasing new networking hardware, main and spare, provisioning space for that hardware, power sources, temperature control.

Another important aspect is security. By replacing network devices with virtualized applications, the risk of unauthorized hardware access becomes close to impossible, which alleviates a renown network security attack vector.

It is necessary to notice, that even though we often mention security as a technical advantage of all virtualization technologies, this fact remains to be a highly debated topic. While making a step forward in terms of security, at the same time we make two steps backward. Even without going into detail, simply the increase of the code base that implements virtualization gives a larger attack surface for malicious intents. Existing NFV security approaches recommend applying corresponding hardening techniques to each separate security domain of the virtualization scheme [4, page 20,22].

Another challenge that NFV introduces is performance. In the beginning of NFV development, hardware platforms were not completely ready for network virtualization. It turned out that virtualization poses a great amount of issues for network performance. It turned out that transferring packets from the physical connection to the virtual guest required a significant increase of the amount CPU interrupts. This created a major bottleneck for all virtual network function implementations.

Vendors have reacted to these problems, for example Intel has introduced technologies such as Virtual Machine Device Queues VMDq and Single Root Input/Output Virtualization (SR-IOV). VMDq allows each virtual machine to recieve network messages on it's own queue, removing the necessary interupts of the hypervisor CPU [10]. SR-IOV allows the NIC to provide seperate resources, such as memory space and a transmit and receive queue, directly to a VM. According to Intel, this should allow VNF to achieve 10Gb on selected hardware [9].

These solutions are considered to be a compromise: the hypervisor bypassing techniques that are utilized by SR-IOV pose restrictions on the possibility to orchestrate network functions, since SR-IOV is dependent on hardware support.

Another challenge is the organizational aspect of NFV. NFV is a distinctly new concept in networking. In order to implement it, we must consider not only technical aspects, but also organizational issues. Not only must NFV users replace heaps of existing networking hardware with new commodity off the shelf hardware (COTS), but they should also reassess existing processes. Switching to a NFV environment is a serious investment which requires engagement from all departments - IT, marketing, top management, sales and R&D. While some telecommunication companies have began integrating NFV, the return of investment is not easy to estimate for a large amount of other companies. A reason for this is that simply virtualizing previously available hardware network functions is just a minor step towards becoming a cloud telecommunications provider, a step which brings practically little business value. A large percentage of companies call high level organizational issues their major roadblock for NFV [2].

Lack of a standard orchestration tool is also a major impediment. At the moment of writing the purposed effort to bring management and orchestration (MANO) to the NFV standard OPNFV is barely half an year old. This is expanded in the MANO section.

The fact that NFV allows COTS hardware to be used creates another challenge: vendors experience reluctance towards NFV. This is not really a technological issue of NFV in general, as much as a business model one. Large network equipment companies are not eager towards NFV development since it is very likely to damage their profits. Key problems are:

- NFV makes existing developed technologies that took years to develop obsolete
- NFV is a vendor agnostic technology that frees com-

doi: 10.2313/NET-2016-09-1_08

panies from proprietary lock-in. Companies have no incentive to achieve that.

As a result, it can be expected that equipment producers might try to develop their own version of NFV.

As mentioned earlier, hardware replacement is a burden even for the network owners. Companies have spent a great amount of money and years building their network systems. Replacing recent hardware and existing support contracts is not an attractive decision, especially while some companies are still struggling to get rid of their legacy network technologies.

2.2 NFV Standards

A major complaint among companies interested in NFV was the lack of guidance in the implementation of NFV [20]. Particularly the telecommunications industry has strict requirements for security, performance and reliability, therefore it is fair to state that the success of NFV relies on standards. Several institutes have been making efforts in the standardization of NFV.

2.2.1 TM Forum

TM Forum (formerly TeleManagement Forum) decided to consolidate their operations management expertise across their members in order to realize a "Zero-touch, Orchestration, Operations and Management" (ZOOM) strategy for NFV implementations. The ZOOM project is supposed to build a new clear architecture blueprint that will enable flexible and agile virtual network services [23]. TM Forum focuses on bringing together organizations interested in NFV to address various challenges of the technology, while working together with ETSI and other efforts. An example of such work groups are various Catalysts, which are "rapid fire, member-driven proof-of-concept projects which both inform and leverage TM Forum best practices and standards" [22].

2.2.2 OASIS TOSCA / IETF YANG

Developed by The Organization for the Advancement of Structured Information (OASIS), the Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standard language built specifically for orchestration of different cloud based web services. The TOSCA NFV profile specifies a Network Functions Virtualisation (NFV) specific data model using TOSCA language [12]. Complementary with YANG, a data modeling language for the Network Configuration Protocol created by the IETF [7], TOSCA can be used to manage and deploy NFV in an automated manner, where YANG is responsible for configuration and TOSCA is used for the orchestration mechanism.

2.2.3 Linux Foundation

In September 2014 the Linux Foundation has founded OPNFV. The goal of the organization is to accelerate the development of NFV technology by developing an integrated open source ecosystem that includes existing open source software an allows new solutions to be developed with the participation of vendors [13].

3. OPNFV

It has been 2 years since the Open Platform for NFV Project (OPNFV) has began working, and has already delivered on two software versions and developed an expansive series of questions regarding requirements, architectures and use cases for NFV.

3.1 OPNFV versions

3.1.1 OPNFV Arno

8 Months since it's creation, OPNFV was ready to release the first software version called Arno. It's main elements featured the Linux based virtualization solution KVM as hypervisor environment and an OpenStack architecture with an OpenDaylight-based SDN controller. Key capabilities of OPNFV Arno included:

- the ability for the users to deploy NFVs on the platform to test their functionality and performance
- a continuous integration toolchain that allows projects to do automatic builds and verification as Open-Source components are developed independently.

It was expected that Arno would attract users to explore the platform and it's capabilities to satisfy their networking requirements, therefore accelerating NFV integration in the industry. [14]

3.1.2 OPNFV Brahmaputra

Brahmaputra is the second and latest version of OPNFV. According to Chris Price, technical steering committee chair of the project: "Building on the foundation of Arno, the OPNFV community worked tirelessly to integrate and combine components from multiple communities to deliver Brahmaputra, which brings end-to-end feature realization" [18].
Brahmaputra includes almost double the projects that Arno had and brings enhancements such as: layer 3 VPN management, initial support for IPv6, better fault detection and recovery, developements with Data Plane Development Kit (DPDK) for data plane performance boosts and improved infrastructure testing capabilites [18].

3.2 OPNFV projects

OPNFV manages activities in form of projects, created by the Technical Steering Committee. Each project has a team of commiters, which manages it and reviews contributions. As of writing, OPNFV has little over active 40 projects [15]. Many of these projects are critical for the success of NFV, for example OPNFV DPACC, that addresses the issues in data plane performance that where discussed in section 2. Describing every single one of them in detail is not within the scope of this paper, so only a couple would be described.

3.2.1 OPNFV Doctor

OPNFV Doctor is a project that maintains network function status awareness and uses a notification system to warn in changes of the statuses. The Doctor project is a key enabler of High-Availablity (HA) for network functions, since it can signal Virtual Network Function Managers to take recovery actions once a failure has accured [16].

doi: 10.2313/NET-2016-09-1_08

3.2.2 *OPNFV Prediction*

The Prediction system consists of a data collector, a predictor and a management module. Tools such as OpenStack's Celiometer and Monasca are used for data collection and are interact with other tools such as Zabbix or Nagios. Prediction runs real-time analysis and applies machine learning techniques on the data provided by the collection tools and then sends notifications to the Virtual Network Function Managers that will act upon the notification [19].

3.2.3 *OPNFV MANO*

In 2015 the OPNFV has approved the MANO "requirements for integration" founded by Telefónica, RIFT.io, Mirantis, Intel, and Canonical. Until that moment, OPNFV was limited to virtualized network function and accompanying lower layer management [17]. By including MANO, OPNFV now openly works on a full NFV reference implementation. According to industry representatives, the lack of a management and orchestration layer was a common reason that kept operators reluctant towards NFV.

3.3 OPNFV Architecture

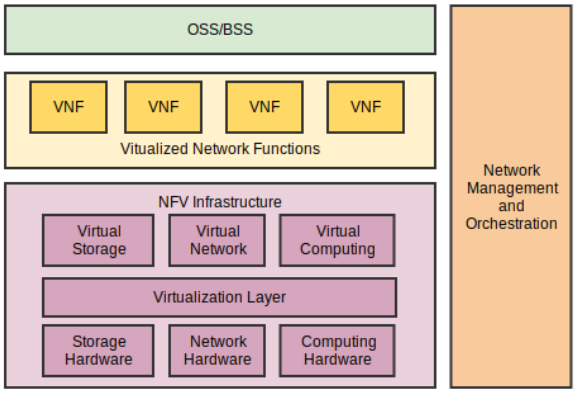


Figure 2: OPNFV architecture

The current top-level state of the OPNFV project structure is displayed on Figure 2 according to [15].

- Virtual Network Function
  Technically, a virtual machine (VM) or a group of VMs that realize the given Virtual Network Function (VNF). VNFs are controlled by the Element Management System.
- Network Function Virtualization Infrastructure (NFVI)
  The physical hardware that constitutes as the infrastructure of the network, their configurations. These include the servers, storage devices, virtual switches, hypervisors and others.
- NFV Management and orchestration
  The part that controls VNFs and the underlying infrastructure and their operations. The management component controls VNFs and the Orchestration controls the NFVI.
- Operations/Business support systems (OSS/BSS)
  Systems for monitoring, control, billing, provisioning, into which NFV are integrated either through MANO, or one by one through direct interfaces.

4. USE CASES

It is always important that any new technology does not remain to be a solution looking for a problem. NFV can be proven to be anything but that, by viewing it from several perspectives: helping cloud providers in datacenter management, helping Internet service providers by easing customer service and even bringing flexibility in factories of the future to a new level.

4.1 Datacenter network optimization

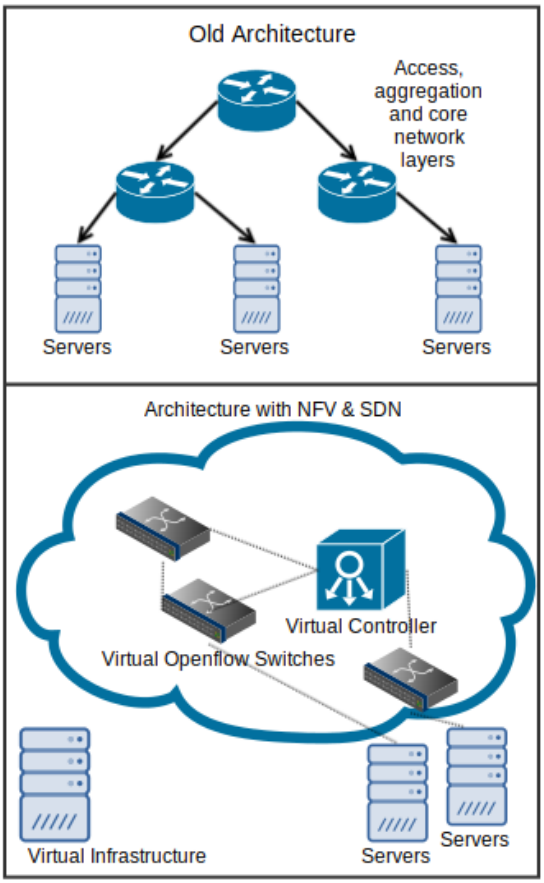


Figure 3: SDN and NFV in the datacenter

NFV pared with Software Defined Networks (SDN) can bring a new level of flexibility to modern datacenters. A rough explanation of SDN is that SDN is an effort to separate the control and data plane, while using a standardized and centralized controller. The controller manages forwarding rules in the data plane by sending messages to SDN switches. The currently popular implementation of SDN is a standard called OpenFlow. By uniting NFV and SDN in their datacenters, organizations can finally consolidate their regular systems operations with network operations, allowing more flexibility, scalability and availability. SDN simplifies the orchestration of virtual SDN switches by abstracting the configuration and those network functions all feature benefits of virtualization discussed in section 2. The SDN controller

can also be virtualized. The concept comparing the old architecture and NFV/SDN one is illustrated on figure 3.

4.2 Customer Gateway Virtualization

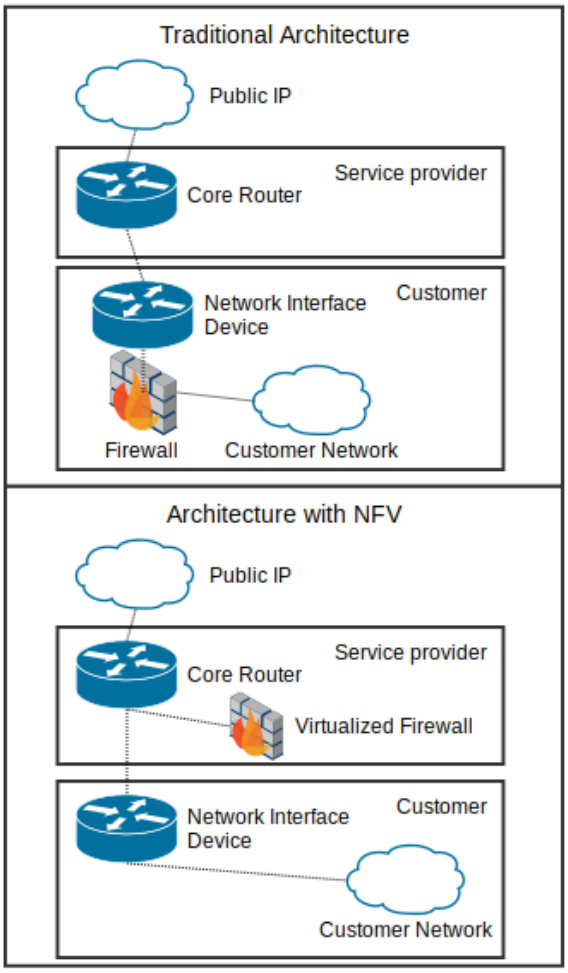


Figure 4: Deployment of a firewall

When a new customer connects to the network of an internet service provider (ISP), usually the ISP assists in hardware installation: either on site, or by providing device settings. Some customers would appreciate additional network services, such as firewalls, VPN support, and DDoS protection. Often the customers are small and medium businesses without an established IT team. The operation of supporting these setups is not a straightforward task, especially since the customer requirements for their network are hard to predict.

NFV can allow the ISP to provide the customer with a simple gateway device which will have all of the network function logic on the servers of the provider. Figure 4 illustrates an example of how a service such as a firewall can be deployed. Large organizations can also apply the same principle inside their corporate intranets to organizational units, while hosting the network functions in their own datacenter.

This will allow end users to rent a network service, while the ISP handles everything behind the curtains. Since deploying a new virtual network function is much faster than a piece of hardware at the users site, this brings a series of advantages: functions are deployed more quickly and have increased elasticity, meaning that functions can be deployed promptly at any time.

4.3 NFV for Industrial Systems

Industrial automation has been considered to be a conservative field, though various industries have already began adapting technologies from the IT world, including networking and virtualization.

In the last decades, controlling machines in industry automation and in Supervisory Control and Data Acquisition (SCADA) systems have begun using more and more technologies and standards which are built upon Ethernet and Internet technologies, instead of the usual CAN, Fieldbus and EtherCAT. For example, the Siemens SIMATIC S7-300 controller has support for the CP 343-1 Lean communcations processor, that add support of TCP/IP, UDP, and Ethernet to the controller. [21]

Industrial automation systems have been receiving new requirements such as adaptability and distribution. This has even led to IEC developing a new standard under the code 61499. The IEC 61499 standard provides the architecture, tools and rule of compliance for development of distributed and reconfigurable automation and systems [6]. The pharmaceutical industry is a great example of a field where quick reconfigurablity could give benefits - the product range changes quickly and the software logic behind production needs to changed often, much more often than the hardware.

An example of such an existing production system virtualization deployment is a New England Controls project in a pharamaceutical company Biogen [11]. The project was deemed to be a success. More similar proofs of concept show that this is just a beginning. Another example is a TM Forum Catalyst (discussed in section 2.2) called Smart Industrial Manufacturing which explores a concept "Robot As a Service", where multiple robot cells can be reconfigured through a hardware abstracted API. [24]

IEC 61499 introduces a number of principles that can appear to be analogous to those that exist in the virtualization scene, as well as NFV and even SDN. Control applications are distributed to control devices in a similar way how virtual machines are deployed to hypervisors. Some similarities could be drawn between the separation of data and event flow proposed by IEC 61499 and the data and control plane separation concept ingrained in SDN technologies. [6]

It is logical to assume that the next step would be to adopt network function virtualization in industrial components and systems, once manufacturers become more confident in the capability of virtualization technologies to meet their requirements. It is also fair to expect that a standardization effort will be crucial for the adoption of NFV, particularly requirements for real-time operation.

5. PROOF OF CONCEPTS

According to IHS, 82% companies have evaluated or deployed SDN in their networks, though the deployment scale remains very small. SDN is a more mature technology than NFV, though already 35 % of companies surveyed planned to deploy NFV in 2015 [8]. Looking in to existing deployments should give an idea how NFV is expanding in the industry.

5.1 CMCC in Shaanxi

In September 2015, China Mobile began a pilot deployment of a cloud-based network core. This started a period of NFV testing in live mobile networks in China. In this pilot project, Huawei, together with the Shaanxi Branch of China Mobile, performed a comprehensive set of verifications and a small-scale field trial of the cloud-based network [5]. Even though Shaanxi is only a small province of China, CMCC has about 24 million customers in the network, of which 6 million use LTE provided by 29 thousand LTE stations [1].

According to Huawei, "the trial was aimed at verifying the cloud-based networking, technical specifications, service capabilities, maintenance, and full lifecycle management, laying a solid foundation for large-scale trial and commercial use of cloud-based core networks in the future [5].

On December 10, 2015, China Mobile made the first VoLTE call over the pilot NFV-based network. This cloud-based VoLTE call demonstrated that the IMS system deployed in a multi-vendor hardware environment had successfully connected with a live network. Achieving this step was not possible without problems. Key issues encountered during the pilot project were [1]:

- High Equipment cost. Even though commodity hardware from multiple vendors could now be used, the purposed benefit of cost reduction didn't realize. Up to 600 servers were purchased, but due to organization problems average load stayed at about 20%.
- Slow Time to Market, up to 9 months
- High Servicing costs
- Consolidation organization problems, due to various departments owning hardware.

5.2 Orange Global Software-Defined WAN

ClearPath together with Orange and Intel demonstrated a joint proof of concept at the 2015 OPNFV summit that showed secure, distributed virtual Customer premise equipment, a concept discussed in section 4. According to Clearpath, their "NanoServices, open source VNFs, and containers significantly reduce vCPE costs for each end user" [3]. The report confirms that customer gateway virtualization, built according to the OPNFV reference implementation, is a viable NFV deployment.

6. CONCLUSION

In this paper, we have examined the novel technology of network function virtualization. NFV has been gaining an increasing amount of interest from the networking industry. Modern datacenters, cloud providers and telecommunication companies are processing ever-increasing data volumes, which makes network agility a most important goal. NFV is proven to be a key enabler of the demanded flexibility.

We have examined the concept of NFV and currently existing standards, including one particular standard, OPNFV, which was analyzed in greater detail. OPNFV has been making serious advances in building an effort to provide a much needed guidance in NFV implementation. An increasing amount of projects backed by large vendors induces the support that it is receiving from the interested industry members.

Current use cases and proof of concepts demonstrate that NFV brings it's benefits to existing and new networking models, such as operator networks and customer gateway virtualization. It's advantages could serve not only operators, but also industrial systems and data centers. We concluded that the standardization efforts will probably need to be developed further in order for NFV to enter new fields, such as the automation industry.

We observed that NFV still has certain challenges to go through. Key issues have to be addressed before NFV adaptation becomes more widespread. Nevertheless, we believe NFV is going to be one of the most revolutionary technologies for networking in the following decade and the development of standards like OPNFV is the key towards advancements in network function virtualization.

7. REFERENCES

[1] Aleksey Shalaginov. Operator SDN: Implementation Attempt (Operatorskie seti SDN: Opyt realizacii), 2016. https://shalaginov.com/2016/05/16/%d0%be%d0%bf%d0%b5%d1%80%d0%b0%d1%82%d0%be%d1%80%d1%81%d0%ba%d0%b8%d0%b5-%d1%81%d0%b5%d1%82%d0%b8-sdn-%d0%be%d0%bf%d1%8b%d1%82-%d1%80%d0%b5%d0%b0%d0%bb%d0%b8%d0%b7%d0%b0%d1%86%d0%b8%d0%b8/; last accessed on 2016/06/19.

[2] Dr. Jim Metzler, Ashton, Metzler and Associates. When Will NFV Cross the Chasm. http://ashtonmetzler.com/When%20will%20NFV%20Cross%20the%20Chasm.pdf; last accessed on 2016/06/19.

[3] Dustin Do. ClearPath teams up with Orange and Intel to enable secure managed services for the enterprise, 2015. http://www.clearpathnet.com/clearpath-teams-up-with-orange-and-intel-to-enable-secure-managed-services-for-the-enterprise/; last accessed on 2016/06/19.

[4] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE Network*, 28(6):18–26, Nov 2014.

[5] Huawei. NFV Has Entered the Fast Lane: Core Networks will be the First to Take a Stroll in the Clouds, 2016. http://www.huawei.com/en/mwc2016/topics/nfv-has-entered-the-fast-lane; last accessed on 2016/06/19.

[6] IEC. IEC 61499 - The New Standard In Automation.

http://www.iec61499.de/; last accessed on 2016/06/19.

[7] IETF, M. Bjorklund, Tail-f Systems. RFC6020, 2010. https://tools.ietf.org/html/rfc6020; last accessed on 2016/06/19.

[8] Infonetics Research. 35 Percent of Operators Surveyed Will Deploy NFV This Year, 2015. http://www.infonetics.com/pr/2015/NFV-Strategies-Survey-Highlights.asp; last accessed on 2016/06/19.

[9] Intel. PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology. https://www-ssl.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html; last accessed on 2016/06/19.

[10] Intel. Virtual Machine Device Queues. https://www-ssl.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/io-acceleration-technology-vmdq.html; last accessed on 2016/06/19.

[11] Matthew Daniels, Michael Kalvaitis. Virtualized Infrastructure Leads to More Flexible Process Automation for Pharmaceutical Manufacturers, 2016. http://files.massbio.org/file/Virtualized-Infrastructure-Takes-Hold-at-Biogen.pdf; last accessed on 2016/06/19.

[12] OASIS. OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC, 2016. https://www.oasis-open.org/committees/tosca/; last accessed on 2016/06/19.

[13] ONF. https://www.opennetworking.org/about/onf-overview, 2016. https://www.opennetworking.org/about/onf-overview; last accessed on 2016/06/19.

[14] OPNFV. OPNFV Arno. https://www.opnfv.org/news-faq/press-release/2015/06/opnfv-delivers-open-source-software-enable-deployment-network; last accessed on 2016/07/20.

[15] OPNFV. OPNFV Wiki. https://wiki.opnfv.org/; last accessed on 2016/06/19.

[16] OPNFV. Doctor Wiki, 2016. https://wiki.opnfv.org/display/doctor/Doctor+Home; last accessed on 2016/06/19.

[17] OPNFV. MANO Wiki, 2016. https://wiki.opnfv.org/display/PROJ/OPNFV-OPEN-O; last accessed on 2016/06/19.

[18] OPNFV. OPNFV Delivers Second Release of Open Source Network Functions Virtualization Platform , 2016. https://www.opnfv.org/news-faq/press-release/2016/03/opnfv-delivers-second-release-open-source-network-functions; last accessed on 2016/06/19.

[19] OPNFV. Prediction Wiki, 2016. https://wiki.opnfv.org/display/prediction/Prediction; last accessed on 2016/06/19.

[20] Ray Le Maistre. Vodafone Demands More From NFV Vendors, 2016. http://www.lightreading.com/nfv/nfv-strategies/vodafone-demands-more-from-nfv-vendors/d/d-id/723690; last accessed on 2016/06/19.

[21] Siemens. CP 343-1 Lean. http://w3.siemens.com/mcms/industrial-communication/en/ie/system-interfacing/simatic-s7-sinumerik-o/s7-300/pages/cp343-1lean.aspx; last accessed on 2016/07/20.

[22] TM Forum. Catalyst Program. https://www.tmforum.org/collaboration/catalyst-program/; last accessed on 2016/06/19.

[23] TM Forum. ZOOM (Zero-Touch Orchestration Operations and Management). https://www.tmforum.org/zoom/; last accessed on 2016/06/19.

[24] Tony Poulos. Industrial manufacturing gears up for robots as a service, 2016. http://inform.tmforum.org/features-and-analysis/featured/2016/04/industrial-manufacturing-gears-up-for-robots-as-a-service/; last accessed on 2016/06/19.

Data Breaches in IT Systems

Magdalena Neumann
Betreuer: Heiko Niedermayer
Seminar: Innovative Internettechnologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: neumamag@in.tum.de

KURZFASSUNG

Datenpannen haben sich im letzten Jahrzehnt zunehmend zu einem schwerwiegenden Problem entwickelt. Obwohl der Diebstahl von Daten nicht immer nur aus finanziellen Gründen erfolgt, entstehen meist hohe Kosten für die Geschädigten. Diese Arbeit ordnet den Begriff Datenpanne ein, stellt aktuelle Zahlen vor und beschäftigt sich mit der Frage, wie sich der Trend in den letzten Jahren entwickelt hat. Dabei werden zwei verschiedene Datensammlungen genauer untersucht und Gemeinsamkeiten sowie vor allem Unterschiede vorgestellt. Insgesamt wird die Problematik der Vergleichbarkeit thematisiert. Zusätzlich wird der Umgang mit Datenpannen in der Realität anhand zweier bekannter Vorfälle gezeigt und daraus resultierende Herausforderungen, sowie die rechtliche Situation in verschiedenen Ländern und entstehende Kosten für Beteiligte analysiert.

Schlüsselworte

Datenpannen, LinkedIn Hack, Sony Pictures Entertainment Hack, Privacy Rights Clearinghouse, World's Biggest Data Breaches

1. EINLEITUNG

Seit 2005 werden Datenpannen von der Privacy Rights Clearinghouse Organisation in einer der größten Sammlungen an Datenlecks erfasst und analysiert. Bei der Betrachtung der Entwicklung der Datenmenge, die gehackt wurde bzw. anderweitig ungewollt abhanden kam, fällt auf, dass bis 2012 die Anzahl der Datenpannen gestiegen ist [8]. Allein im Jahr 2008 wurden so viele Datensätze gestohlen, wie in den vier vorhergehenden Jahren zusammen [30]. Dabei ist nicht jeder Verlust an Daten auf einen Hack zurückzuführen. Tatsächlich geht aus dem Datensatz der in [1] erfassten Datenpannen hervor, dass jeder zehnte Eintrag durch versehentliche Veröffentlichung entstand. Aus diesem Grund ist besonders für Unternehmen, die große Mengen an sensiblen Daten verwalten, interessant, welche Arten von Datenpannen auftreten können, welche Kosten entstehen, ob Trends in diesem Bereich feststellbar und welche Vorhersagen daraus ableitbar sind. Hat ein Unternehmen ein Datenleck entdeckt, so sind abhängig vom Standort außerdem gesetzliche Richtlinien bezüglich der Inkenntnissetzung der Betroffenen einzuhalten. In dieser Ausarbeitung soll deshalb zunächst ein Überblick rund um die Grundlagen von Datenpannen gegeben, sowie aktuelle Zahlen, Fakten und Trends veranschaulicht werden. Anschließend werden zwei konkrete Fallbeispiele im Detail betrachtet. In Kapitel 4 werden die verschiedenen Herausforderungen und Probleme insbesondere durch die entstandenen Kosten für Unternehmen, aber auch für die direkt Betroffenen diskutiert. Abschließend werden die Ergebnisse zusammengefasst und Erkenntnisse abgeleitet.

2. GRUNDLAGEN

Mit der steigenden Zahl an Datenpannen ist es aus analytischer Sicht von großer Bedeutung, sämtliche gemeldete Fälle sinnvoll zu erfassen und einzuordnen. Dabei gilt es in erster Linie zu überprüfen, ob es sich definitionsgemäß überhaupt um eine Datenpanne handelt und welcher Kategorie diese zuzuweisen ist. Hierfür werden in diesem Kapitel zunächst verschiedene Definitionen vorgestellt. Anschließend werden unterschiedliche Möglichkeiten zur Klassifizierung betrachtet. In einer kurzen Übersicht werden konkrete Zahlen und ein möglicherweise erkennbarer Trend veranschaulicht.

2.1 Definitionen

In der Literatur finden sich viele verschiedene Formulierungen, die den Begriff „Data Breach" beschreiben. In [28] wird z.B. folgende Definition festgelegt:

> „The term "data breach" means the loss, theft, or other unauthorized access, other than those incidental to the scope of employment, to data containing sensitive personal information, in electronic or printed form, that results in the potential compromise of the confidentiality or integrity of the data."

Diese Beschreibung bezieht einen (nicht unbedingt absichtlichen) Verlust explizit mit ein. Damit zählen beispielsweise auch verloren gegangene Datenträger als Datenpanne. Im Gegensatz hierzu wird in [9] folgende Definition verwendet:

> „A data breach is an incident where information is stolen or taken from a system without the knowledge or authorization of the system's owner."

Dabei beschränkt sich die Formulierung auf den aktiven Eingriff durch Stehlen oder anderweitiges Entwenden, sodass z.B. versehentlich abhanden gekommene Laptops nicht den Forderungen der Definition genügen. Für weitere Definitionen sei auf die Internationale Organisation für Normung (ISO)[16] und im weiteren Sinne auch auf das deutsche Bundesdatenschutzgesetz (BDSG)[6] verwiesen. Da in

doi: 10.2313/NET-2016-09-1_09

den meisten Sammlungen an Datenpannen durchaus zwischen absichtlichem Entwenden und versehentlichem Verlust unterschieden wird, wird in dieser Ausarbeitung die in [28] formulierte Begriffserklärung als Grundlage für die kommenden Kapitel verwendet.

2.2 Kategorien

Im Jahr 2013 wurde bekannt, dass Mitglieder der Social Media Plattform Facebook aufgrund eines Fehlers im System mit Hilfe eines Buttons ihre persönlichen Profilinformationen herunterladen konnten, die fälschlicherweise auch private Daten wie Telefonnummern und E-Mail Adressen von Freunden enthielten. Insgesamt waren rund 6 Millionen Datensätze betroffen[11]. Im selben Jahr wurden bis zu 2 Millionen Datensätze durch einen internen Mitarbeiter bei Vodafone entwendet[23]. Beide Fälle haben gemein, dass sie unter den Begriff „Datenpanne“ fallen. Sie unterscheiden sich jedoch in Art und zu Grunde liegendem Ablauf. Wie bereits in 2.1 angedeutet und auch in [26] zu finden, lassen sich diese in zwei große Kategorien einordnen: bösartig (malicious) und fahrlässig (negligent). Diese grobe Einteilung lässt sich wiederum, wie in Tabelle 1 aufgelistet, angelehnt an die in [1] verwendete Untergliederung weiter verfeinern. Während die-

Tabelle 1: Verfeinerung der Kategorien

Bösartig
Hacks
Insider
Diebstahl
Fahrlässig
Versehentlich veröffentlicht
Konfigurationsfehler
Verlust
Schwache Sicherheit

se Unterteilung den Fokus auf die Intention der am Datenleck Beteiligten legt, wird in [30] nach Art der Herkunft kategorisiert. Dabei zählen Angriffe von Hackern, kriminellen Organisationen und auch höhere Gewalt wie das Wetter zur Kategorie *Extern*, Datenpannen von Mitarbeitern innerhalb eines Unternehmens wie z.B. Administratoren mit hohen Zugriffsrechten zu *Intern* und von Partnerunternehmen ausgehende Attacken zu *Partner*. Eine weitere Möglichkeit besteht darin in *Softwarebasiert* (Hacks und Malware), *Hardwarebasiert* (Diebstahl und Verlust) und *menschliche Unachtsamkeit* (Fehler und versehentliche Weiterleitung)[27] zu unterteilen. Generell kann eine Datenpanne auch eine Verkettung mehrerer Punkte aus verschiedenen Kategorien umfassen. Dies wird beim Betrachten des auf das LinkedIn Profil des Facebookinhabers Mark Zuckerberg ausgeführten Hacks in Abschnitt 3.2 deutlich.

Jede der genannten Methoden zur Einteilung der Arten von Data Breaches hat Vor- und Nachteile. Für die schnelle und einfache Erfassung aller gemeldeten Fälle ist eine möglichst grobe Kategorisierung besser geeignet, als eine eher detaillierte Aufschlüsselung, insbesondere dann, wenn Begriffe wie Hacks noch weiter in einzelne Unterscheidungen wie Malware, Brute Force und SQL Injection zerlegt werden. Jedoch könnte eine nur sehr grobe Einteilung spätere Analysen erschweren, wenn beispielsweise gezielt Informationen über die Anzahl an Brute Force Angriffen gewünscht sind oder möglichst genaue Prognosen für die Zukunft abgeleitet werden sollen.

2.3 Datentypen

Um entscheiden zu können, wie schwerwiegend eine Datenpanne ist, muss bei der Erfassung der Daten anhand ihres Datentyps unterschieden werden. In der von [1] erstellten Sammlung wird hierbei jeder Datensatz mit Hilfe eines Sensibilitätswertes gekennzeichnet. E-Mail Adressen und Online Informationen wie beispielsweise Benutzernamen (ohne Passwort) werden als eher harmlos eingestuft (Wert 1), während Kreditkartennummern (Wert 3000) und E-Mail Passwörter (Wert 4000) als besonders sensible Daten gesehen werden. Laut dem California Data Breach Record 2016 sind Sozialnummern mit einer Gesamtheit von 24 Millionen Datensätzen die am häufigsten durch Datenpannen betroffene persönliche Information [27]. 39% der betrachteten Fälle enthalten Kreditkarteninformationen, 19% medizinische Daten [27].

2.4 Allgemeiner Ablauf

Je nach Art der Datenpanne lässt sich ein Muster bezüglich des Ablaufs erkennen. Bei Betrachtung der Angriffe der Kategorie *Bösartig* aus Tabelle 1 lassen sich diese mit drei aufeinanderfolgenden Schritten beschreiben [9]. Generell unterscheidet sich die Länge der einzelnen Schritte stark [30].

Schritt 1: Recherche
Der Angreifer sucht nach Möglichkeiten in das Zielsystem einzudringen. Die dafür eingesetzte Vorbereitungszeit bewegt sich hauptsächlich zwischen mehreren Stunden bis hin zu Monaten [30].

Schritt 2: Angriff
Für gefundene Schwachstellen im Bereich der Infrastruktur werden gezielt Angriffe in Form von beispielsweise SQL Injection oder Session Hijacking eingesetzt. Weitere Varianten sind Täuschungsversuche mit Hilfe von Phishing oder Spam. Für detaillierte Erklärungen dieser Angriffe sei auf [25] verwiesen. In 77% der Fälle erfolgt der Zugriff dabei innerhalb von wenigen Tagen, knapp ein drittel erfordert nicht einmal eine Stunde [30].

Schritt 3: Datentransfer
Sobald der Zugriff in Schritt zwei erlangt wurde, erfolgt der eigentliche Diebstahl der Daten, indem der Angreifer die Daten aus dem Zielsystem heraus transferiert. Die Zeitspanne hierfür erstreckt sich oft bis zu dem Zeitpunkt, an dem der Angegriffene die Attacke bemerkt und unterbindet. Bei knapp 50% der Datenpannen läuft diese Phase sogar über mehre Monate, wobei zwischen dem Entdecken des Problems und Schließen der Lücke in 42% der Fälle nochmals Wochen vergehen [30].

Kommen Daten direkt durch intern arbeitendes Personal abhanden, so erfolgen die Schritte eins und zwei implizit, da meist der Zugriff bereits besteht. Diese Art eines Angriffs ist demnach auch für Unternehmen oftmals am Schwierigsten zu verhindern. Für den Ablauf von Datenpannen der

Kategorie *Fahrlässig* ist jedoch oft kein eindeutiges Muster erkennbar. Generell gilt, dass für solch einen Vorgang zunächst eine Unachtsamkeit benötigt wird, die anschließend von einer kriminellen Instanz ausgenutzt wird. Zur Vermeidung der im ersten Schritt erwähnten Fahrlässigkeit sind allerdings Tipps wie ausführliches Testen, ständige Wartung und regelmäßige Aktualisierungen der Sicherheitsstandards der Systeme hilfreich. Weitere Vorschläge finden sich auch in [9]. Insbesondere ist jedoch das Wählen sicherer und einzigartiger Passwörter, die sich für sämtliche Zugänge unterscheiden sollten, ein wichtiger Aspekt, der auch in 3.2 eine große Rolle spielt.

2.5 Aktuelle Zahlen und Trends

Nun stellt sich auch für viele Unternehmen die Frage, ob die Häufigkeit und Größe von Datenpannen tatsächlich immer weiter steigt und wie „schlimm“ die Entwicklung wirklich ist. In diesem Kapitel wird einerseits die Datengrundlage der Analyse vorgestellt und aus dieser anschließend Trends abgeleitet.

2.5.1 Datensammlungen im Detail

Im Folgenden werden zwei Datensammlungen genauer beschrieben, die bereits in Kapitel 1 erwähnt wurden und besonders für die weitere Betrachtung in den Abschnitten 2.5.2 und 2.5.3 verwendet werden. Beide Datensammlungen haben gemein, dass die zugrunde liegenden Datensätze frei zugänglich und online zur Verfügung stehen und damit besser analysiert und verglichen werden können. Im Gegensatz dazu wird in [30] auf eine interne Datensammlung verwiesen, die nicht für eigene Untersuchungen zugänglich ist.

Privacy Rights Clearinghouse Organisation
Im Jahr 1992 gründete Beth Givens die kalifornische non-profit Organisation Privacy Rights Clearinghouse (PRC) mit dem Ziel Konsumenten über Datenschutz zu informieren und deren Privatsphäre zu verteidigen [8]. Seit 2005 wird eine über deren Webseite öffentlich zugängliche Sammlung von Datenpannen dokumentiert und gepflegt. Insgesamt umfasst die Datenbank 4860 Einträge mit knapp 90 Millionen betroffenen Datensätzen [8]. Abbildung 1 zeigt die Anteile der verschiedenen Kategorien an der Gesamtheit der Datenpannen auf Grundlage der PRC Datensammlung. Dabei wird deutlich, dass Hacks und Malware mit Abstand die häufigste Ursache von Datendiebstählen sind [8].

World’s Biggest Data Breaches (WBDB)
Der Autor, Journalist und Designer David McCandless hat sich auf die Visualisierung von großen Datenmengen spezialisiert und verschiedene Projekte mit Zugriff auf die zugrunde liegende Datenmenge online zur Verfügung gestellt. Dabei werden insbesondere Datenpannen mit über 30.000 betroffenen Datensätzen erfasst. Obwohl lediglich 218 Einträge vorliegen, ergibt sich eine Gesamtmenge von über 2,6 Milliarden betroffenen Datensätzen. Die Sammlung beschreibt daher Pannen, die als besonders schwerwiegend erachtet werden können, da eine große Menge an Daten pro Eintrag betroffen war [1].

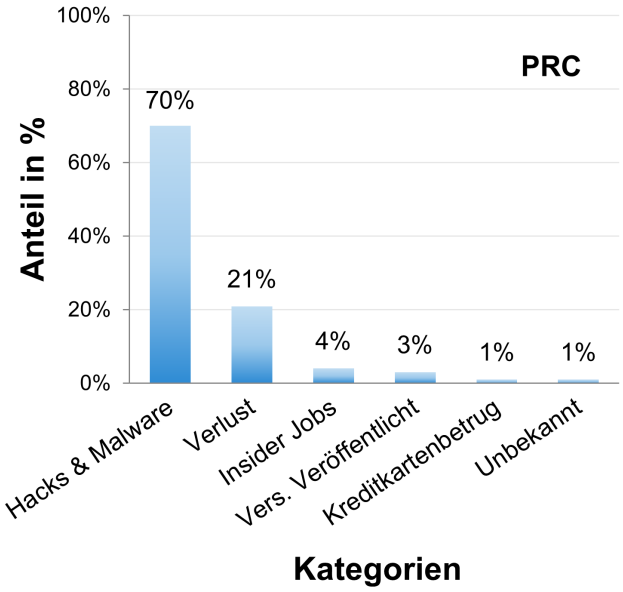


Abbildung 1: Anteil der einzelnen Kategorien an der Gesamtanzahl an Datenpannen der PRC Datensammlung [8]

2.5.2 Vergleich der Datenbanken

Im direkten Vergleich erscheinen die genannten Mengen der gestohlenen Daten zunächst widersprüchlich. Beim genaueren Untersuchen fällt jedoch auf, dass die PRC Organisation nur dann Werteangaben über die Höhe der betroffenen Datensätze einpflegt, sofern diese tatsächlich bestätigt wurden und es sich bei dem Verlust um sensible Daten handelt [8]. Für den in Abschnitt 3.1 betrachteten Fall gibt es beispielsweise keine offiziell bestätigten Quellen mit konkreten Mengenangaben. Die Angreifer jedoch behaupten über 100 Terabyte an Daten entwendet zu haben [4]. In der Verteilung der Anzahl der gestohlenen Datensätze pro eingetragener Datenpanne lassen sich ebenfalls Unterschiede erkennen, wie Abbildung 2 veranschaulicht. Im oberen Diagramm ist deutlich erkennbar, dass nur eine geringe Anzahl an Datenpannen eine wirklich große Menge an Datensätzen enthält. Bei Betrachtung der Datenbank von WBDB im unteren Diagramm fällt hingegen auf, dass viele der Datenpannen zum Verlust großer Datenmengen führten. Dies lässt sich durch die verschiedenen zugrunde liegenden Modelle beider Sammlungen erklären, da, wie im Namen des World’s Biggest Data Breaches Datensatzes enthalten, dort nur die größten Fälle erfasst und auch nicht eindeutig bestätigte Vorfälle aufgenommen werden.

2.5.3 Trends

In [26] wurde die Entwicklung der Häufigkeit und Größe an Datenpannen mit Hilfe der Daten des PRC untersucht, um passende Verteilungsfunktionen zu finden, die den Verlauf in den letzten Jahren möglichst genau beschreiben. Damit wurden auch Ansätze für verschiedene Vorhersagen vorgestellt. Die dadurch erlangten Erkenntnisse sind allerdings stark vom gegebenen Datensatz abhängig und könnten aufgrund der in 2.5.2 genannten Unterschiede völlig anders für die von WBDB zur Verfügung gestellten Datensammlung

doi: 10.2313/NET-2016-09-1_09

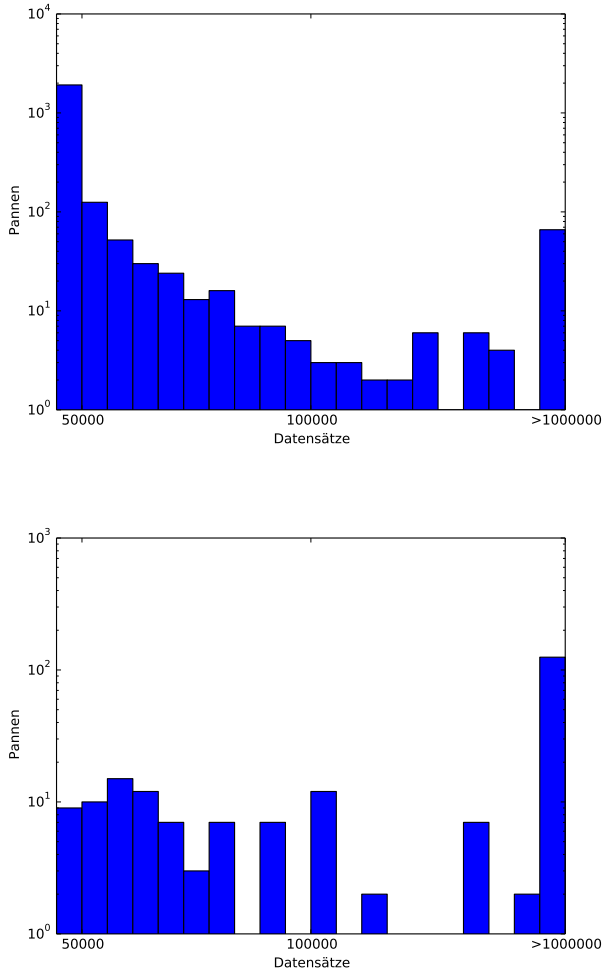


Abbildung 2: Verteilung der betroffenen Datensätze pro eingetragene Datenpanne. Zugrundeliegende Daten des PRC (oben) [8] und Datenbank von WBDB (unten) [1].

aussehen. Insofern lässt sich schwer festlegen, ob die Vorhersagen auch wirklich der Realität entsprechen. Die Betrachtung eines zweiten Datensatzes könnte die Genauigkeit jedoch verbessern. Zusätzlich werden Aussagen über die Entwicklung von Datenpannen auf Basis der PRC Datensammlung abgeleitet. Dabei wird insbesondere in der Menge an *bösartigen* Angriffen ein Rückgang und zeitgleich konstante Häufigkeit über einen Zeitraum von zehn Jahren ermittelt [26], während in anderen Quellen auf einen ansteigenden Trend der Menge an Datenpannen hingedeutet wird [15]. Oft trifft dies nur auf bestimmte Bereiche wie beispielsweise den Gesundheitssektor zu, während in anderen Gebieten die Zahl wiederum rückläufig ist. So zeigt auch der aktuelle California Data Breach Bericht von 2016, dass von 2012 bis 2015 die Menge an durch Hacks verursachten Datenpannen von 45% auf 58% stieg, jedoch gleichzeitig die Anzahl der auf Diebstahl oder Verlust zurückzuführenden Data Breaches um 10% sank. Abbildung 3 zeigt die Anzahl an Datenpannen über die letzten zwölf Jahre. Auch wenn es den Eindruck erweckt, als gäbe es ab 2012 einen eher gleichbleibenden bis sogar rückläufigen Trend bei der Menge an erfassten Datenpannen, sind für eine derartige Aussage zu wenig Datenpunkte vorhanden, um wirklich von einem „Trend“sprechen zu können. Insbesondere können durch die in Abschnitt 2.4 aufgezeigten Zeiten über die Länge der Dauer bis eine Lücke überhaupt erst entdeckt wird noch nachträgliche Eintragungen in der Datensammlung erfolgen, die bereits vergangenen Jahren zuzuordnen sind und die Entwicklungskurve aus Abbildung 3 nochmals verändern könnten. Andererseits wird beispielsweise in [2] erwähnt, dass in Deutschland mit Einführung der Informationspflicht Unternehmen einen erhöhten Fokus auf Sicherheitsaspekte legen, um dem steigenden Druck durch die mediale Aufmerksamkeit für Sicherheitslücken entgegen zu wirken. Derartige Gesetzesänderungen können somit Einfluss auf die Anzahl der erfassten Datenpannen nehmen.

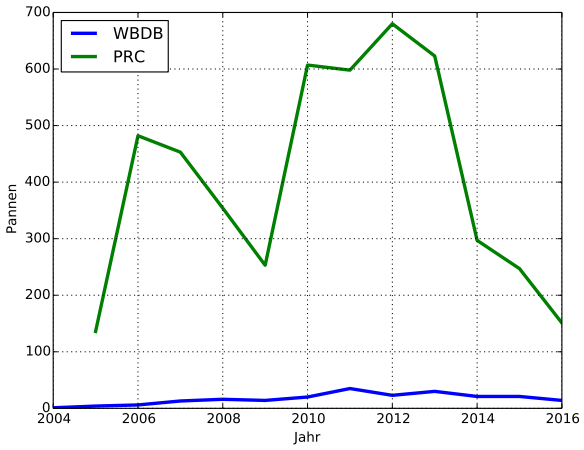


Abbildung 3: Entwicklung der Anzahl an Datenpannen von 2004 bis 2016 basierend auf den Datensätzen PRC [8] und WBDB [1].

3. FALLBEISPIELE

Nachdem in Kapitel 2 die Grundlagen und Begrifflichkeiten betrachtet wurden, sollen nun einige Fallbeispiele im Detail beleuchtet werden. Dabei wird auch deutlich, dass die Gründe und Folgen in manchen Fällen über typische Datenverluste hinaus gehen und sogar politische Bedeutung finden.

3.1 Sony Pictures Entertainment Hack

Der Sony Pictures Entertainment Hack erhielt im Jahr 2014 große mediale Aufmerksamkeit. Dabei sorgten nicht nur die Menge an gestohlenen Daten für große Diskussionen, sondern auch die Herkunft der mutmaßlichen Drahtzieher und deren angeblich politische Motivation. Dieser Vorfall veranschaulicht daher insbesondere, welche Konsequenzen Datenpannen auf internationaler Ebene nach sich ziehen können.

3.1.1 Beschreibung

Die zeitliche Entwicklung des Skandals um den Sony Hack im November 2014 erstreckte sich über eine Zeitspanne von etwa einem Monat. Eine genaue Auflistung der Ergebnisse

doi: 10.2313/NET-2016-09-1_09

in chronologischer Reihenfolge wurde in [22] erarbeitet. Obwohl bekannt ist, dass die Datenpanne auf eine Sicherheitslücke zurückzuführen ist, über die Malware in das System von Sony Entertainment Pictures eingebracht wurde, konnte bis heute nicht abschließend geklärt werden, wie viele Daten genau gestohlen wurden. Die Berichterstattung begann damit, dass Mitarbeiter der *Sony Pictures Entertainment Inc.* aufgrund einer Sperre ihrer Arbeitsrechner durch ein Bild mit dem Text „Hacked By #GOP“ an der Aufnahme ihrer Arbeit gehindert wurden. Die zuständige IT-Abteilung stellte daraufhin sogar fest, dass alle der Infrastruktur angehörenden Rechner, gesperrt waren und ein Teil der im Unternehmensnetzwerk befindlichen Daten zerstört wurde. Es folgte eine Kontaktaufnahme der Gruppe *Guardians of Peace (#GOP)*, welche angab, die genannte Sperre installiert zu haben, und außerdem behauptete, mehrere 100 Terabyte an Daten aus dem Firmennetzwerk entwendet zu haben. Als Beweis wurden durch GOP fünf von Sony Pictures Entertainment produzierte Filme online verbreitet, die noch nicht veröffentlicht und daher nur unternehmensintern zugänglich waren. Die kurze Zeit später gestellten Forderungen der Gruppe richteten sich einerseits an Mitarbeiter des Konzerns, die sich öffentlich von Sony Pictures Entertainment distanzieren sollten, andererseits an Sony selbst, mit dem Verlangen, die Veröffentlichung des Films *The Interview* zu unterlassen. Zur Erhöhung des Drucks wurden laufend weitere interne Daten des Konzerns durch die Hackergruppe veröffentlicht. Der genannte Film stellt dabei ein kritisches Werk gegenüber dem nordkoreanischen Staatsoberhaupt Kim Jong-un dar und wurde in Nordkorea derart negativ beurteilt, dass dort die Veröffentlichung des Bildmaterials sogar als „Kriegshandlung“ und „unverhohlene Unterstützung von Terrorismus“ eingestuft wurde [19]. Es erhärtete sich somit der Verdacht, dass die Regierung Nordkoreas bei dem Angriff auf Sony Pictures Entertainment involviert war. Dies führte sogar dazu, dass das FBI nach einer Untersuchung des Falls offiziell bestätigte, dass die Regierung Nordkoreas den Angriff durchgeführt hätte, was von dieser jedoch sofort dementiert wurde. Bis heute konnte eine Beteiligung Nordkoreas nicht zweifelsfrei nachgewiesen werden. Aufgrund des enormen Drucks, der durch die Gruppe #GOP ausgeübt wurde, entschieden sich die Verantwortlichen von Sony zwischenzeitlich gegen eine Veröffentlichung des Films, annullierten diese Entscheidung jedoch kurze Zeit später und willigten einer Ausstrahlung des Werkes in Kinos ein.

3.1.2 Vorgehensweise

Zur Vorgehensweise der Angreifer wurden nie Informationen explizit im Bezug zum Sony Pictures Entertainment Hack veröffentlicht. Jedoch veröffentlichte das *US-CERT*, eine US-Organisation des Department of Homeland Security, eine Stellungnahme, in der beschrieben wird, dass eine Malware namens *SMB Worm Tool* verwendet wurde, um Attacken auf ein „großes Unterhaltungsunternehmen“ auszuführen. Die Vermutung liegt nahe, dass es sich dabei um den Sony Pictures Entertainment Hack handelt, da die beschriebene Schadsoftware sämtliche Funktionalität aufweist, um den auf die Infrastruktur des Unternehmens ausgeführten Angriff durchzuführen. Eine genaue Beschreibung des Wurms wurde in [14] veröffentlicht.

Die Software wurde auf unbekannte Weise in die Infrastruktur des Konzerns eingebracht und besitzt die Eigenschaft, sich über das *Server Message Block (SMB)* Protokoll im gesamten Netzwerk zu verbreiten. Sie besteht aus mehreren Modulen, die einerseits Funktionalität zur Durchführung von Brute Force Angriffen auf SMB-Freigaben zur Verfügung stellen, um eine Verbreitung im Netzwerk zu ermöglichen. Andererseits werden Funktionen zur Verfügung gestellt, um mit einer *command and control (C2)* Infrastruktur zu kommunizieren, die einen Informationsaustausch und die Steuerung der Malware durch die Hacker von außerhalb ermöglicht. Zusätzlich installierte die Software Module, die Daten im infizierten Netzwerk unwiederbringlich löschen konnten bzw. die gesamte Festplatte eines Hostsystems durch die Zerstörung des Master Boot Records (MBR) ermöglichte [5].

Der gesamte Ablauf des Angriffs erfolgte also in drei Schritten. Zuerst wurde die Malware auf ungeklärte Weise in das Netzwerk der Firma Sony geschleust. Es wird vermutet, dass dafür ein Phishing Angriff durchgeführt wurde [21]. Daraufhin wurde die Schadsoftware über SMB-Freigaben im Netzwerk verteilt und Daten abgegriffen. Nach Angaben der Gruppe #GOP dauerte die Ausführung dieses Schritts mindestens ein Jahr [22]. Schließlich wurden Daten im Netzwerk gelöscht und die infizierten Rechner unbrauchbar gemacht.

3.1.3 Folgen

Vor allem aufgrund des hohen politischen Interesses am Sony Pictures Entertainment Hack hebt sich der Fall klar von anderen Datenpannen ab. Der Diebstahl von Daten und die Verwendung dieser für Erpressungsversuche führte zwischenzeitlich sogar dazu, dass die Regierung der Vereinigten Staaten von Amerika eine Verfügung des Präsidenten veröffentlichte, die zusätzliche Sanktionen gegen Nordkorea ankündigte und dabei explizit Bezug auf die Attacke gegen Sony nahm [10].

3.2 LinkedIn und Mark Zuckerberg

Anfang Juni 2016 erregte ein Hackerangriff auf die Accounts verschiedener prominenter Persönlichkeiten auf sozialen Plattformen Aufsehen. Neben Berühmtheiten wie Rolling-Stones-Musiker Keith Richards und Reality-TV-Star Kylie Jenner war auch Mark Zuckerberg, der Gründer und Vorstandsvorsitzende der größten Social Media Plattform *Facebook*, betroffen [20]. Der Fall brachte eine besondere Brisanz mit sich, da es der breiten Masse möglich war, den Verlauf des Angriffs über die betroffenen sozialen Medien zeitnah zu verfolgen.

3.2.1 Beschreibung

Obwohl bei den Vorfällen im Jahr 2016 der Hergang zuerst nicht klar war, stellte sich später heraus, dass die zugrundeliegende Ursache schon auf das Jahr 2012 zurück ging. Damals wurde seitens LinkedIn, einem sozialen Netzwerk, das überwiegend auf berufliche Kontakte ausgelegt ist, bekannt gegeben, dass ein Einbruch in die Datenbank stattgefunden hatte und dabei die Passwörter „einiger Mitglieder“ entwendet wurden [17]. Vier Jahre später wurde klar, dass es sich bei „einigen“ Mitgliedern um etwa 6 Millionen Nutzer handelte, deren Zugangsdaten gestohlen wurden. Diese wurden verwendet, um gefälschte Informationen über die Accounts öffentlicher Persönlichkeiten zu verbreiten.

3.2.2 Vorgehensweise

Bis heute ist nicht abschließend geklärt wie sich Hacker Zugriff auf die Nutzerdatenbank bei LinkedIn verschaffen konnten. Jedoch wurden Informationen veröffentlicht, die belegen, dass die Nutzerpasswörter als ungesalzener SHA1-Hash vorlagen [13]. Grundsätzlich ist die Vorgehensweise richtig, Passwörter nicht im Klartext zu speichern, sondern durch eine Einwegfunktion (Hash) unkenntlich zu machen. Bei der Umsetzung wurde jedoch in diesem Fall auf einen Salt-Wert verzichtet, einer zufälligen Zeichenkette, die an jede Eingabe der Hashfunktion angehängt wird, um die Verwendung von vorberechneten Tabellen zur Wiederherstellung des Klartextes aus dem berechneten Hashwert zu erschweren [29]. Aus den mutmaßlich 6 Millionen entwendeten Passworthashes konnten daher innerhalb weniger Tage die Passwörter im Klartext ermittelt werden. Auf der Plattform LinkedIn selbst wurden zwar alle Passwörter zeitnah zurückgesetzt, doch wurde die Gefahr unterschätzt, dass viele Nutzer ein und das selbe Passwort auf vielen Social Media Plattformen verwenden. Es war den Angreifern damit möglich, sich mit den von LinkedIn entwendeten Passwörtern Zugriff auf Accounts in anderen Portalen zu verschaffen. Auch Mark Zuckerberg verwendete auf LinkedIn, Twitter und dem Photodienst Pinterest das selbe Passwort, das unbestätigten Berichten zufolge im Klartext „dadada“ lautete [18]. Die Hacker hatten damit die Möglichkeit, Falschinformationen über die gehackten Accounts zu verbreiten. Außerdem wurde Zuckerbergs Pinterest Auftritt derart geändert, dass die Profilbeschreibung den Text „gehackt vom OurMine Team“ enthielt [12].

3.2.3 Folgen

Aufgrund der Tatsache, dass die Identität der Gruppe, die unter dem Namen *OurMine Team* operiert, ungeklärt ist, sind in Sachen Strafverfolgung keine Ergebnisse bekannt. Die betroffenen sozialen Netzwerke reagierten mit einer Sperrung der Accounts, die mit der Hackgergruppe in Verbindung gebracht wurden. Ein Aufruf des Twitter-Profils *_OurMine_* wird automatisch auf die Website `https://twitter.com/account/suspended` umgeleitet, die stets bei dem Versuch aufgerufen wird, das Profil eines bei Twitter gesperrten Accounts zu besuchen.

4. HERAUSFORDERUNGEN

Wie in Abschnitt 2.4 bereits erwähnt, kann sich der Zeitraum vom Beginn eines Angriffs bis hin zur Entdeckung und Eindämmung über mehrere Monate, oder sogar Jahre erstrecken, was für alle Beteiligten besondere Herausforderungen birgt. Im Falle einer Datenpanne ist nicht nur wichtig diese möglichst schnell zu entdecken und weitere Zugriffe zu unterbinden, sondern auch Betroffene zeitnah zu informieren und entstandene Schäden zu begrenzen. Dies gelingt beteiligten Unternehmen nicht immer, was Folgeangriffe, wie im Fallbeispiel in Abschnitt 3.2 gezeigt, erst möglich macht und auch Jahre nach dem Angriff noch zu Konsequenzen führen kann. Derartige Skandale können außerdem hohe finanzielle Einbußen nach sich ziehen. Im Folgenden werden daher die Verantwortlichkeiten der Firmen erläutert und anschließend näher auf Art und Höhe der entstehenden Kosten eingegangen.

4.1 Rechtliche Meldepflicht

Da kein international geltendes Recht für Datenpannen existiert, hängt der korrekte Umgang mit diesen von der Rechtsprechung des jeweiligen Standortes ab. Dies erschwert die Beurteilung der Fälle zusätzlich, da keine einheitliche Rechtsauffassung zugrunde gelegt werden kann. In den Vereinigten Staaten von Amerika variieren die rechtlichen Vorgaben sogar von Staat zu Staat, wobei 47 der 51 Staaten eine Informationspflicht bei Verlust von persönlichen Daten gegenüber Betroffenen vorschreiben. 15 Staaten haben dabei Regularien bezüglich der Form der Informationsmitteilung festgelegt, wie in [24] in tabellarischer Form dargestellt. In Deutschland sind die Rechte und Pflichten im Bundesdatenschutzgesetz geregelt. Dort ist eine Informationspflicht bei Auftreten von Datenpannen festgelegt. Seit der Aufnahme der neuen Vorschrift in §42a stieg das Interesse der Firmen, sich stärker im Bereich IT-Sicherheit zu positionieren[2]. Dabei ist auch auf die Form der Meldung zu achten, die vorgibt, dass ein Unternehmen eine Datenpanne in „mindestens eine halbe Seite umfassen, [und] in mindestens zwei bundesweit erscheinenden Tageszeitungen“ [6] publiziert werden muss.

4.2 Kosten

Wird gegen die im vorhergehenden Abschnitt 4.1 genannten Pflichten verstoßen, so drohen hohe Bußgelder bis zu 300.000€[7]. Insbesondere leidet das für viele Firmen wichtige Image unter mangelhafter Aufklärung, was Kosten in nicht absehbarer Höhe verursachen kann. Neben einer genaueren Analyse der Kostenentwicklung werden in [3] die finanziellen Aufwendungen in vier Kategorien eingeteilt. Dabei wird zwischen *Erkennung und Eskalierung* (Ermittlung der Faktenlage und interne Kommunikation), *Benachrichtigung* (Publikation der Panne), *ex-post Kosten* (Nachbereitung und Schließung der Lücken) und *Businessverlust* (Abgang von Kunden und Geschäftspartnern) unterschieden. Statistisch gesehen stiegen die finanziellen Aufwendungen jedes Jahr stetig an, während Businessverlust und ex-post Kosten den größten Teil ausmachen. Auch der durchschnittliche finanzielle Gesamtschaden pro Datenpanne stieg über die Jahre bis auf 3.79 Millionen US-Dollar (2015) [3].

5. ZUSAMMENFASSUNG

In dieser Arbeit wurden die Grundlagen zu Datenpannen vorgestellt und gängige Definitionen als Basis für weitere Analysen dargestellt. Dabei wurde auch auf die unterschiedlichen Interpretationen des Begriffs bei der Klassifikation bekannter Fälle eingegangen und zwei Datensammlungen analysiert, die auf verschiedene Weise die Pannen erfassen. Es zeigte sich, dass die resultierenden Datenbanken in hohem Maße abhängig von der Einordnung bezüglich unbestätigter Informationen und Vollständigkeit der zugrundeliegenden Datensätze sind. Mit dieser Erkenntnis wurde ein Trend herausgestellt, der einen Anstieg der Häufigkeit von Datenpannen bis 2012 beschreibt. Für die Bestimmung eines Trends der Folgejahre liegen jedoch genügend Daten vor. Es folgte die Aufarbeitung zweier brisanter Beispiele, die zeigten, dass die Entstehung von Pannen sowohl durch eine mangelhafte Sicherheitspolitik bei Unternehmen, als auch durch geringes Sicherheitsbewusstsein beim Kunden selbst massiv begünstigt wird. Abschließend wurden die für Firmen auftretenden Herausforderungen benannt und die geltende Gesetzgebung sowie die entstehenden Kosten untersucht. Es zeigte sich, dass die Einführung von Gesetzen zur Verbesserung

doi: 10.2313/NET-2016-09-1_09

der Unternehmenstransparenz zu einem erhöhten Fokus auf IT-Sicherheit in Firmen führt.

Durch die starken Unterschiede im Design von Datensammlungen lassen sich diese nur bedingt vergleichen. Eine einheitliche und international greifende Regelung zur Einordnung von Datensätzen, sowie global geltende Gesetze zum Thema Meldepflicht wären daher wünschenswert. Interessant wäre zudem eine Anwendung und Analyse des in [26] vorgestellten Modells auf verschiedenen Datensammlungen, um dessen universelle Einsetzbarkeit zu beurteilen.

6. LITERATUR

[1] http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/. 2016-05-20.

[2] 2010 Annual Study: German Cost of Data Breach. http://www.symantec.com/content/de/de/about/downloads/press/2010_annual_study.pdf. 2016-06-19.

[3] 2015 Cost of Data Breach Study: Global Analysis. https://nhlearningsolutions.com/Portals/0/Documents/2015-Cost-of-Data-Breach-Study.PDF. 2016-06-19.

[4] A Look Through The Sony Pictures Data Hack: This Is As Bad As It Gets. https://www.buzzfeed.com/tomgara/sony-hack?utm_term=.rfBKoqaQv#.dukOe4VvQ. 2016-06-18.

[5] Alert (TA14-353A) Targeted Destructive Malware. https://www.us-cert.gov/ncas/alerts/TA14-353A. 2016-06-17.

[6] Bundesdatenschutzgesetz (BDSG) § 42a Informationspflicht bei unrechtmäßiger Kenntniserlangung von Daten. https://www.gesetze-im-internet.de/bdsg_1990/__42a.html. 2016-06-16.

[7] Bundesdatenschutzgesetz (BDSG) § 43a Informationspflicht bei unrechtmäßiger Kenntniserlangung von Daten. https://www.gesetze-im-internet.de/bdsg_1990/__43.html. 2016-06-19.

[8] Chronology of data breaches security breaches 2005 - present. http://www.privacyrights.org/data-breach. 2016-05-20.

[9] Definition - data breach. http://www.trendmicro.com/vinfo/us/security/definition/data-breach. 2016-05-20.

[10] Executive Order – Imposing Additional Sanctions with Respect to North Korea. https://www.whitehouse.gov/the-press-office/2015/01/02/executive-order-imposing-additional-sanctions-respect-north-korea. 2016-06-17.

[11] Facebook admits year-long data breach exposed 6 million users. http://www.reuters.com/article/net-us-facebook-security-idUSBRE95K18Y20130621. 2016-06-16.

[12] Hacker legen Mark Zuckerberg rein. http://www.rp-online.de/digitales/internet/cyper-attacke-hacker-nehmen-facebook-chef-mark-zuckerberg-ins-visier-aid-1.6027071. 2016-06-17.

[13] Hacker puts up 167 Million LinkedIn Passwords for Sale. http://thehackernews.com/2016/05/linkedin-account-hack.html. 2016-06-17.

[14] Hackers Used Sophisticated SMB Worm Tool to Attack Sony. http://www.securityweek.com/hackers-used-sophisticated-smb-worm-tool-attack-sony. 2016-06-17.

[15] Internet Security Threat Report 2014. http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf. 2016-06-19.

[16] ISO/IEC 27040. https://www.iso.org/obp/ui/#iso:std:iso-iec:27040:ed-1:v1:en. 2016-06-16.

[17] LinkedIn resets passwords on millions of accounts as new data-leak reports surface. http://venturebeat.com/2016/05/18/linkedin-resets-passwords-on-millions-of-accounts-as-new-data-leak-reports-surface/. 2016-06-17.

[18] Mark Zuckerberg's password was 'dadada'. What hope do the rest of us have? http://www.telegraph.co.uk/technology/2016/06/06/mark-zuckerbergs-password-was-dadada-what-hope-do-the-rest-of-us/. 2016-06-17.

[19] Nordkorea protestiert gegen Kinofilm über Kim Jong-un. http://www.faz.net/aktuell/politik/ausland/brief-an-un-generalsekretaer-nordkorea-protestiert-gegen-kinofilm-ueber-kim-jong-un-13038080.html. 2016-06-17.

[20] Passwortsünder Zuckerberg? http://www.golem.de/news/social-media-passwortsuender-zuckerberg-1606-121326.html. 2016-06-17.

[21] Sony Got Hacked Hard: What We Know and Don't Know So Far. https://www.wired.com/2014/12/sony-hack-what-we-know/. 2016-06-18.

[22] Sony Hack: A Timeline. http://deadline.com/2014/12/sony-hack-timeline-any-pascal-the-interview-north-korea-1201325501/. 2016-06-16.

[23] Vodafone Deutschland Ziel eines Angriffs. http://www.vodafone.de/privat/hilfe-support/kundeninformation.html. 2016-06-16.

[24] F. Bisogni. Data breaches and the dilemmas in notifying customers. In *The 14th Annual Workshop on the Economics of Information Security*, pages 23–25, 2015.

[25] W. A. S. Consortium et al. Threat classification. *Online at: http://www. webappsec. org/projects/threat/v1/WASC-TC-v1 0. pdf*, 2004.

[26] B. Edwards, S. Hofmeyr, and S. Forrest. Hype and heavy tails: A closer look at data breaches. 2015.

[27] K. D. Harris and A. General. California data breach report. 2016.

[28] O. of the Federal Register. *Code of Federal Regulations, Title 38, Pensions, Bonuses, and Veterans' Relief, Pt. 18-End, Revised as of July 1 2009*. U.S. Government Printing Office, 2009.

[29] D. Todorov. *Mechanics of User Identification and*

Authentication: Fundamentals of Identity Management. CRC Press, 2007.

[30] R. Verizon Business. Team, 2009 data breach investigations report, 2009.

doi: 10.2313/NET-2016-09-1_09

Garbled Circuits

Frederic Naumann
Betreuer: Marcel von Maltitz
Seminar Innovative Internet-Technologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: naumann@in.tum.de

ABSTRACT

In 1982, Andrew Yao published a paper which described possible ways of handling Secure Multi-Party Computation, but only in a very theoretical manner. In the following years, Yao developed a conceptual implementation approach to this subject which he titled as "Garbled Circuits", although he never actually published any of his work on Garbled Circuits but only mentioned and explained the idea behind the algorithm in several talks. At the time this concept was presented, it was deemed more of a theoretical concept than an actual implementation due to the limited computation power. But over the years, the computational possibilities grew and actual implementations became feasible.
This paper is set out to explain the function of Yao's original algorithm in detail and also evaluate it under various aspects such as performance and resistance to certain attacks. We will also talk about improvements to Yao's algorithm that have been proposed during the last nearly thirty years, and finally get into some actual implementation of these algorithms.

Keywords

Secure Multi-Party Computation, Secure Function Evaluation, Garbled Circuit Protocol

1. INTRODUCTION

The general *Secure Multi-Party Computation* (SMC) problem is defined as the situation where N parties wish to securely compute the value of a function $f(x_1, ..., x_N)$, where each party i delivers exactly one input x_i. During execution, no information about the x_i must be leaked to any party $j \neq i$, at least no information that can not be derived from the computation result [20, 4].
A simple example for a two-party application of this problem is the so called "Millionaires' Problem"[20]. In this problem, two millionaires want to find out who of them is richer, but neither of them wants the other one to know their exact wealth. So this problem can be formulated using the terminology introduced above with $N = 2$ and $f(x_1, x_2) = 1$, if $x_1 > x_2$ and 0 else. Yao presented a simple protocol solving this specific problem in [20], along with the theoretical foundations for the development of an extended protocol.
A more practical example where SMC can be applied is *secret voting*[20], where N parties wish to secretly and securely host a voting. The result is to be computed in private without a third party that handles the voting evaluation, and without any party learning how some other party voted.
Yao's approach to a functional protocol for Two Party Computation is the *Garbled Circuit Protocol* (GCP), where a function is transformed into a boolean circuit modelling the same function, which is then altered in a way that no information can be extracted from the resulting circuit. Note that this protocol has been developed explicitly for Two Party Computation, i.e. a SMC with $N = 2$. In the following sections, we will define the basic terms and concepts that are needed for the protocol, in section 4 we will present and evaluate Yao's original GCP, and in section 5 we will discuss a few modern extensions to the GCP that seek to enhance performance and security. In section 6, we will briefly present a couple of different working implementations for the GCP, and in section 7 we conclude.

2. DEFINITIONS

This section defines the security terminology we will use in this paper, it will mostly rely on the terms introduced by [17].

2.1 Security requirements

When one wishes to evaluate some function f using SMC, one needs to cover some requirements in order to make the securely computed function f_s a correct secure computation for f. In [19], Yao introduced the concept of comparing a protocol to an "ideal-oracle" that fulfills the three requirements listed below, and that a *Secure Function Evaluation* (SFE) is correct if it performs exactly like this ideal-oracle[17].
The ideal-oracle evaluates a function f with inputs x_1, x_2 that are delivered by two parties and outputs the value to both parties without revealing the inputs.

2.1.1 Validity

The most obvious requirement is that the evaluation of f_s must always deliver a correct result just as f would. So $f(x) = f_s(x)$ must hold true for all x for f_s to be a valid evaluation function for f.

2.1.2 Privacy

The *privacy* definition for the ideal-oracle forces a SFE system to prevent any party from learning the other party's input like the ideal-oracle would, provided that the protocol is carried out correctly. It is interesting to note that this does only guarantee that there are no unwanted values leaked during protocol execution and that this privacy definition does not account for any party trying reverse engineering methods on the result, e.g. an addition could be

doi: 10.2313/NET-2016-09-1_10

computed privately complying with this definition of privacy, but a participant would still be able to learn the other participant's input by subtracting his own input.

2.1.3 Fairness

A protocol is called *fair* when it securely computes the function value and then correctly transmits the output to all parties that participated in the computation. In contrary, an *unfair* protocol is one that refrains from actually sending the output to all parties, but holds back the information for certain (or all) parties.

2.2 Adversary Models

In the evaluation section, we will evaluate our protocol with respect to different *adversary models*. In SMC, we do not deal with classic "Man-in-the-Middle" or side channel attacks. In SMC, we are communicating and cooperating with a possible attacker, so we need to take into account different levels of protocol obedience for our possible attacker.

2.2.1 Semi-Honest Adversaries

An adversary is said to be *semi-honest*, or *honest-but-curious*, when it is not willing to deviate from the protocol at any time but tries to gather as much information about the other parties as possible by using data that is leaked during protocol execution and by the output [17]. For example, in a SFE protocol a *semi-honest* participant might try to deduce the other participants' inputs from the output, e.g. by assuming a uniform distribution of values and then guessing the right value with a certain probability. Also, an adversary that is *semi-honest* will take any protocol conforming step that it can take profit from, as long as it doesn't make the adversary's position any worse.

2.2.2 Malicious Adversaries

In contrary to *semi-honest adversaries*, a *malicious* adversary will violate the protocol in an arbitrary manner, which means that the adversary might deviate from the protocol at any point of the execution to gather information about the other parties [3]. For example, the corrupt party might send incorrect values or even no values at all, or simply abort the protocol at any time [17].

3. BASIC CONCEPTS

3.1 Oblivious Transfer

An important concept needed for the execution of Yao's Garbled Circuit protocol is *Oblivious Transfer* (OT). In general, OT is the problem of sending a single value from a set of values, without either the sender learning which exact value from the set was received or the receiver finding out any other value than the one he actually intended to receive [17]. Formally speaking, we have a set of N values on the sender side and an index i with $0 \leq i < N$ on the receiver side. After execution of the protocol, the receiver has learned only the value of N_i and no N_j where $j \neq i$, and the sender has not learned i. This version is known as *1-out-of-N Oblivious Transfer* [17]. A less general version which will be used in Yao's protocol is the one for the case $N = 2$, known as *1-out-of-2 Oblivious Transfer*. A simple protocol for this version will be presented below.

3.1.1 1-out-of-2 Oblivious Transfer

1-out-of-2 Oblivious Transfer (1-2 OT) is a special case for the concept described above, where $N = 2$ and the receiver may only choose $i \in \{0, 1\}$. An original protocol version for 1-2-OT has been proposed by Rabin [16] in 1981, the protocol presented here has originally been introduced by Lindell[10]. It is secure against *semi-honest* adversaries, and provides an easy understanding which will be needed in the upcoming protocol execution. In the following section, we will call the sending party S and the receiving party R.
Assume that S holds a pair of strings (s_0, s_1) one of which is to be sent to R. R selects $i \in \{0, 1\}$, depending on whether she wants to learn s_0 or s_1. She then generates a pair of asymmetric cryptography keys (k^{priv}, k^{pub}), and in addition to that another value $k^{\perp}$ that looks like a public key to S, but to which R has no private key. Then, R chooses the working public key to be k_i^{pub} and $k^{\perp}$ as k_{i-1}^{pub}, and advertises them to S as keys for s_0 and s_1, respectively. S then encrypts s_0 with the received k_0^{pub} and s_1 with k_1^{pub} and transmits the resulting c_0 and c_1 to R, who will then decrypt her desired value c_i with the corresponding k_i^{priv}, which then results in the correct s_i. R will not be able to decrypt the c_{i-1} because she has not generated a corresponding private key for $k^{\perp}$, and S will not know which value R has actually seen. Therefore, the proposed protocol guarantees a working 1-2-OT.
This protocol is only secure against *semi-honest* adversaries. It is easy to observe that one party could obtain additional information by deviating from the protocol, e.g. R could just generate two public/private key pairs and advertise both public keys, then she would be able to receive both s_0 and s_1. An OT protocol that is secure against *malicious* adversaries will be discussed later.

3.2 Cryptographic Hash Functions

A *cryptographic hash function* is a one-way function[1] where, in addition to the one-way property, there is also no possibility to draw conclusions from the way the values are distributed, i.e. the values of the function are uniformly distributed over the function's image space. Also, a *cryptographic hash function* prevents collisions of hash values. All hash values have the same length and appear completely random[2].

4. YAO'S GARBLED CIRCUIT PROTOCOL

Until now, we have only defined the problem we want to solve with the GCP and several concepts we need for the execution. Yao's GCP assumes that every function can be represented as a boolean circuit that only consists of binary gates (AND, OR, XOR, ...). This assumption has been proven correct for polynomial functions.[4] The general idea of the protocol is to transform the function into a boolean circuit and then disguising the circuit (garbling) so that one can not derive any values from the execution (e.g. intermediate function values). We will give a short description of the protocol before describing the protocol step by step.

[1] A *one-way function* is a function that is easy to compute, but hard to invert, so $f(x)$ is easy, whereas $f^{-1}(y)$ is not feasible with polynomial time effort

[2] This property is in literature referred to as "performing like a random oracle"[17]

doi: 10.2313/NET-2016-09-1_10

4.1 Short description

Let the two communicating parties be P_1 and P_2 and $f(x_1, x_2)$ the function the two parties want to compute, where P_1 delivers x_1 and P_2 delivers x_2. First, $P1$ transforms f into a corresponding boolean circuit c_f. Each gate in c_f has a truth table that details the gate's output. P_1 then turns c_f into its garbled version c_g by garbling each gate's truth table. Now P_1 also garbles his input so that it fits c_g and sends this garbled input over to P_2 along with the complete garbled circuit c_g. Now, P_2 holds c_g and P_1's garbled input, but not the garbling procedure, so P_2 does not know how to transform and where to use her own input. P_2 receives the garbled version of x_2 by using *1-out-of-2 Oblivious Transfer*. P_2 then computes the garbled circuit c_g gate by gate and outputs the ungarbled result to P_1 to complete the protocol.

4.2 Detailed description

After this brief explanation, we will give a detailed step-by-step explanation of Yao's protocol as described in [4] and [6].

4.2.1 Boolean Circuit Representation

Let $f(x_1, x_2)$ be the function to be evaluated securely. This function is transformed into a boolean circuit c_f that satisfies $\forall x, y : f(x_1, x_2) = c_f(x_1, x_2)$, which is assumed to be possible for all functions, although only proven correct for polynomial time functions with fixed size input[17, 4].

4.2.2 Garbling the circuit

P_1 has now completed the transformation of the function f into a boolean circuit c_f. This circuit consists of binary gates, and each of these gates has a truth table to compute the gate's output. The goal now is to garble these truth tables and turn c_f into the garbled version c_g.

To show how the garbling of a gate's truth table works, we will look at a logical AND gate, let's call this gate $g^{\&}$. The initial truth table for $g^{\&}$ is shown in figure 1a[3]. Then, P_1 generates an encryption key for each possible boolean value at each wire, so in total he generates six keys, two for each input and output.

Afterwards, P_1 encrypts all entries in the output column (w_2) with the help of the corresponding input keys. The table of garbled values is given in figure 1b[4], which also includes the final garbled value. In the figure, $g^{\&}$ is the gate identifier, which merely serves as a nonce to guarantee that no duplicate encrypted values[17] appear in the whole circuit. Also, P_1 reorders the entries of the garbled truth table to further abstract from the original truth table.

The encryption serves two purposes. On the one hand, each encryption produces a random output since it uses a cryptographic hash function that is assumed to mimic a random oracle. By doing so, it removes any correlation between input and output values. Our example gate $g^{\&}$ produces three identical outputs, but all the resulting garbled values are uniformly distributed and therefore they do not offer any possibility to gather information about the actual values.

On the other hand, the encryption makes it impossible for P_2 to tamper with the circuit during the evaluation process, as she will not be able to obtain any additional values than those she is intended to receive, because she will not get the keys for the respective inputs.

w_0	w_1	w_2
0	0	0
0	1	0
1	0	0
1	1	1

(a) Original table

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0 \| \| k_1^0 \| \| g^{\&}) \oplus k_2^0$
k_0^1	k_1^0	k_2^0	$H(k_0^1 \| \| k_1^0 \| \| g^{\&}) \oplus k_2^0$
k_0^0	k_1^1	k_2^0	$H(k_0^0 \| \| k_1^1 \| \| g^{\&}) \oplus k_2^0$
k_0^1	k_1^1	k_2^1	$H(k_1^0 \| \| k_1^1 \| \| g^{\&}) \oplus k_2^1$

(b) After garbling

Figure 1: Truth table for $g^{\&}$

P_1 will garble the complete circuit, gate by gate and pass on the output values to the next gate, except for the final output gates, which do not need to be garbled as they display the actual function output which is to be learned by both participants, so P_2 can learn this value.

4.2.3 Transmitting the garbled values

After generating the garbled circuit c_g, P_1 needs to also create a garbled version of his input x_1. First, he will transform x_1 into the boolean value which corresponds to the inputs for the original circuit c_f. Then, he will replace each bit in this boolean value with the key for the corresponding input to c_g. Assume the following example, considering the gate example $g^{\&}$ from figure 1: the first bit of P_1's input is destined for w_0 and the value of this bit is 0. Then P_1 would select the corresponding key k_0^0. This replacement is done for all of P_1's input and finally results in a garbled version of P_1's input. This value is then sent to P_2, along with the garbled circuit c_g.

4.2.4 Receiving P_2's garbled input

P_2 now has c_g and a garbled version of P_1's input, but still needs a garbled version of her own input to be able to actually evaluate the garbled circuit. In the garbling process described in 4.2.2, P_1 has constructed the garbled values for P_2's possible inputs but does not have knowledge about P_2's actual input. P_2, on the other hand, knows her own input, but can not determine the keys generated for these input values.

This is where *1-out-of-2 Oblivious Transfer* as described in 3.1 comes into play. For each of P_2's input bits, she engages in an Oblivious Transfer to receive the garbled value corresponding to her input. In this case, P_1 is the sender and P_2 is the receiver, P_1 inputs (k_i^0, k_i^1), where i is the current wire's identifier, and P_2 inputs either 0 or 1, depending on the actual input. That way, P_2 learns a full garbled version of her input without either P_2 learning more than needed about the circuit or P_1 learning P_2's input, and is then able to compute the circuit.

4.2.5 Circuit evaluation

Now that P_2 has received her garbled input, she can look up the value for each output wire. Since she has no idea which table entry they belong to, she has to try the decryption for all four possible outputs. Provided that the protocol has been followed correctly by both parties, only one decryption will work, all others will result in $\perp$. The result is then used as input for the next gate in the circuit, or the output.

[3]Here, w_0 and w_1 are the gate's inputs whereas w_2 is the output

[4]The terminology for the keys is as follows: k_a^b means that this is the key for w_a having the value b

doi: 10.2313/NET-2016-09-1_10

A short example: again, we look at our example gate $g^{\&}$. Assume P_2 has looked up the input keys k_0^0 for w_0 and k_1^1 for w_1. She then computes $H(k_0^0|k_1^1|g^{\&})$, and by $\oplus$-ing that value with the garbled output value from the truth table, she receives the value for k_2^{05}, which is then carried over to the next gate and used as input key.
This procedure continues through the complete circuit, until finally, the circuit outputs the resulting bits to P_2 who then assembles them into the correct output for f. To complete the protocol, P_2 has to output the value to P_1.

4.3 Evaluation

In this section, we evaluate Yao's original protocol with respect to mainly two aspects, *security* and *performance*. In a later section, we will provide improvement proposals that have been found in recent research for these two aspects. We will then also evaluate these improvements with respect to their feasibility.

4.3.1 Security

As mentioned above, Yao's protocol is secure against *semi-honest* adversaries, but not against *malicious* ones. That finding is quite trivial to observe. On the one hand, the OT protocol we introduced is only secure against *semi-honest* adversaries, and on the other hand, there exist other points during execution where one of the parties could gather additional information by just not following the protocol. There are various possible attack points for *malicious* adversaries in Yao's protocol, a malicious P_1 could generate a corrupt circuit c_g computes a value different from the original function f, and also, one party could send over corrupt values to allow for some reverse engineering to get the other party's input, for example. The model of the *semi-honest* adversaries is not realistic in a real-world context, as we can not expect a communicating party that we do not trust with our input to not take the chance of stealing it by deviating from the protocol. At least, we need to develop a mechanism to discover *malicious* behavior.

4.3.2 Performance

The GCP presented above solves the SFE problem in polynomial time, at least against *semi-honest* adversaries[6]. But there exist SFE problems that demand for the generation of circuits with possibly over 1 billion gates[7]. Each gate in this circuit is represented and stored as four output keys (one for each wire-value combination), each key being a multi-byte string. Assuming a key length of 160 bit (e.g. with SHA-1 as hash function), this yields a total circuit size of about 800 megabytes for the billion-gate circuit. Each P_1 and P_2 need to keep this circuit stored during execution, and the complete circuit needs to be transferred from P_1 to P_2. This leads to different possible enhancement points for performance. The first one is optimizing the communication between the two parties to lower the amount of data that needs to be exchanged, then we could improve how P_2 evaluates the circuit, and we could develop a better way of constructing the circuit. We will show an example method for each of these fields in the following section.

[5]This is possible since $\oplus$ is self-inverse

[6]After applying the enhancements proposed in 5.1, this also holds true against *malicious* adversaries

5. POSSIBLE ENHANCEMENTS

As we have shown in the previous section, the GCP still has some weaknesses considering security and performance, which this section is seeking to abolish or at least improve.

5.1 Security

Here we introduce some solutions from recent research that attempt to solve these problems. For each of these areas, there are many different solutions that sometimes take completely different approaches, but we will limit the explanation and only introduce one protocol version at most.

5.1.1 Securing Oblivious Transfer

In this section, we introduce a protocol to secure OT against *malicious* adversaries as described in [1]. This protocol is based on the Diffie-Hellman[2] assumption: with a given g^x and g^y without knowledge of x or y, computing g^{xy} is hard. The protocol steps are listed below.
Let p be some prime number, g a generator for the group $\mathbb{Z}_p^*$ [7] and C some element in $\mathbb{Z}_p^*$. Assume that the tuple (p, g, C) is known to everyone, but the discrete logarithm of C is kept secret. Also, we have the sender S holding two strings $\{s_0, s_1\}$ and the receiver R wishing to receive s_i with $i \in \{0, 1\}$ as in our original OT protocol.
Before the actual OT starts, R needs to generate his public key. He does so by randomly picking $i \in 0, 1$ and $x_i \in \{0, \ldots, p-2\}$ and then calculating $\beta_i = g^{x_i}$ and $\beta_{1-i} = C \cdot (g^{x_i})^{-1}$. Then, β_0 is R's public key with x_i being the corresponding private key. We observe that the equation $\beta_0\beta_1 = C$ holds, therefore it is possible for S to check that R's public key is correct. Also, since the discrete logarithm of C is not known, R can not know the discrete logarithms of both β_0 and β_1, but only for the β_i that has been generated to be the working public key with x_i being the corresponding logarithm. In addition to that, S can not find out which β_i R knows because they are randomly distributed[8].
S now chooses random $y_0, y_1 \in \{0, \ldots, p-2\}$, calculates $\alpha_0 = g^{y_0}$ and $\alpha_1 = g^{y_1}$ and sends them to R. In the same course, S also initializes $z_0 = \beta_0^{y_0}$ and $z_1 = \beta_1^{y_1}$ and sends $r_0 = s_0 \oplus z_0$ and $r_1 = s_1 \oplus z_1$ to R. Now, $P2$ can compute $z_i = a_i^{x_i}$ and is then able to retrieve s_i by computing $s_i = z_i \oplus r_i$.
Those who are familiar with the matter will recognize a great similarity to a Diffie-Hellman key exchange which relies on the possibility for two parties to generate a mutual key pad without receiving the other party's key part. To decrypt r_i, R needs to compute z_i, which S initialized as $z_i = \beta_i^{y_i}$, but R does not have y_i, only x_i, and also $\alpha_i = g^{y_i}$. This makes $\alpha_i^{x_i} = g^{y_i^{x_i}} = g^{y_i \cdot x_i}$. In addition, recall that $\beta_i = g^{y_i}$ and z_i was constructed as $\beta_i^{x_i}$. This in return makes $\beta_i^{y_i} = g^{x_i^{y_i}} = g^{x_i \cdot y_i}$, which is notably the equal to $\alpha_i^{x_i}$, as multiplication is commutative over $\mathbb{Z}_p^*$. Therefore, R can construct the same pad as S and decrypt r_i.

5.1.2 Better security in garbling process

We also need to secure the actual circuit construction, because the circuit constructing party could easily generate a

[7]As our working group is $\mathbb{Z}_p^*$, all arithmetic in this section will be done mod p, so g^x is actually short for $g^x \bmod p$

[8]They are randomly distributed over the set of all tuples (x, y) with $x \cdot y = C$ and $x, y \in \mathbb{Z}_p^*$

doi: 10.2313/NET-2016-09-1_10

corrupt circuit that, using a simple example, just outputs the other party's input (a simple example would be $f(x, y) = y$). Therefore, we need to come up with a solution that forces P_1 to construct a correct and non-malicious circuit. An idea of such a solution is given here, based on [12].
The mechanism we introduce here is called *cut-and-choose*. Instead of just garbling the circuit, P_1 now generates m garbled versions c_i of the circuit, but each of them is garbled differently, i.e. with different keys for output encryption every time. In addition, P_1 also computes the corresponding garbled inputs x_i, with $0 \leq i < m$. Now, P_1 also generates a commitment $C_i = H(x_i)$ for each $0 \leq i < m$, where H is some *cryptographic hash function*. Then, P_1 sends all m circuits to P_2, along with the respective commitments C_i, and also structural information, i.e. which keys have been used on the respective gates.
P_2 then randomly chooses a $0 \leq j < m$ and has P_1 de-garble all circuits except for the jth one to inspect the other $m-1$ circuits. If any of them is malformed, i.e. does not match the garbled version, P_2 assumes that P_1 is malicious and aborts the execution. If all circuits have been checked and proven correctly formed, P_2 continues to receive P_1's garbled input like in a normal execution of the protocol. But first, P_2 verifies that P_1 did not send a corrupt input by checking if $C_j = H(x_j)$. If so, P_2 aborts, else P_2 assumes P_1 is not sending corrupt data and continues the protocol.
This example protocol is supposed to give an intuition of the *cut-and-choose* concept, the concept can be further improved by choosing a probabilistic approach as described by Lindell and Pinkas in [11]. Instead of taking $m-1$ circuits for proof of correctness, P_2 will only check circuit correctness on $m/2$ circuits, then compute the other half and take the majority result of this computation as correct output. Instead of immediately aborting execution when a corrupt circuit has been detected, P_2 will continue evaluating all $m/2$ circuits and then take the majority result. To understand the reasoning behind this, consider the following example: P_1 constructs all circuits correctly, except a single one, where the output is the actual function value $\oplus$'ed with P_2's first input bit. Then, P_1 could just observe if P_2 aborts the computation prematurely, thus has encountered the corrupt circuit, or returns a value, then P_1 can also draw conclusions on P_2's input. A more detailed description of this attack can be found in [17].

5.1.3 Corrupt inputs

By now, we have ensured that a *malicious* adversary can not exploit the *Oblivious Transfer* phase, and also that the circuit has been constructed correctly. But an adversary could still inject corrupt inputs to gain information from the other party, as shown in this short example:
During the actual OT process, S is not able to find out anything about R's input, but this does not imply that S does not have any other way to learn about R's input. S can send over the correct garbled value for 0, but a corrupt value for 1. If the requested value was 0, R will continue the protocol execution, if not, R will notice the corrupt value and abort due to our secured protocol. But either way, by watching R's behavior, S is able to find out which value S requested. Again, Lindell and Pinkas have come up with a solution to this security issue in [11]. Assume that S's input is the boolean form of x_2 with $|x_2|$ being the number of bits. To secure the protocol against leaking information to corrupt inputs, we have P_2 replace each bit of his input x_2 by a combination of XOR-gates that take s new input bits introduced by P_2. P_1 can now no longer corrupt P_2's input, but only the new inputs to the XOR-gates. The more new bits P_2 introduces, the more ways she has of constructing her actual input through the XOR-gates.

5.1.4 Open Problems

Although these measures have abolished a lot of the security issues of the original protocol, there are still a few remaining that are yet to be solved for the *malicious* setting. One of them is to make sure that P_2 returns the correct result of the circuit computation. By now, P_2 could just send P_1 a wrong result for the circuit computation and keep the actual result to herself, or even send no result at all, or the other way around, P_1 could leave the output encrypted and by that force P_2 to send the output, but then P_1 could keep the result to himself. A unilateral solution for this problem is yet to be found, some work has been put into at least making sure that a returned solution is correct.

5.2 Performance

In the last section, we have aimed to provide a more secure version of Yao's protocol, now we will try to improve the protocol's performance. But first, we need to give a few thoughts on the terms *performance* and *efficiency*. Snyder states that Yao's protocol can be executed in polynomial time, even with the security enhancements made in 5.1[17], which is comparably performant and efficient by means of execution time. But nevertheless, as mentioned before, the protocol is very costly, even quite simple operations on comparably small operands demand for the generation of circuits with possibly billions of gates, which for many cases is too expensive and therefore makes the use of GCP in this situation impractical and nearly impossible.
We have in general three possible points in the protocol where we could try to make the protocol execute more efficiently. These are optimizing the communication, i.e. reducing the amount of data that needs to be exchanged, optimizing the actual execution time of each gate and finally cutting unnecessary gates from the circuit by finding more efficient circuit constructions

5.2.1 Optimizing communication traffic

The major cost generator in the GCP is in the most cases the transmission of the garbled circuit from P_1 to P_2. Kreuter et al. [9] have found that calculating the *Levenshtein Distance*[9] of two strings of approximately 500 bytes requires around 5.9 billion gates. These gates are connected by wires, of which each is represented by four keys. A key can be assumed to be a multi-byte string, and even if you make the unrealistic assumption that each key is one byte long, this still results in the c_g's size exceeding 20 gigabytes (for our *Levenshtein distance* example. So, we will first have a look at some ideas on how to lower the transmission needs. Note that this section will solely focus on communication optimizations, which sometimes might come with extra computation

[9]The *Levenshtein Distance*, also known as the edit distance, is the minimum number of basic bit operations (insertion, deletion, bit flipping) needed to transform one string into another, and is not to be confused with the *Hamming distance*. It is commonly used as a benchmark for two-party-computation.

doi: 10.2313/NET-2016-09-1_10

time, but this is acceptable since communication traffic is making for the vast part of the GCP, because transmitting data via a network is slower than processing the same data on a CPU.
The procedure we will discuss here is called *Random Seed Checking* and has originally been presented by Goyal et al.[5]. In detail, this procedure consists of two parts that each base on the same idea. We recall that the construction of the c_g involves P_1 assigning each wire in the circuit a random value (see 4.2.2). Instead of doing so, we have P_1 choose a random seed and then construct the wire keys deterministically based on that seed, like a pseudo-random generator. For the second part of this version, we recall the *cut-and-choose* procedure described in 5.1.2 to secure the protocol in the presence of *malicious* adversaries, where P_1 would generate m semantically equivalent circuits and corresponding commitments and then P_2 would check the correctness of $m-1$ circuits to prove that P_1 is not generating corrupt circuits. We also discussed the majority result optimization to that concept, which will be further enhanced performance-wise here. In this optimization, P_1 will not send P_2 m full circuits, but only the *Random Seeds* that have been generated to deterministically build each circuit, and also a "commitment", that can, for simplification, again be seen as some sort of *Cryptographic Hash Function*. P_2 then generates the $m/2$ circuits to check herself from the seeds he got from P_1[10]. After the construction, P_2 can check a circuit's correctness by checking the corresponding commitment and will then, if the proof of correctness succeeded, request the remaining $m/2$ circuits to evaluate and continue as described before.

5.2.2 Optimizing Circuit Evaluation

After reducing the amount of communication traffic needed for the exchange of the garbled circuits, we will reduce the amount of resources needed for the actual circuit execution with respect to execution time and computational power. To delimit this section from the following one, in this section we discuss improvements to the circuit execution that can be applied without any changes to the circuit's internal structure.
The first approach towards achieving such optimization we will discuss here is the *Fast Table Lookups* technique as described in [7][11]. The idea of this procedure is to facilitate the gate evaluation for P_2 by hinting her on which table entry for the next gate to use. In the original protocol, P_2 needs to decrypt all four garbled output values to find out which is the real one (the one value where the decryption works). In this improved version, P_1 appends an extra bit to each garbled output value. The two extra bits from the input wires form an index in the range $\{0,\dots,3\}$ that points to the next encrypted value that can be decrypted with the received input keys. Due to the fact that the table rows are mixed, this does not constitute a security flaw, since there is still no way to deduce the mapping of keys to wire values.
Another possible approach for execution optimization is aiming to parallelize the execution. Normally, the protocol execution is done in a linear manner, P_1 generates the circuit (or possibly all m circuits when utilizing the *cut-and-choose* procedure), while P_2 is idle, because she needs the circuits first to start her procedure. The idea of *Pipelined Circuit Execution* has been originally presented by Huang et al. in [7] where they came to the conclusion that it is possible for P_1 to parallelize the execution by sending P_2 parts of the circuit as soon as possible, ideally right after generating them, and P_2 will start evaluating them, as long as there are gates to which the input is available. First, this reduces the amount of memory needed, since no party is obliged to store the entire circuit in their memory, but keeps on evaluating the gates. Additionally, it also reduces the computation time, roughly according to the following formula. Normally, the execution consists of the garbling time (t_{garble}) plus the time needed for transmitting P_2's garbled input via OT (t_{OT}) and the time P_2 needs to evaluate the garbled circuit (t_{eval}), so the total circuit execution time sums up to $t_{total} = t_{garble} + t_{OT} + t_{eval}$, whereas when using circuit pipelining, the total execution time will only be $t_{total} = max(t_{garble}, t_{eval}) + t_{OT}$[12]. A major downside of this proposed procedure is that it is only secure against *semi-honest* adversaries per default[17], because it doesn't comply with the *cut-and-choose* strategy proposed in 5.1.2, as this has P_1 generate m circuits that need to be generated and stored together for P_2 to be able to choose a set of circuits to evaluate, the same also holds true for *random seed checking*. Kreuter et al. have developed a solution to secure *pipelined execution* against *malicious* adversaries where P_1 has to generate each circuit twice, possibly using *random seed checking*, once like he normally would, starting to send the circuits to P_2 right away, and again after P_2 has chosen which circuits he wants to evaluate [9].

5.2.3 Optimizing Circuit Construction

While in the last section, we presented concepts to execute the circuit faster without actually altering it, we will try to achieve improvements to the GCP's resource needs by making the circuit construction more efficient. The general idea of this class of approaches is to significantly reduce the number of garbled values that need to be generated, stored and exchanged.
The first approach to this is the trivial strategy of simplifying the circuit and therefore having less gates to be garbled. The main part of this approach is to remove inefficiencies from the circuit, i.e. to eliminate partial circuits that always evaluate to a constant or that can be represented by fewer gates. While this might sound like a simple and minuscule improvement, Pinkas et al. [15] have shown that it is possible to improve *Fairplay's*[13] performance by 60 %.
Kolesnikov and Schneider[8] have developed another approach that goes beyond simply reducing the number of gates that are generated by the function-to-circuit transformation. They introduce the possibility of replacing a garbled XOR gate by a standard XOR operation, which they call the *free XOR* technique. In the default protocol, P_1 generates the garbled values representing 0 and 1 for each wire at each gate at random. When using the *free XOR* technique, we instead have the values be generated in relation to XOR gates. This allows P_2 to simply execute a single XOR operation for each such gate instead of the lookup and decryption of values from the garbled truth tables. The following description of

[10] P_1 also delivers any needed information about each circuit's structure
[11] The original concept has been postulated in [12].
[12] At least in the ideal case.
[13] *Fairplay* will be introduced in 6.1

doi: 10.2313/NET-2016-09-1_10

the *free XOR* algorithm is taken from Snyder[17].
Let k be the length of the garbled values generated for the circuit. We have P_1 generate a secret k-bit integer $K \in \{0,1\}^k$. Let G be the set of all XOR gates contained in the generated circuit, and g such a gate with two input wires, of which the input values are delivered by the two gate g_{in_0} and g_{in_1}. Also, let $k^b_{in_i}$ be the key that corresponds to the input gate g_{in_i} having the value $b \in \{0,1\}$. Then, set $k^1_{in_0} = K \oplus k^0_{in_0}$ and $k^1_{in_1} = K \oplus k^0_{in_1}$ for all gates $g \in G$ and replace the gate g with some function that returns $k_{in_0} \oplus k_{in_1}$. This eliminates the need to hold garbled truth tables for all XOR gates and thus reduces the size of the garbled circuit by $|G| \cdot |k|$, with k again being the size of a garbled value in the circuit. A more detailed example of how to derive the correct output values using the XOR operations is given by Kolesnikov and Schneider in their paper on *free XOR*[8].

5.3 Combining different strategies

As we mentioned before, some of the proposed improvements conflict with each other because they are doing different approaches to different fields of problems, and the respective measures contradict each other. In some cases, the different strategies don't interfere with each other, but other concepts might not be feasible at the same time by default. In these cases, additional effort needs to be taken to make the different concepts work together. We already discussed the possibility of running *pipelined circuit execution* together with a *cut-and-choose* approach, and the necessary additional computation load in this case about doubles the effort, as P_1 needs to generate each circuit twice. So, before blindly applying different improvements to an implementation of the GCP, one needs to consider potential side effects the enhancements have on each other. For some improvements that initially contradict each other, there exist solutions to make a parallel application of two conflicting approaches possible, but this requires the development of additional strategies dedicated to this single purpose. Another example where such extra work is needed is combining Huang et al.'s *Fast Table Lookups* and the *Free XOR* technique[17]. The *free XOR* technique is not trivially capable of handling the extra bits appended to the output to directly determine the next table entry used, and the solution to this problem presented by Kreuter et al.[9] requires some more effort and will thus not be discussed here, it is just used as an example to illustrate the difficulties that come with the desire to implement multiple enhancement strategies.

6. IMPLEMENTATIONS

Originally, Yao developed his GCP as a theoretical concept, and for a long time, an actual implementation of the protocol did not seem possible as the needs in computation power and memory consumption were apparently too expensive to afford a working implementation. But, as predicted by Moore's Law[14], the number of transistors on integrated circuits and thus the computational power grew exponentially over the years, and so did the possibility of realizing a computation-intensive protocol like the GCP. So, in this section we will present some actual implementations. The principle is similar throughout the different approaches, they offer the possibility to enter the function to be computed in a high-level language and compile those to a language dedicated to describing boolean circuits, which are then garbled and sent.

6.1 Fairplay

When new versions of the GCP are developed, the developers almost always compare them to the *Fairplay*[12] implementation. Malkhi et al. developed the approach in 2004, and it is considered one of the first ever implementations for the GCP. Recent implementations use *Fairplay* as a benchmark for comparison, and build upon it. *Fairplay* implements the simple *cut-and-choose* strategy proposed in 5.1.2 to provide basic security against *malicious* adversaries, and also uses the *Fast Table Lookups* technique. It uses its own *Secure Function Definition Language* (SFDL) and is compiled to VHDL-like *Secure Hardware Definition Language* (SHDL) that describes the circuit, which is then output as a Java object. The sample problems that were used to test and evaluate the *Fairplay* implementation can be considered rather simple: the "Billionaires' Problem", a version of the Millionaires' Problem with larger numbers, a bitwise AND operation, a database search for keyed elements and finding the median of two arrays of size 10. The latter problem required the construction of 4383 gates and a computation time of 7.09 seconds[12]. The problems might have been simple, but *Fairplay* was certainly a proof of concept for the possibility of implementing Yao's GCP.

6.2 Huang et al.

In [7], Huang et al. present an implementation that aims to achieve a high performance above all. They implemented various performance enhancements that we have presented in this paper, such as the *Fast Table Lookups*, *Free XORs* or *Pipelined Circuit Execution*, which led to astonishing computation speedups, but they only focused on performance, meaning they did not secure their protocol against *malicious* adversaries. In contrary to *Fairplay*, Huang et al. tested their protocol against more difficult problems and compared them to the previously best known implementation solving the respective problem.They used problems such as the *Hamming Distance* of 900 bit strings where they achieved an overall speedup of 4100 times (213 seconds to 0.051 seconds)[7], the *Levenshtein Distance* of 200 bit strings which produced the largest computed circuit with over 1 billion gates, or AES-128. AES had already been implemented using garbled circuits by Pinkas et al. [15] using *Fairplay*'s SFDL, but Huang et al. follow a different approach by orienting towards the traditional AES code. A complete description of their AES approach would be beyond the scope of this paper, though.

6.3 Kreuter et al.

Kreuter et al.[9] again focused on developing an implementation for security against *malicious* adversaries. They created an own language for entering functions, such as *Fairplay* did with the SFDL, and also a new compiler that they compared against the *Fairplay* compiler and which brought a vast speedup in generating larger circuits. Snyder labels Kreuter et al.'s implementation the "state of the art"[17] when it comes to providing an implementation of the GCP that is secure against *malicious* adversaries, they implemented all of the aforementioned security enhancements (5.1) and also all performance improvements (5.2), they also developed additional techniques to employ strategies that were originally

doi: 10.2313/NET-2016-09-1_10

contradicting each other (see 5.3). They evaluated their system against typical "benchmark problems" such as AES or the *Levenshtein distance*, as mentioned before, where they used two 4095-bit strings as input and computed the resulting circuit consisting of over 5.9 billion gates in 8.2 hours, which they compared to Huang et al.'s implementation that computed the result faster, but is only secure against *semi-honest* adversaries.

6.4 Mood et al.

The implementation we will present in this section is slightly different from the others, as it is written to make using the GCP on mobile devices possible. Mood et al.[13] wanted to adapt to the trend of smartphones becoming increasingly popular in 2012 and took the challenge of creating a GCP port for Android. They used *Fairplay*'s SFDL to define functions and a specialized *Pseudo Assembly Language Compiler* (PALC) to compile to *Pseudo Assembly Language* (PAL). The PALC was designed specifically for Android devices and provides a significant improvement over a direct *Fairplay* port. On Android devices, the problem size is not only limited by execution time and memory, but in addition also by Android itself restricting the file size to 4 GB[13]. Mood et al. evaluated their implementation on 2011 made HTC Thunderbolts, which have 768 MB of RAM, which is significantly lower than todays Android phones have available, so it is possible that larger circuits are feasible on today's Android phones than have been in 2012. In addition to the usual benchmarking process of evaluating their implementation against various problems, they also developed an actual Android application where a user can on the one hand solve the same benchmarking problems as Mood et al. did and on the other hand engage a password vault where a password can be encrypted and only unlocked if the two parties that created the vault enter their password.

7. CONCLUSION

The goal of this paper was to give the reader a firm understanding of Yao's GCP and all its basics, along with an overview of enhancements that have been introduced during the 30 years since this protocol was first introduced. We also presented some state of the art implementations that might be further developed into an actual GCP-based security framework in the following years, with computation power still growing, although Moore's Law might no longer be applicable[18]. It is to be noted that SFE is an open research field in cryptography, and there is a lot of effort put into developing new approaches to SFE as well as to further enhancing the GCP.

8. REFERENCES

[1] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology – CRYPTO '89 Proceedings*, pages 547–557. Springer, 1989.

[2] W. Diffie and M. E. Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, pages 644–654. IEEE, 1976.

[3] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, pages 86–97, 1998.

[4] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[5] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology - EUROCRYPT 2008*, pages 289–306. Springer, 2008.

[6] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[7] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, volume 201, 2011.

[8] V. Kolesnikov and T. Schneider. Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. 2008.

[9] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 14–14. USENIX Association, 2012.

[10] Y. Lindell. Secure two party computation in practice. lecture given at technion-israel institute of technology tce summer school 2013, 2013. `https://www.youtube.com/watch?v=YvDmGiNzV5E`.

[11] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2007*, pages 52–78. Springer, 2007.

[12] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[13] B. Mood, L. Letaw, and K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *Financial Cryptography and Data Security*, pages 254–268. Springer, 2012.

[14] G. E. Moore. Cramming more components onto integrated circuits. In *Proceedings of the IEEE*, volume 86, pages 82–85, 1986.

[15] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Advances in Cryptology–ASIACRYPT 2009*, pages 250–267. Springer, 2009.

[16] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[17] P. Snyder. Yao's garbled circuits: Recent directions and implementations, 2014.

[18] M. M. Waldrop. `http://www.nature.com/news/the-chips-are-down-for-moore-s-law-1.19338`. Accessed: 2016-06-20, 10:21.

[19] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[20] A. C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

ISBN 978-3-937201-52-8
DOI 10.2313/NET-2016-09-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)