OFFWall: A Static OpenFlow-Based Firewall Bypass

Bastian Germann[1] Mark Schmidt[2] Andreas Stockmayer[3] Michael Menth[4]

Abstract: Stateful firewalls are becoming bottlenecks for high-speed communication networks. To counteract, trusted network flows may statically bypass the firewall. As access control lists (ACLs) of moderately priced switches do not allow port selection, they cannot be used for implementation of a static firewall bypass. In this work, we present a software-defined networking (SDN) based solution for a static firewall bypass based on moderately priced commodity hardware. We propose OFFWall, an OpenFlow (OF) controller that translates a whitelist of trusted flows into flow rules and installs them on an SDN switch to implement the firewall bypass.

OFFWall has been developed according to the demands of network administrators. Its goal is simplicity and stability so that it can run for long time without updates. Therefore, it is programmed in Rust for runtime stability and compiled to an executable file. Moreover, it offers only a minimal feature set required to install and remove flow rules on the switch.

After successful tests in a virtual and physical setup with different complexity, we deployed OFFWall on the network of the Department of Computer Science of the University of Tuebingen.

Keywords: SDN; OpenFlow; firewall; bypass; whitelist; traffic flow; Rust

1 Introduction

With increasing transmission rates in communication networks, firewall clusters at moderate cost become bottlenecks. Diverting some traffic around the firewall can alleviate the bottleneck effect. Network traffic that is considered secure, e.g., internal traffic that is forwarded to a different net within the same organisation, can be bypassed, which we refer to as "firewall bypass."

Fig. 1 depicts the general concept of a firewall bypass. The router and the switch are the edge devices between two networks. The switch is connected to a firewall. A packet from the switch's outside port (1) is forwarded to the firewall-out port (2) by default. After firewall

[1] University of Tuebingen, Chair of Communication Networks, Sand 13, 72076 Tuebingen, Germany
bastian.germann@student.uni-tuebingen.de
[2] University of Tuebingen, Chair of Communication Networks, Sand 13, 72076 Tuebingen, Germany
mark-thomas.schmidt@uni-tuebingen.de
[3] University of Tuebingen, Chair of Communication Networks, Sand 13, 72076 Tuebingen, Germany
andreas.stockmayer@uni-tuebingen.de
[4] University of Tuebingen, Chair of Communication Networks, Sand 13, 72076 Tuebingen, Germany
menth@uni-tuebingen.de

processing, it is received at the firewall-in port (3) and output to the switch's inside port (4). A packet from the inside network is forwarded in the opposite direction by default. The basic idea of a firewall bypass is a shortcut for trusted network flows from the outside port to the inside port of the switch and in the opposite direction.

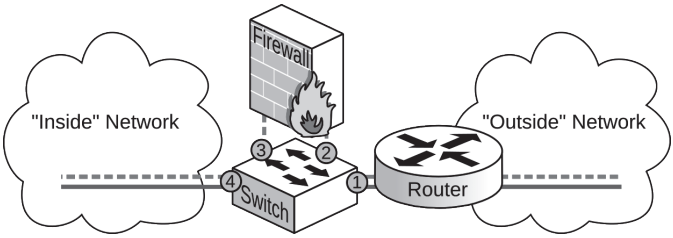

Fig. 1: A firewall bypass.

A *static* firewall bypass uses static whitelist rules in the switch to define network flows that are bypassed. All other flows take the default route through the firewall. A whitelist rule maps a 5-tuple (source IP address, destination IP address, protocol, source port, destination port) plus the incoming switch port to the intended output switch port.

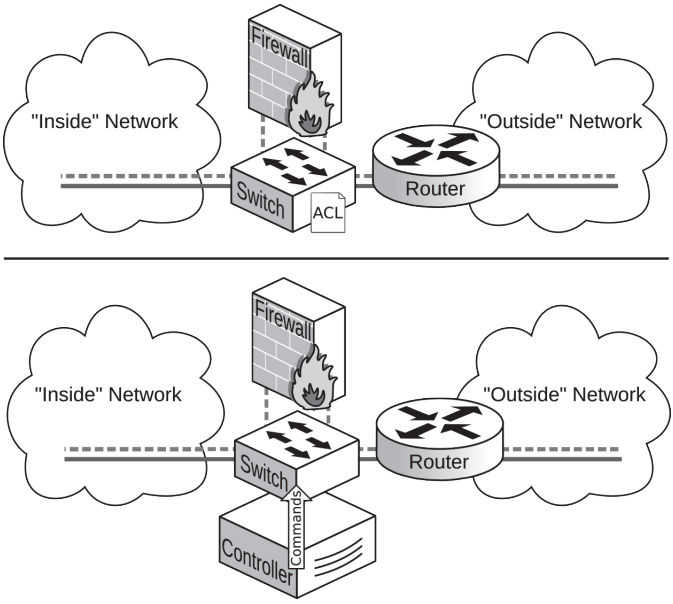

Fig. 2: A firewall bypass can be implemented via ACLs or SDN.

Fig. 2 depicts two possible ways to specify whitelist rules for network flows. One way involves a switch's built-in ACL. We found the ACLs of different Layer-3 switch devices to be insufficient for this task. E.g., the edge device used at our campus network does not

support port selection of outgoing packets via ACLs. Devices that have this feature are much more expensive.

In this paper, we present a firewall bypass based on SDN instead of ACLs. A switch can be utilised to implement a firewall bypass if it supports an SDN protocol to manipulate its network flows. In the SDN-based firewall bypass, the default traffic flow and the firewall bypass rules are installed on an SDN-capable switch by an SDN controller that is run in a separate management network. This approach is a cost-effective alternative to acquiring a device that provides sophisticated enough ACLs for this problem.

The most widely used SDN protocol that addresses manipulating network flows is OF. It is available in many Layer-3 switch devices. OF switches are programmed by an OF controller. The rules controlling the network flow are called flow entries. There are several OF controller frameworks which offer a rich feature set but lack runtime stability that is desirable for continuously operating an OF controller for an edge device. We observe that for a firewall bypass only a small subset of the OF specification is utilised on the controller side to install flow entries.

Therefore, we propose OFFWall [Ge] which is a functionally reduced OF controller. This controller implements just enough OF specifics to proactively install flow entries on an OF switch. All of these specifics belong to the required OF feature set. This approach allows for a small amount of source code in our implementation that runs solidly and is easy to reason about.

The rest of the paper is structured as follows. Sect. 2 reviews related work on firewall bypassing and OF. Sect. 3 elaborates on the overall architecture and implementation of OFFWall. In Sect. 4, we validate the achievement of OFFWall's design goals with tests and present the final deployment. Sect. 5 concludes this work.

2 Related Work

In this section, we first present an intuitive Layer-3 based concept for a firewall bypass that we discarded for practical reasons. Then we introduce background knowledge of SDN and OF. Finally, we report on dynamic firewall bypasses that have been considered in scientific work and projects.

2.1 A Router-Based Static Firewall Bypass

Given, the Cisco ASA firewalls in use at our campus network can be operated in transparent or routed mode (Chapter "Transparent or Routed Firewall Mode" in [Ci17]), there is another intuitive, valid approach to firewall bypassing.

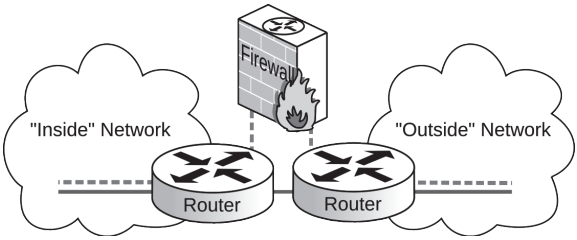

Fig. 3: Firewall bypass with two routers.

Fig. 3 shows a static firewall bypass where traffic is routed via two routers and a firewall which runs in routed mode, thus cannot be operated in transparent mode.

In this scenario, the rules for firewall bypassing can be expressed with the corresponding routes on each router. The routes have to be kept in sync in all routing tables which is non-trivial to automate.

Running a firewall in routed mode makes the firewall appear as a hop on Layer-3 and decreases the forwarded IP packets' time to live whereas in transparent mode, the firewall acts as a Layer-2 bridge device.

2.2 SDN and OpenFlow

SDN separates a network's data plane from its control plane. Unlike traditional standalone switches, SDN switches require a controller to configure their forwarding behaviour. A so-called southbound interface defines the communication between a switch and its controller.

OF is an SDN southbound interface. Its specification is published by the Open Networking Foundation (ONF) [Op15].

OF uses flow entries to define forwarding behaviour in a switch's OF tables. The flow entries are added and deleted on the switch by flow mod messages.

All published OF versions provide structures for flow entries that are sufficient to express firewall bypass rules. Flow match structures can match a 5-tuple plus the incoming switch port. A flow entry can hold instructions, e.g., an action can be applied to a packet. We need the output action which supports output switch port selection. Therefore, OF provides all capabilities needed for implementing an SDN-based firewall bypass.

Popular OF controller frameworks typically provide a programming environment with a complete OF feature set across different OF versions. This allows for flexible usage of the protocol but leads to large codebases. Those frameworks tend to use programming languages that come with heavyweight runtime environments which extend their codebase

even further and have to be regularly updated by system administrators. E.g., Ryu [Ry] is implemented in Python, Trema [Tr] uses Ruby, OpenDaylight [Op] and Floodlight [Fl] use Java. To avoid dependency on a heavyweight runtime environment, we decided not to use any of these popular frameworks. An additional concern with Ryu is its usage of the eventlet library that patches the Python standard library at runtime [Ev].

OFFWall is implemented from scratch in Rust [Ru]. Another Rust OF controller is rust_ofp which is not utilised because it is in an experimental development state. Its author elaborates on the advantages of an OF implementation in Rust rather than OCaml [Ba16].

2.3 Dynamic Firewall Bypass

A *dynamic* firewall bypass installs bypass rules traffic-driven on-demand. That means, all new flows are forwarded via the firewall. The controller may sample traffic leaving the firewall and select some flows for offloading. Thus, a dynamic firewall bypass dynamically generates the set of whitelisted flows and is more complex than a static firewall bypass.

Google holds a patent on a scalable stateful firewall design in OF-based networks [Pa14]. After processing new connections in a firewall, this design differentiates connections in flow state from connections in breach state. The connections in flow state are bypassed.

The authors of Science DMZ [Da13] suggest using a demilitarized zone (DMZ) for trusted network flows. They mention the possibility of using OF to implement the Science DMZ architecture.

SciPass [BR14] and NFShunt [MH15] build on Science DMZ and extend it with an OF-based firewall bypass. Both of them are dynamic firewall bypasses. SciPass provides support for an intrusion detection system in Science DMZ which also allows for dynamically identifying trustworthy network flows for the purpose of bypassing. NFShunt discusses the use of a Linux software firewall to deliver such information to an OF firewall bypass.

In previous work of ours [He17], static and dynamic firewall bypasses are compared. An OF-based dynamic firewall bypass with rule learning by sampling traffic via sFlow is suggested. The authors aim to use the bypass only in congestion situations. They present a proof of concept implementation and an analytical performance evaluation. Based on them they observed that this approach is only feasible if the OF switch can hold a large number of flow entries that exceeds the number of feasible flow entries of typical OF switches by an order of magnitude. This work is a follow-up and implements the simpler static firewall bypass.

3 Implementation of OFFWall

We describe OFFWall's requirements, the programming environment, the implementation design, and implementation details.

3.1 Requirements

We do not rely on an existing OF controller framework and implemented an OF controller from scratch that provides only the minimal set of OF features required to install the needed flow entries on the switch.

In our use case scenario there is no need for the controller to manage more than one switch or auxiliary connections. Therefore, OFFWall controls only a single switch. OFFWall preempts using any other OF controller if the OF switch cannot specify any additional controller.

This minimalistic approach aims at providing production-quality software that runs continuously for a long time. Furthermore, it should be possible to change the bypass rules without interrupting the execution of the switch or controller. The bypass rules should be validated semantically so that each rule contains an IP address belonging to the inside network. Any other checks are not involved. The operator has to pay attention to set only rules that describe trusted flows.

The target switch supports OF 1.0 and 1.3. Version 1.3 is implemented for the controller because it specifies the OF Extensible Match format which deprecates the 1.0 flow match structure and allows for more flexible matches.

3.2 Programming Environment

We decided to use the Rust programming language which is compiled to machine code like C and C++. It does not require a special runtime environment or a garbage collector. Rust has the additional benefits of guaranteed memory and thread safety, which eliminates some classes of run-time errors that are common causes of security problems. The guarantees are given at compile-time as opposed to many garbage-collected languages, which also provide memory safety at the expense of a runtime performance penalty. They come with a strong, static type system [MK14].

Version 1.0.0 of the implementation has 2626 lines of Rust code (excluding dependencies). 459 of these lines are type definitions constructed from parts of the *openflow.h* header file that is delivered with the OF specification [Op15]. That header file consists of 2335 lines of C code and defines the types needed for a complete OF implementation.

3.3 Implementation Design

Fig. 4 depicts an overview of the following implementation description.

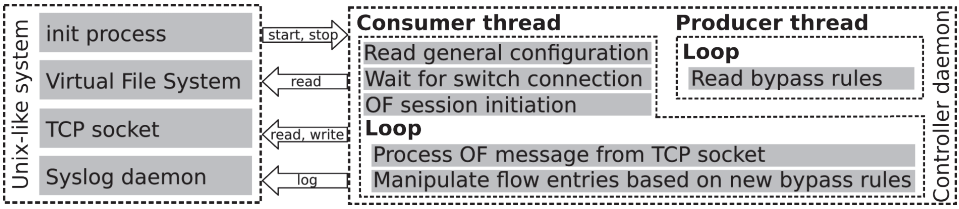

Fig. 4: Implementation overview and system interface usage of OFFWall.

The OF controller runs as a daemon on a Unix-like operating system. The daemon is started by the system's init process. It takes two command line arguments that are configuration file paths to a general configuration file and a bypass rules file. Sect. 3.4 explains them in detail.

The daemon uses two threads that communicate via a channel with a producer-consumer pattern. In the main thread (the consumer), the general configuration file is read and parsed. Then the process binds to a TCP socket and listens on it for an OF switch to connect as client. Sect. 3.5 explains the OF session initiation and the handling of the subsequent OF messages. If a connection error occurs, a new connection is offered immediately.

In the second thread (the producer), the bypass rules file is watched via the operating system's file watching mechanism. If no such mechanism is available, the file is polled once per second. When the file changes, this thread parses it and sends the modified 5-tuples to the main thread.

Various logging levels are implemented via the standard Unix logging facility Syslog and can be configured as an optional command line option.

3.4 Configuration Files

The general configuration file has INI syntax. Fig. 5 provides an example. The file contains, e.g., the controller's IP and port that it uses to open a TCP connection, the target OF table ID, and the OF port numbers in use, corresponding to the four switch ports in Fig. 1.

Fig. 6 provides an example for the bypass rules file. It has CSV syntax and holds the 5-tuples that belong to the firewall bypass rule set. All fields have wildcard support with a *, which means exclusion from the match. IP addresses are given in CIDR notation which allows for address block matches. We implemented only support for TCP and UDP over IPv4 because there is no need for firewall bypassing any other protocol at our campus network.

```
[Connection]
uri=tcp:192.0.2.1:6633

[Table]
id=0

[Networks]
inside=10.0.0.1/32

[Ports]
inside=10
fw_in=11
fw_out=12
outside=13
```

Fig. 5: An INI configuration file.

```
# src_ip;src_port; dst_ip;dst_port; proto
10.0.0.1/32; *;  10.0.0.2/32; 6060;  TCP
10.0.0.1/32; *;  10.0.0.2/32; 7070;  UDP
10.0.0.2/32; *;  10.0.0.1/32; 8080;  TCP
10.0.0.2/32; *;  10.0.0.1/32; 9090;  UDP
```

Fig. 6: A CSV file with bypass rules.

3.5 OpenFlow Message Handling

The following describes the handling of OF messages in the main thread. They are sent and received via exactly one TCP socket. All messages start with a fixed-length OF header that contains the OF version, message type, the message's length and a transaction identifier [Op15]. This structure allows for a straightforward message deserialisation.

Once a switch connects, the handshake consisting of the message types OFPT_HELLO, OFPT_FEATURES_REQUEST and OFPT_FEATURES_REPLY is exchanged. If the switch does not support OF 1.3, the connection is terminated.

By default, the switch will notify the controller on some events. Therefore, an OFPT_SET_ASYNC message is sent to unsubscribe from any switch notifications, which avoids implementing unnecessary message types.

Then the controller installs the default flow entries that forward packets to the firewall. It uses OFPT_FLOW_MOD messages for this task. The flow entries are derived from the [Ports] section of the general configuration file.

The connection is fully set up at this stage and the controller may install bypass rules. In an infinite loop new OF messages are read and deserialised.

When the controller receives a keep-alive message (OFPT_ECHO_REQUEST), it sends a reply (OFPT_ECHO_REPLY).

After a message is handled, the inter-thread channel is checked for any changed 5-tuples. According to the changes, several OFPT_FLOW_MOD messages are sent to add or delete flow entries that cause a firewall bypass. Each 5-tuple causes the creation of two flow entries:

inbound and outbound. The incoming switch port is derived from the IP range of the inside network given in the [Networks] section of the general configuration file. The switch port augments the 5-tuple in the flow entry's match structure.

4 Validation and Deployment

We present two different test environments that we used for a functional validation of OFFWall and describe the performed tests. After the successful test phase, we deployed OFFWall in an extended setup that is also presented.

4.1 Experiments for Validation

We implemented unit tests for the OF controller as base validation. Additionally, we performed system tests in a virtual environment and with a production-like setup. We describe the latter two in the following.

4.1.1 Virtualised Testbed

Fig. 7 shows the virtualised testbed which is realised with the Mininet [Mi] virtualisation environment running on a Linux system. The virtual bypass switch uses Open vSwitch with the proposed OF controller. This setup resembles the SDN-based firewall bypass in Fig. 2.

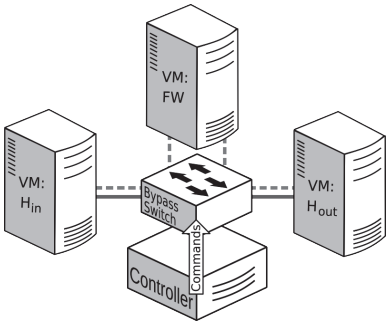

Fig. 7: Logical structure of the virtualised testbed.

The test network is set up as follows. An inside host VM (H_{in}) and an outside host VM (H_{out}) are both connected to the bypass switch enabling communication between them. To that end, they both listen on a UDP and on a TCP socket with their IP addresses and ports set according to Fig. 6. The firewall is represented by the VM FW with a link to the firewall-in port and another link to the firewall-out port of the bypass switch. We configure the VM FW such that it relays received frames between its two interfaces and logs every relayed packet.

The test verifies the correct installation, operation, and deinstallation of bypass rules on the bypass switch. We proceed in three steps: (1) transmission without bypass rules, (2) transmission with installed bypass rules, (3) transmission with deinstalled bypass rules. In each step we test bidirectional forwarding of TCP and UDP traffic between H_{in} and H_{out}. The logs of FW reveal whether traffic is bypassed.

In the first step, traffic is not bypassed and FW logs the communication. Then, the bypass rules from Fig. 6 are added to the controller's bypass rules file without restarting the bypass switch or the controller. The traffic is no longer logged by FW, i.e., bypass rules operate correctly. Finally, the bypass rules are removed from the bypass rules file without restarting the bypass switch or the controller. Now, FW logs the traffic again, i.e., bypass rules are successfully deinstalled.

4.1.2 Produktion-Like Physical Testbed

Fig. 8 depicts the production-like testbed. The bypass switch is an HP 5406zl and the firewall (FW) is a Cisco ASA 5500. The firewall is run in transparent mode and with logging enabled. It expects the incoming Ethernet frames to have a VLAN tag and also tags the outgoing frames. We did not implement VLAN support in OFFWall to keep the controller implementation minimal. As a workaround, we introduced an additional VLAN switch that VLAN-tags frames going to the firewall and untags VLAN frames coming from the firewall. This VLAN switch is a logical partition of the HP 5406zl so that no additional device is needed.

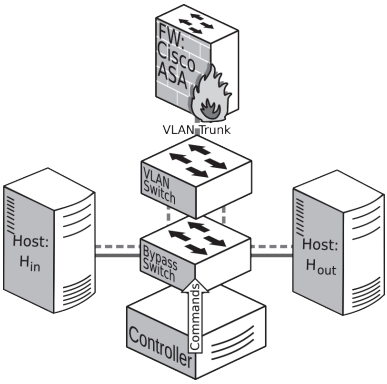

Fig. 8: Logical structure of the production-like testbed with VLAN tagging.

Each net is represented by a host (H_{in}, H_{out}) which are operated by arbitrary machines and operating systems. The tests are carried out as described for the virtualised testbed.

4.2 Deployment

After validating the OF controller implementation, OFFWall was deployed at the network of the Department of Computer Science of the University of Tuebingen. Fig. 9 depicts this setup that resembles the SDN-based firewall bypass in Fig. 2. The institutional network is the inside network whereas the backbone to the campus network and the Internet is the outside network.

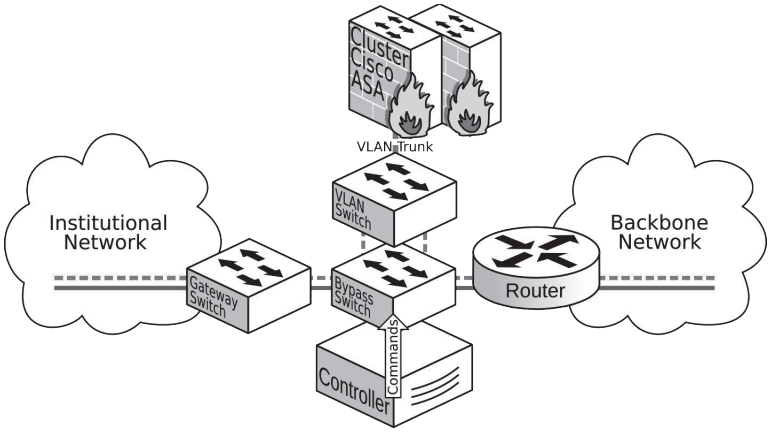

Fig. 9: OpenFlow-based firewall bypass with gateway switch and firewall cluster with VLAN tagging.

The OF-capable switch is HP 5406zl. The firewall consists of two Cisco ASA 5555 in cluster configuration and is run in transparent mode. The switch is partitioned like in the production-like testbed. One partition is responsible for the firewall bypass, the other is responsible for VLAN tagging and untagging.

When a conventional switch receives a frame from an unknown MAC address at one of its interfaces, it stores the MAC to interface mapping in its forwarding table. Future frames addressed to this MAC are then forwarded specifically to this interface (MAC learning). Usually, this learning function has to be implemented in OF switches, too. This occupies additional flow rules that are limited by OF switches. In order to avoid that, we install an HP Aruba 2930F gateway switch between the institutional network and the bypass switch. The gateway switch applies MAC learning and forwards internal and inbound traffic to the internal devices. As a consequence, the bypass switch does not need to apply MAC learning and we do not need to extend OFFWall.

5 Conclusion

As existing ACLs in switches are not flexible enough for implementation of a firewall bypass, we presented OFFWall, an OF controller with a reduced feature set for just this use case. It

is written in Rust and compiled to machine code with the purpose of runtime stability and without the need of a runtime environment.

We explained the implementation of OFFWall including configuration file structure. We reported tests on Mininet and a production-like hardware setup that were performed to validate OFFWall. We described the integration of OFFWall into the network of the Department of Computer Science of the University of Tuebingen which required an additional gateway switch for MAC learning.

References

[Ba16] Baxter, S.: OpenFlow 1.0 Protocol in Rust, 2016, URL: https://baxtersa.github.io/2016/12/30/rust-openflow-0x01.html.

[BR14] Balas, E.; Ragusa, A.: SciPass: A 100Gbps Capable Secure Science DMZ using OpenFlow and Bro, INDIS at SC '14, 2014, URL: https://scinet.supercomputing.org/workshop-sc14/?q=papers.

[Ci17] Cisco Systems, Inc.: CLI Book 1: Cisco ASA Series General Operations CLI Configuration Guide, 9.9, 2017.

[Da13] Dart, E.; Rotman, L.; Tierney, B.; Hester, M.; Zurawski, J.: The Science DMZ: A Network Design Pattern for Data-intensive Science. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. SC '13, 2013.

[Ev] Eventlet Contributors: Monkeypatching the Standard Library, URL: http://eventlet.net/doc/patching.html.

[Fl] Project Floodlight: Floodlight, URL: http://www.projectfloodlight.org/floodlight/.

[Ge] Germann, B.: OFFWall, URL: https://crates.io/crates/offwall.

[He17] Heimgärtner, F.; Schmidt, M.-T.; Morgenstern, D.; Menth, M.: A Software-Defined Firewall Bypass for Congestion Offloading. In: Proceedings of the International Conference on Network and Service Management. CNSM '17, 2017.

[MH15] Miteff, S.; Hazelhurst, S.: NFShunt: A Linux Firewall with OpenFlow-enabled Hardware Bypass. In: IEEE Conference on Network Function Virtualization and Software Defined Network. NFV-SDN '15, 2015.

[Mi] Mininet Team: Mininet, URL: http://mininet.org.

[MK14] Matsakis, N. D.; Klock II, F. S.: The Rust Language. In: Proceedings of the Conference on High Integrity Language Technology. HILT '14, 2014.

[Op] OpenDaylight Project, Inc.: OpenDaylight, URL: https://www.opendaylight.org.

[Op15] Open Networking Foundation: OpenFlow Switch Specification, Version 1.3.5, ONF, 2015.

[Pa14] Pani, A.: Scalable Stateful Firewall Design in OpenFlow based Networks, US8789135, Google Inc., 2014, URL: http://www.google.com/patents/US8789135.

[Ru] Rust Project: Rust, URL: https://www.rust-lang.org.

[Ry] Ryu SDN Framework Community: Ryu, URL: http://osrg.github.io/ryu/.

[Tr] Trema Project: Trema, URL: http://trema.github.io/trema/.