

FIREFLOW— High Performance Hybrid SDN-Firewalls with OpenFlow

Andreas Fiessler and Claas Lorenz
genua GmbH

Research and Project Development
Munich, Germany

Email: {andreas_fiessler, claas_lorenz}@genua.de

Sven Hager and Björn Scheuermann
Humboldt University of Berlin

Computer Engineering Group
Berlin, Germany

Email: {hagersve, scheuermann}@informatik.hu-berlin.de

Abstract—Today, the most widely distributed type of firewalls are software firewalls, running as applications on standard systems. Dedicated networking hardware like SDN switches has also been used to implement firewall functionality, although their native classification capabilities are restricted to relatively simple checks, such as subnet tests. However, native SDN hardware can satisfy high performance requirements that would be challenging for standard software firewalls. This motivates to build a hybrid combination of fast SDN hardware with a standard software firewall. Our approach directly offloads simple rule policies instead of flows to the SDN switch, in order to exploit the limited storage capacity more efficiently. An effective packet diversion algorithm based on header space analysis avoids expensive communication with the back-end. This way, it can achieve the throughput of a native SDN switch while still being able to resort decisions to the full extent of a software firewall. Our evaluation demonstrates a 23-fold classification performance increase over a standard software firewall.

Index Terms—Packet classification, OpenFlow, SDN, Firewall

I. INTRODUCTION

Network firewalls are key security building blocks for IT infrastructure. For larger scale networks, *Software-defined Networking (SDN)* has become a widely adopted concept [1], and accordingly, there are also approaches to implement firewalls based on SDN components [2]. Such firewalls typically are either implemented as *proactive* SDN firewalls, using security policies by statically deployed rules in the SDN switches. This is the predominant mode of implementation in real world controllers. However, the size of the rule memory in SDN switches is fixed and limits the deployment of large policies. In literature, *reactive* SDN firewalls are more prominent (e.g. [3]–[5]). In general, security policies are kept in the controller and a new flow is checked against them. If accepted, suitable flow rules are pushed into the switches.

Practical firewall often require more complex matching capabilities than simple stateless classification based on IP addresses or port numbers, such as payload analysis. This is well supported by existing software firewalls like *iptables*, *pf*, or *ipfw*, which, on the other hand, struggle with current line-speed requirements [6], [7]. A typical dedicated SDN switch hardware, however, cannot be utilized for these complex tasks: albeit the switch hardware is well suited for the quick processing of simple rules based on simple header checks, it lacks the matching capabilities for complex rules.

Such decisions would be sent to the SDN controller, which is not suitable—and not intended—to be used for large amounts of data [8]. There is currently no conceivable development towards SDN switches that would natively implement features comparable with standard software firewalls like [9].

As one way to tackle this limitation with common standard components, we propose FIREFLOW, a hybrid concept that utilizes a standard OpenFlow-based SDN switch in combination with a standard software firewall in order to achieve high throughput and the feature richness of a software firewall, as sketched in Figure 1. Our approach handles simple decisions within the data plane’s scope of the SDN switch with full line speed. FIREFLOW offloads simple policy decisions, rather than flows, to the faster SDN hardware, which is used as an accelerator for a standard software firewall. Note that the FIREFLOW architecture allows the use of a P4-enabled switch or network card as well. Nevertheless, in this work we focus on the integration of existing OpenFlow infrastructure.

One key challenge in such a hybrid design could be identified as efficiently handling the dependencies between simple and complex rules [6] in order to avoid unnecessary diversion while ensuring a correct classification result. An algorithm based on *Header Space Analysis (HSA)* [10] is used to minimize the redundant workload. To achieve an overall high performance, the overlap of rules with regard to the header space must be small—a property that was shown in [6] for both synthetic and real rule sets.

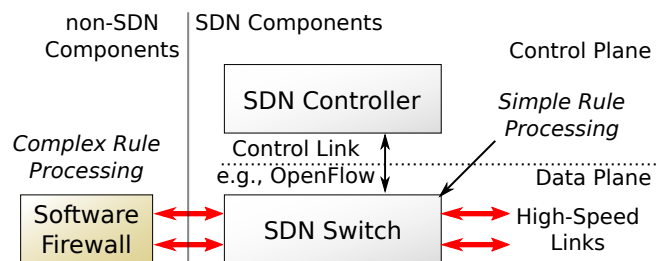


Fig. 1: FIREFLOW setup of an SDN firewall. FIREFLOW offloads the complex rule processing to a software firewall and thus bypasses the software path or removes the bottleneck via the control link.

In summary, this work provides the following contributions: first, we describe the challenges and present solutions for hybrid setups of SDN and standard software firewalls. Second, we show how the approach can be applied as a performance-increasing measure for software firewalls. Third, we demonstrate how existing SDN infrastructure can be upgraded seamlessly into a fully fledged firewall. Finally, we evaluate the feasibility of FIREFLOW and demonstrate an over 23-fold classification speed-up with real-world rule sets compared to a software firewall.

II. PROBLEM STATEMENT

Both firewalls and SDN switches need to tackle an algorithmic challenge called the *packet classification problem* [11]. We define the rule set $\mathcal{R}$ as a prioritized list of n rules with $\mathcal{R} = \langle R_1, \dots, R_n \rangle$, where each rule specifies a sequence of *tests* and an *action*. For a set of d given packet header fields, we regard each rule R_i as a $(d + 2)$ -tuple $R_i = (t_i^1, \dots, t_i^d, c_i, a_i)$, with the t_i^j being *simple tests*, c_i being a potentially empty *complex test*, and a_i being the rule's action. We further define the rule R_i^- as the *simple reduction* of R_i , which only contains its simple checks t_i^j and removes the complex part c_i . We consider a test a simple test if it matches a header field against a port range, a subnet, or a fixed value, otherwise we refer to the test as a complex test. We call a rule R_i a *simple rule*, if $R_i^- = R_i$. Otherwise, we refer to R_i as a *complex rule*. Furthermore, we say that two rules R_i and R_j *overlap* iff $R_i^- \cap R_j^- \neq \emptyset$. All subsequent overlapping rules of a rule R_i are called *HSA-dependent* rules of R_i . The packet classification problem can now be defined as follow: for a given packet p , identify the first rule R_i that matches on p .

III. HYBRID PACKET PROCESSING

FIREFLOW is a hybrid system consisting of a software firewall used for complex firewall tasks and an SDN switch hardware for fast processing of simple classification tasks. Figure 2 depicts the different utilized logical components. A standard software firewall like `iptables`, `pf`, or `ipfw` can be used as the back-end for FIREFLOW.

In the remainder of this paper, we will use the following naming convention for the SDN switch network ports as shown in Figure 2: 1) SDN switch to and from external networks $E2S$ (ingress) and $S2E$ (egress), 2) SDN switch to and from software firewall $S2F$ and $F2S$, 3) SDN switch to SDN controller (both directions) $S2C$.

The key for achieving significant performance gains with FIREFLOW lies in the packet routing decision that enables fast hardware processing. Hence, our goals are keeping as much workload as possible on the fast SDN data plane and reducing the software firewall workload to the necessary minimum. A further goal is limiting the size of the rule set on the SDN switch. In our evaluation setup, the Aruba 3810m switch could hold less than 1,500 rules with the desired expressiveness. Rule set sizes exceeding the limit will be forced to be

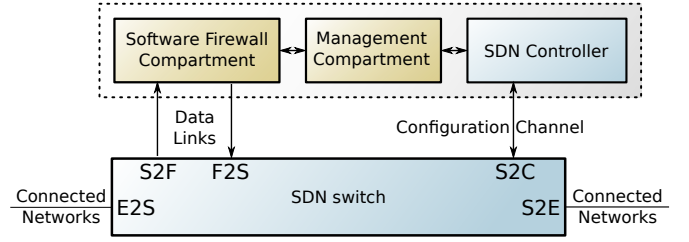


Fig. 2: High-level setup of FIREFLOW.

```
-s 192.0.3.0/24 -m string --string "SELECT"           # RA1
--algo bm -j DROP                                     #
-----
-s 192.0.2.0/24 -m string --string "BAD"              # RA2
--algo bm -j DROP                                     #
-----
-s 192.0.2.0/24 -p tcp --dport 80 -j ACCEPT           # RA3
-----
-s 192.0.4.0/24 -p udp --dport 53 -j ACCEPT           # RA4
```

Listing 1: Rule set $\mathcal{R}_A$.

processed in significantly slower compartments, in the worst case the SDN controller itself.

To demonstrate the operation, we will use an example rule set $\mathcal{R}_A$ shown in Listing 1. For better readability, we omit the chain parameter `-A FORWARD` used in `iptables` in the example. In rules R_{A3} and R_{A4} , only IP addresses, port numbers, and protocol types are examined. As these rules only consist of simple checks, they can be implemented as SDN hardware-only rules on the SDN switch that forwards matching packets from port $E2S$ to port $S2E$.

However, the example also includes two string matching rules R_{A1} and R_{A2} , which is an operation that requires being processed in the software firewall. Therefore, FIREFLOW will place diverting rules to the software firewall for packets matching the stated subnets and software rules including the string matching part in the software firewall. The redirection can be accomplished as follows. A first method, called *rule divert*, places two correspondent SDN hardware rules into the switch in front of rule R_{A3} . This is given by the function `PARTITION_RULE_SET_RULE_DIVERT` in Algorithm 1, which is based on [6] and takes $\mathcal{O}(n^2)$ time. These additional rules correspond to the simple reduction of the complex rules (i.e., R_{A1}^- and R_{A2}^-) and divert all traffic matching the addresses and protocol from switch port $E2S$ to port $S2F$. Hence, the switch will classify the packet against a part of each complex rule that it can handle in hardware, leaving the rest to the software firewall. Two further SDN hardware rules will forward any traffic from the software firewall to the corresponding switch output ports ($F2S$ to $S2E$).

In order to respect all rule dependencies for both methods, an HSA can be used to identify all simple rules that interfere with any of the complex rules not present in the SDN switch. The software firewall must then be configured with either the full rule set, or only complex and HSA-dependent rules. The rule set $\mathcal{R}_A$ further contains a possible interference between rules R_{A2} and R_{A3} . This can occur whenever a packet from

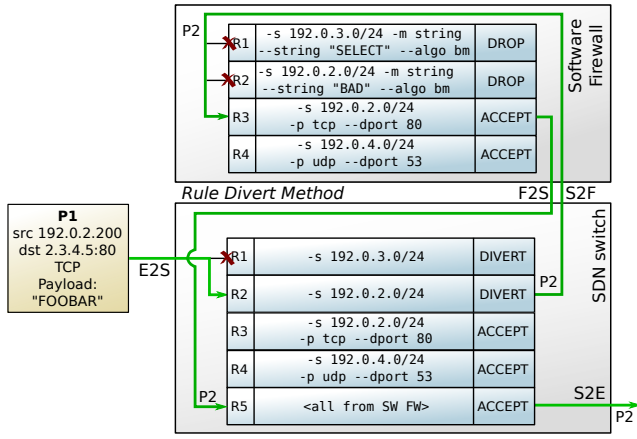


Fig. 3: Packet accepted due to complex software firewall rule.

the source network 192.0.2.0/24 arrives, but does not match the string in R_{A2} in the software firewall. It can still match rule R_{A3} , though. Hence, the software firewall rule set must at least contain the rules R_{A1} , R_{A2} , and R_{A3} .

A second method, called *default divert*, can be used if it is desirable to reduce the number of SDN hardware rules. It is not necessary to insert the simple reduction R^- of all complex rules into the SDN switch. Instead, only the subset of all simple rules can be used, with the HSA-dependent simple rules set to divert. All other divert operations are executed by a default divert rule at the bottom of the switch rule set. This method is given by the function `PARTITION_RULE_SET_DEFAULT_DIVERT` in Algorithm 1, again with time complexity $\mathcal{O}(n^2)$.

Figure 3 illustrates the former example of the rule set in Listing 1 with an exemplary packet, using the rule divert method. The packet is diverted to the software firewall due to the required string matching operation. The packet does not match the string part in software rule R_2 (which corresponds

Algorithm 1 Partition rule set $\mathcal{R}$

```

1: function PARTITION_RULE_SET_RULE_DIVERT(Rule set  $\mathcal{R}$ )
2:   for Rule  $R_i \in \mathcal{R}$  do
3:     if IS_COMPLEX( $R_i$ ) then
4:        $\mathcal{R}_{\text{complex}} \leftarrow \mathcal{R}_{\text{complex}} + [R_i]$ 
5:        $\mathcal{R}_{\text{simple}} \leftarrow \mathcal{R}_{\text{simple}} + [\text{MAKE\_DIVERT\_RULE}(R_i^-)]$ 
6:     else if IS_HSA_DEPENDENT( $R_i$ ,  $\mathcal{R}_{\text{complex}}$ ) then
7:        $\mathcal{R}_{\text{complex}} \leftarrow \mathcal{R}_{\text{complex}} + [R_i]$ 
8:        $\mathcal{R}_{\text{simple}} \leftarrow \mathcal{R}_{\text{simple}} + [R_i]$ 
9:     else
10:       $\mathcal{R}_{\text{simple}} \leftarrow \mathcal{R}_{\text{simple}} + [R_i]$ 
11:   return ( $\mathcal{R}_{\text{simple}}$ ,  $\mathcal{R}_{\text{complex}}$ )

12: function PARTITION_RULE_SET_DEFAULT_DIVERT(Rule set  $\mathcal{R}$ )
13:   for Rule  $R_i \in \mathcal{R}$  do
14:     if IS_COMPLEX( $R_i$ ) then
15:        $\mathcal{R}_{\text{complex}} \leftarrow \mathcal{R}_{\text{complex}} + [R_i]$ 
16:     else if IS_HSA_DEPENDENT( $R_i$ ,  $\mathcal{R}_{\text{complex}}$ ) then
17:        $\mathcal{R}_{\text{complex}} \leftarrow \mathcal{R}_{\text{complex}} + [R_i]$ 
18:        $\mathcal{R}_{\text{simple}} \leftarrow \mathcal{R}_{\text{simple}} + [\text{MAKE\_DIVERT\_RULE}(R_i)]$ 
19:     else
20:        $\mathcal{R}_{\text{simple}} \leftarrow \mathcal{R}_{\text{simple}} + [R_i]$ 
21:   return ( $\mathcal{R}_{\text{simple}}$ ,  $\mathcal{R}_{\text{complex}}$ )

```

to R_{A2}) and must be further analyzed by the following rules. It matches software rule R_3 (corresponding to R_{A3}), redirecting it back to the SDN switch for further transmission by SDN rule R_5 .

IV. EVALUATION

The performance of the hybrid setup is assessed in relation to a standard software firewall. A set of synthetic and real world firewall rule sets provides the basis for our measurements. We compare overall packet classification rates and packet loss due to complex rule processing.

A. Evaluation Setup

We aim at placing a high workload on the classification engine. Therefore, the payload is kept small and packets are sent at the highest rate possible. A setup of two machines is used to generate and count packets using 10 Gbit/s links. A bridging-type firewall setup is realized by placing the SDN switch in between. An additional system is used for the SDN controller, the software firewall and the FIREFLOW management tool. We were using a Ryu 3.3 SDN controller for the Aruba 3810m SDN switch via OpenFlow 1.0.

We generated ten synthetic rule sets of 1,100 rules each with the tool ClassBench [12]. The number of rules was chosen to leave enough distance to the maximum number of rules on the switch, which, when exceeded, results in the automatic incorporation of a slow software-based compartment in the switch. The complex matching part of the utilized rules were set as a custom string match in the packet's payload. Additionally to the synthetic rule sets, three real world rule sets of a customer allowed us to compare the synthetic results to measurements closer to real conditions. The Aruba 3810m switch supports a total number of 10,000 entries in the IPv4 hardware-based routing table. However, with the desired expressiveness that also requires matching port numbers and protocol types, this number drops to less than 1,500 entries—the exact number depends on the rule set's characteristics.

Throughout this evaluation the rule divert method was used. A comparison measurement using the default divert method with this setup showed no notable difference with regard to the classification performance, which was expected due to the $\mathcal{O}(1)$ classification in the switch hardware.

B. Reference Measurement

In order to measure reference performance values, the same firewall tasks that were used to evaluate FIREFLOW are applied to the software firewall as stand-alone setup. The test rule sets are first configured to contain only simple rules. Then, we adapt an increasing number of rules to require a complex string matching operation. For the reference measurement, this only increases the processing complexity for the software firewall. In the following hybrid setup evaluation, however, these rules force a diversion of packets from the hardware accelerator to the software firewall. We also evaluated a native SDN firewall setup using the SDN controller via OpenFlow for complex rule processing, with the expected constraints due to

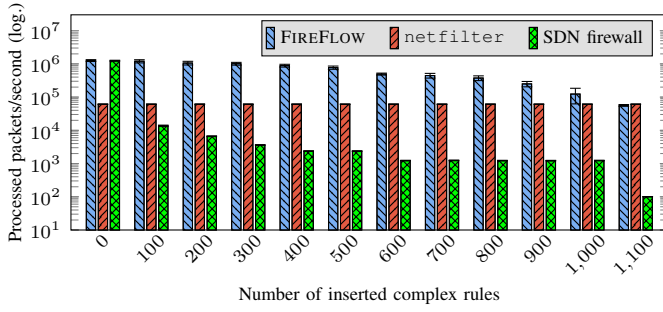


Fig. 4: Throughput comparison of FIREFLOW and stand-alone software firewall (*netfilter*) for varying numbers of complex rules. Bars show averaged throughputs for three real rule sets. Error bars show the respective standard deviations.

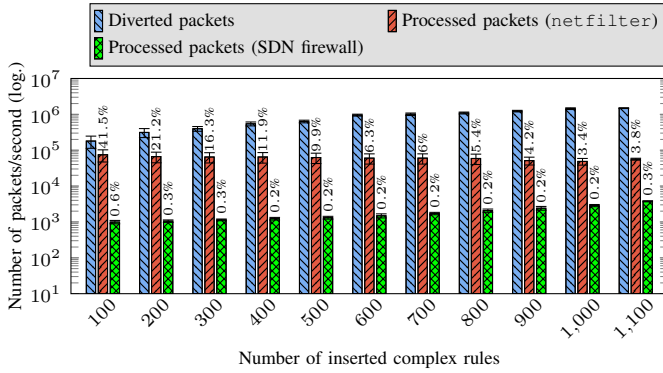


Fig. 5: Avg. number of diverted packets and share of diverted packets that could be processed by the FIREFLOW software firewall compartment compared to an SDN controller and firewall application (*SDN firewall*). Percentages indicate fraction of processed vs. diverted packets, remainder are lost packets. Error bars show the respective standard deviations.

using the control link for data traffic. For each of the rule sets, $k \in \{50, 100, 150, \dots, 1100\}$ rules at equally spaced positions are modified for complex processing and the setup is updated via the management tool. Then, for each k , ten test cycles of 20 seconds are conducted, counting the number of successfully classified and forwarded packets. The number of packets arriving at the ingress port of either setup was about 30 million in 20 seconds.

C. FIREFLOW Performance

The evaluation of the hybrid system follows the same procedure as the reference measurement. The software workload for simple rules is expected to be reduced and instead of using the SDN control channel, complex decisions are diverted to the software firewall compartment.

In Figure 4, one can see an almost constant number of packets processed in the reference measurement, where all packets are solely classified by the *netfilter* software firewall. The increasing number of complex rules only has a minor effect on these numbers. This shows that the limitation of the software firewall arises from the software processing overhead

rather than from actual rule complexities. FIREFLOW can improve the performance especially when many simple rules are offloaded to the SDN switch, where it reaches a 23-fold performance gain. As expected, the second reference using a native SDN firewall setup suffers from the slow control channel as soon as complex rules were inserted. In comparison, FIREFLOW processed about 500 times more packets.

Another interesting parameter is the number of packets that are lost due to the diversion operation to the slower software firewall or the SDN controller, respectively. Figure 5 shows a comparison between the number of packets that are diverted and those that are successfully processed by the back-end (either the software firewall or the SDN application). Even with few complex rules, most of the packets sent to the SDN application are lost, which means connections targeting complex rules will suffer immense packet loss. The software firewall back-end of FIREFLOW can sustain the workload better, especially if only few complex rules are present.

V. CONCLUSION

The FIREFLOW approach was introduced to set up a fast and flexible hybrid SDN firewall. The concept primarily relies on the fast SDN data plane to process the majority of the traffic. Only decisions too complex to be processed within the SDN switch are diverted to a software firewall within the context of the firewall's data plane. Only standard components are used—in our example setup a commodity SDN switch and a software firewall based on *netfilter/iptables*. We show that FIREFLOW can achieve an over 23-fold classification performance gain compared to standard software firewalls.

REFERENCES

- [1] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, 2015.
- [2] "Floodlight - Firewall REST API," <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343614/Firewall+REST+API>, last access: Jul 06, 2018.
- [3] J. Pena and W. Yu, "Development of a Distributed Firewall Using Software Defined Networking Technology," in *ICIST '14*, Apr. 2014.
- [4] J. Collings and J. Liu, "An OpenFlow-Based Prototype of SDN-Oriented Stateful Hardware Firewalls," in *ICNP '14*, Oct. 2014.
- [5] H. Hu, W. Han, G. Ahn, and Z. Zhao, "FlowGuard: Building Robust Firewalls for Software-defined Networks," in *HotSDN '14*, ser. HotSDN '14, 2014.
- [6] A. Fiessler, C. Lorenz, S. Hager, B. Scheuermann, and A. Moore, "Hy-PaFilter+: Enhanced Hybrid Packet Filtering Using Hardware Assisted Classification and Header Space Analysis," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, Dec. 2017.
- [7] W. Jiang and V. Prasanna, "Large-scale Wire-speed Packet Classification on FPGAs," in *FPGA '09*, Feb. 2009.
- [8] R. Durner and W. Kellerer, "The cost of Security in the SDN control Plane," in *CoNEXT Student Workshop '15*, Dec. 2015.
- [9] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, Feb. 2014.
- [10] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," in *NSDI '12*, Apr. 2012.
- [11] F. Baboescu and G. Varghese, "Scalable Packet Classification," in *SIGCOMM '01*, Aug. 2001.
- [12] D. Taylor and J. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, Jun. 2007.