Network Security through Software Defined Networking: a Survey

Jérôme François, Lautaro Dolberg, Olivier Festor, Thomas Engel

Network Security through Software Defined Networking: a Survey

Jérôme François[*]
INRIA Nancy Grand Est
SnT - University of Luxembourg
jerome.francois@inria.fr

Lautaro Dolberg
SnT - University of Luxembourg
lautaro.dolberg@uni.lu

Olivier Festor
INRIA Nancy Grand Est
olivier.festor@inria.fr

Thomas Engel
SnT - University of Luxembourg
thomas.engel@uni.lu

ABSTRACT

Network security is a predominant topic both in academia and industry. Many methods and tools have been proposed but the attackers are still able to launch massive and effective attacks. Keeping the pace with the new threats appearing or becoming more sophisticated everyday is of a paramount of importance. Software Defined Networking (SDN) has recently emerged and promotes the programmability of the networks, which thus allows to enable in-network security functions. This includes firewalls, monitoring applications or middlebox support through OpenFlow devices. Therefore, this paper reviews the related approaches which have been proposed by identifying their scope, their practicability, their advantages and their drawbacks.

Keywords

SDN, OpenFlow, Security, Firewall, Monitoring, Anomaly detection

1. INTRODUCTION

Network security is a vital task for today's network because the volume of attacks is continuously increasing over the years and they become to be more sophisticated and/or distributed [12]. The threats are diverse and can target numerous services (email, remote shells, social networks, banking services, P2P, video streaming, gaming...) on heterogeneous devices (computers, mobile phones, industrial systems, webcam, TV screens, cars or any other connected things). Therefore, these problems have been investigated since decades. Recently, Software Defined Networking (SDN) emerged and rapidly gained a lot of interests by empowering the programmability of networks and so to facilitate the deployment of new services and functionalities in the networks. This paper reviews how SDN can play a major role in implementing the network security functions like access control, network monitoring or middlebox deployment. In particular, most of these approaches rely on OpenFlow which obtains a strong interest from the community after being initially proposed in 2008 [9].

For the sake of clarity, Section 2 introduces the necessary background about SDN and OpenFlow. Section 3 focuses on enabling simple access policies (firewall) with SDN whereas Section 4 is dedicated to the network monitoring, which is an essential prerequisite for anomaly detection. More advanced techniques are detailed in Section 5. A final discussion concludes the paper in Section 6.

2. BACKGROUND

2.1 Software Defined Networking

The core concept behind SDN is to decouple the data and control plane. In brief, the switches and routers are only acting as forwarding devices while the decisions about how to forward the packets rely on a separate controller. In this paradigm, switches[1] are cheaper forwarding devices, while algorithmic decisions take place in more complex and resourceful devices (naturally, more expensive than switches). As a direct advantage, the network infrastructure costs could be drastically reduced as the controller may be a commodity computer or can be located in a cloud infrastructure and so use resources on demand only. Actually, the underlying objective is not to run standard routing algorithms in a central controller but to enable new protocols, services and algorithms. For example, at layer 2, the spanning tree protocol (SPT) [15] is suboptimal as every single path between two hosts is based on the global logical tree built by SPT even if a physically shorter path exists but has been discarded to avoid routing loops. Hence, SDN could program the network to use this shortest path only for these two hosts without creating any loops for the others. Other applications include

[*] J. François is currently full time researcher at INRIA and research fellow at the university of Luxembourg

[1] The terms switch and router can be interchanged in the context of SDN

Textual description	Ingress port	Mac Src Addr	Mac Dst Addr	Ip Src Address	Ip Dst Address	Protocol	Src port	Dst port	Instructions
Switching	*	*	AB:CD...:22	*	*	*	*	*	Forward to port 3
Routing	*	*	*	*	1.2.3.*	*	*	*	Set Mac src addr=AB:CD:EF:00:11:33, Mac dst addr = AB:CD:EF:00:11:44, forward to port 5
Firewall (deny incoming web connections)	1	*	*	*	1.2.3.*	TCP	*	22	Drop
Proxy	*	*	*	*	2.3.4.5	TCP	*	80	Set IP addr=10.11.12.13, forward to port 5
Load balancing	1	*	*	*	2.3.4.5	TCP	*	80	set dst addr = 2.3.4.6, Forward to port 4
	2	*	*	*	2.3.4.5	TCP	*	80	set dst addr = 2.3.4.7, Forward to port 6

Table 1: Examples of OpenFLow flow table entries

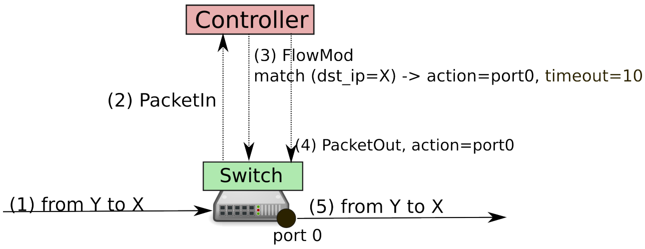


Figure 1: OpenFlow packet processing

quality of service or even security, which is the focus of this paper.

2.2 OpenFlow

OpenFlow has been introduced in [9] and tends to be the *de facto* standard for SDN. Its development is led by the Open Networking Foundation which regularly extends its specification [2]. With OpenFlow, every switch communicates to a controller in order to install, delete or modify flow-based rules to process incoming packets. Examples of controllers are NOX, POX [3] or Floodlight among others [4].

Installing a rule creates an entry in the switch flow table. Each entry is composed of two main components:

- *match fields*: a filter to match packet headers like Ethernet addresses, TCP ports, IP addresses, Time-to-Live value...
- *instructions*: a definition on how to handle an incoming packet matching the rule. It is composed of a set of actions to send the packet towards a single or multiple ports, to drop the packets or to apply some header modifications. In addition, the flows entries maintain counters indicating: number of matched packets, number of bytes, number of errors... Although many counters are proposed in the documentation, most of them are optional.

When a packet arrives at a switch, if it matches a rule, based on the *match fields* (the highest priority rule is selected if multiple of them match the packet), the instruction set is then executed. Otherwise, there is a *table-miss* which specifies a default action to perform. Figure 1 illustrates a standard action for *table-miss* which corresponds to send the packet to the controller (*PacketIn*). The controller is in charge of making the decisions and so can install a specific rule (*FlowMod*) for handling corresponding packets (for example to forward every packet from Y to X) as well as passing back the packet to the switch by specifying the action to do (*PacketOut*). In this example, the message is forwarded through the switch port 0. *FlowMod* and *PacketOut* can be combined or used individually. Indeed, a controller can install a rule in a proactive way, without receiving a *PacketIn* message. Each rule is associated to a hard and soft timeout. The hard timeout is the maximal duration of a rule while the soft timeout is the maximal interval between two packets matching the rule. When one of them is exceeded, the rule is removed. Both can be set to an infinite time.

OpenFlow provides also other features like multiple flow tables, flooding a packet, removing rule, or group table but this Section is limited to the necessary background to ease the reading of this paper and additional information is provided in following Sections when needed. Some simplified flow table entries are given in table 1.

3. FIREWALLS

Firewalls are essential components that act as a first level of access control. A firewall protects a local network from other hosts in Internet, which are not trustworthy. Packet filtering can be done at the different levels of the network stack. Most of them are able to analyze packet headers up to the transport layer such as Netfilter[5] but there exist also application level firewalls. They are usually dedicated to one application protocol, like SIP [8] or Web like NAXSI[6]. They rely on the knowledge of the protocol specification, the message format and syntax. While there are growing discussions around integrating layers 4-7 in OpenFlow, there is still no consensus. Besides, the specification is enough flexible to match any other header assuming it is implemented at the controller and switch sides, which thus is contradictory with OpenFlow's paradigm to use a standard protocol controlling any switch of the network, independently from a vendor or a model implementation. There are very few switches sup-

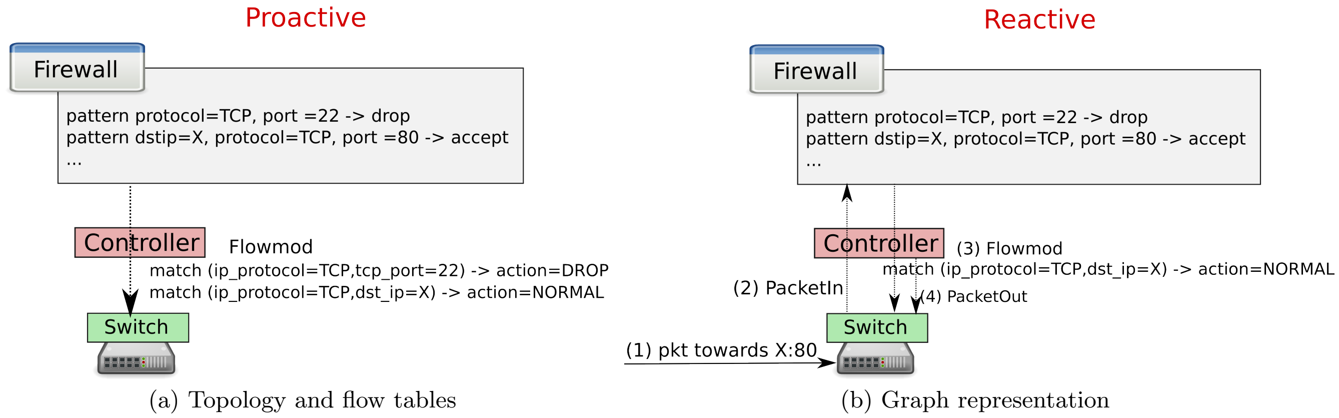
[2] `https://www.opennetworking.org/sdn-resources/onf-specifications/openflow`, accessed on 2014/06/26

[3] `http://www.noxrepo.org/`, accessed on 2014/06/26

[4] `http://www.projectfloodlight.org/`,accessed on 2014/06/26

[5] http://netfilter.org/

[6] `https://www.owasp.org/index.php/OWASP_NAXSI_Project`, accessed on 06/26/14


Proactive
Firewall
pattern protocol=TCP, port =22 -> drop
pattern dstip=X, protocol=TCP, port =80 -> accept
...
Controller
Flowmod
match (ip_protocol=TCP,tcp_port=22) -> action=DROP
match (ip_protocol=TCP,dst_ip=X) -> action=NORMAL
Switch
Reactive
Firewall
pattern protocol=TCP, port =22 -> drop
pattern dstip=X, protocol=TCP, port =80 -> accept
...
Controller
(3) Flowmod
match (ip_protocol=TCP,dst_ip=X) -> action=NORMAL
(2) PacketIn
(4) PacketOut
Switch
(1) pkt towards X:80

(a) Topology and flow tables

(b) Graph representation

Figure 2: OpenFlow-based stateless firewalls

porting layer 7 as for example VortiQa[7].

As a conclusion, achieving traffic filtering above layer 4 can be enabled by OpenFlow by inspecting the packets at the controller side assuming that all incoming packets are forwarded to it with *PacketIn* messages (no rule installed). On one hand, this clearly increases the latency and is thus only viable by limiting the analysis to a few packets (like the first ones of a flow). On the other hand, the inspection on highest layers in switches is also contradictory with the SDN paradigm which aims at keeping switches as forwarding devices only because other processes are resource consuming.

Based on this analysis, the rest of this Section reviews the major firewall use cases which might be supported by OpenFlow, *i.e.* when considering up to the layer 4 (transport), but Section 5 is dedicated to advanced analysis including layer 7 inspection.

3.1 Stateless firewall

A stateless firewall filters the packets, usually accept or drop, based on the values in the headers like the IP address or the port numbers. It does not track the status of a connection to check the legitimacy of a packet. For example, initializing a TCP connection is based on the 3-way handshake (SYN, SYN-ACK, ACK) which thus has to precede any data exchange. A stateless firewall is unable to do such a verification since the decision process solely relies on a single packet without considering any previous history. As already illustrated in table 1, such a firewall might be easily realized with OpenFlow. For example, blocking all connections to SSH (TCP port 22) is done by defining a rule specified as:

```
match fields:
  IP protocol = TCP
  TCP destination port = 22
actions:
  DROP
```

Figure 2 highlights two modes to install the firewall rules on the switches: in a proactive or reactive way. We assume two rules: one to disallow SSH connections on port 22 like previously mentioned and one to allow the connections to a web server on port 80. In a proactive way (Figure 2(a)), the controller installs the rules without being requested by the switch. As this kind of firewall rules are usually applicable for a long period, the timeout value should be high and renewed if necessary. Else the rule can be set as permanent (with 0 as timeout value) and removed when needed.

The controller can also install the rules in a reactive way in Figure 2(b). For example, when the first packet to the port 80 comes in (1), a *PacketIn* message is sent to the controller (2), which installs the rule (3) and requests the switch to emit the packet (4).

The main advantage of the proactive approach is to avoid additional delays (switch-controller communication) when the first packet arrives whereas the main advantage of the reactive approach is to keep small flow tables without filling it with unmatched rules. Hence, the choice of the implementation should be dependent on the firewall configuration, in particular the number of rules and the the traffic profile. It is important to note that, in a reactive way, the controller may have to translate the firewall configuration into OpenFlow rules at high pace if there are traffic bursts, which again introduce large delays.

In [19], the authors propose a language to define firewall rules relying on the POX controller. The rules are installed in a reactive way. The evaluation shows that the implemented functionalities work, allowing or blocking traffic, but does not assess any performance metric, for instance about underlying introduced delays.

The Floodlight project has led to the development of an OpenFlow controller (as introduced in Section 2.2) as well as applications including a stateless reactive firewall. It provides a REST API and the rules are defined by a set of matching fields, where the main ones are the switch id, the Ethernet addresses, the IP addresses and the ports, and only two actions are possible: DENY or ACCEPT (default).

3.2 Stateful firewall

[7] `http://www.freescale.com/webapp/sps/site/homepage.jsp?code=VORTIQA`, accessed on 06/26/2014

In case of a TCP-based stateful firewall, only incoming packets part of a flow which has been initiated by a machine of the local network are usually allowed. Assuming a switch connected to Internet through port 0 and to local machines on the other ports, the initial rule aims at blocking the incoming connections by default and is installed pro-actively:

```
match fields:
  input port = 0
priority:
  0
actions:
  DROP
```

When a local machine (IP address: A.A.A.A on port X) connects to B.B.B.B in Internet, the switch emits a *PacketIn* to the controller which then installs the rules to allow such a connection as well as the reverse traffic:

```
match fields:
  source IP address: A.A.A.A
  destination IP address: B.B.B.B
actions:
  port 0
idle timeout:
  5s

match fields:
  source IP address: B.B.B.B
  destination IP address: A.A.A.A
actions:
  port X
idle timeout:
  10s
priority:
1
```

These examples include only necessary information for the sake of clarity. The priority is only illustrated for the 2 rules matching incoming packets from outside. A higher priority is assigned to the rule for reverse traffic but has to be limited in time by setting a short timeout value. Normally, the end of a TCP connection can be observed as well (FIN flag) but this would require to inspect every individual packet of the flow by the controller which is not viable at a large scale. Therefore, the rule has to be reactivated if the time between two consecutive packets is longer than the timeout. Usually, TCP connections can remain open by emitting keep-alive messages. In the worst case, the A.A.A.A would request B.B.B.B to send again the last packets if not received, which leads the rule to become active again.

Even if it is possible to mimic the behavior of a stateful firewall, OpenFlow does not provide a full support for such a task (up to to the ending of the connection), except assuming a middlebox (section 5).

4. NETWORK MONITORING

Network monitoring is very helpful for security purposes. Actually, anomalies can be observed by collecting statistical data about network traffic. For example, a high increase in the number of flows towards a specific port is significant of a scanning activity or numerous flows from multiple sources towards a single destination can reveal a DDoS (Distributed Denial-of-Service).

Other packet features might be necessary like TCP flags. For example, a disproportionate number of packets with the SYN flag set is a signature of a SYN flooding attack [14]. However, these metrics are not only dedicated to security and can be used for other applications, in particular to evaluate the usage of a network and its status. Hence, this section reviews the existing approaches to retrieve traffic statistics with OpenFlow, which might be used in other domain applications than security as well. In fact, such a task naturally relies on *counters* which are provided by OpenFlow. They are applied to different structures (flow table, flow table entry, switch port) depending on their nature (received/transmitted packets, received/transmitted bytes, duration, received errors...) and are not all mandatory. Therefore, for monitoring a particular traffic flow, a rule which matches it (*match fields*) is installed on a switch and statistics are then collected. While this is the mainly used approach (passive mode), an active approach can leverage the capability of the controller to send a forged packet (*PacketOut*) to create an artificial traffic matching a pre-installed rule in order to retrieve information about the latency, the bandwidth or the number of dropped packets.

Research on network monitoring also explored sketch-based approaches. The sketches are usually hash-based data structures which are optimized to approximate metrics on multiple aggregated features, like counting flows per IP address or per port, without analyzing individual flow. Whereas those are beneficial for security monitoring, they necessitate a major shift from flow-based to sketch-based switch implementation. Since the primary objective of SDN is to apply forwarding policies, major approaches like OpenFlow rely naturally on the flows. Therefore this section considers only flow-based monitoring but recent proposals using sketches for SDN are existing [11, 23].

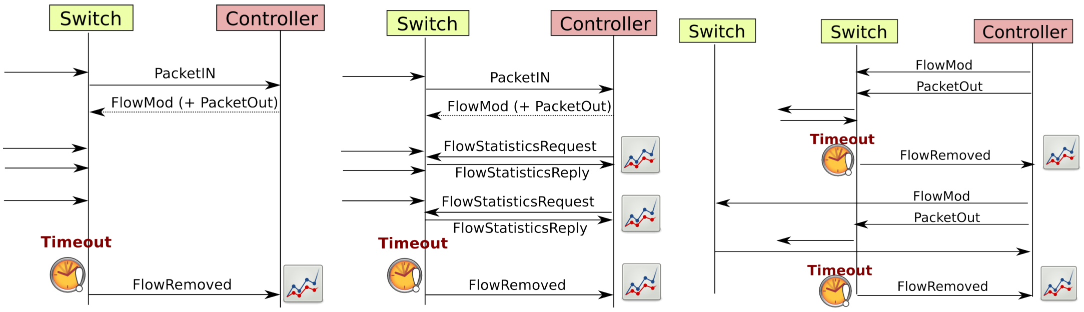
4.1 Passive monitoring

Passive monitoring consists in retrieving information about the flows from the switch to the controller. It can be done using:

- Push mode: The switch sends information to the controller. This is the standard behavior when a flow expires (*FlowRemoved* message),
- Pull mode: The controller requests the switch to get information.

Both mechanisms are illustrated in Figures 3(a) and 3(b). The main advantage of the push mode is avoiding an overload on the switches and the network by the usage of the *FlowStatisticsRequest* and *FlowStatisticsReply* messages. However, the monitoring application needs thus to wait the end of a flow for measuring it. This induces a delay in analyzing flow information and potentially in detecting abnormal behavior. While the pull mode is more resource consuming, the monitoring is more fine-grained and allows to monitor how the behavior (traffic pattern) may change along time by periodic polling.

FlowSense [22] argues for a zero cost approach using the push mode. The objective is to infer the link utilization in a network which corresponds to consider all active flows on this link, *i.e.* which match a rule whose the action sent the outgoing packet on this link. However, such a computation



(a) Passive-Push mode (b) Passive-Pull mode (c) Active mode

Figure 3: OpenFlow-based monitoring

is not so easy assuming a push-mode based monitoring as a link might be used by several flows which do not start and end at the same time. Therefore, FlowSense creates a checkpoint at the arrival of the first (*PacketIn*) and the last packet (*FlowRemoved* excluding the timeout) of each flow. Then, for each interval between two checkpoints $t1$ and $t2$, FlowSense computes the link utilization of those flows which have been active during this interval but now ended (as flow statistic are collected at the end of the flows). Since these flows might have been also active before $t1$, the participation of each flow to the link utilization is averaged regarding their entire duration. Therefore, this introduces some approximation by considering that the traffic is regular over time without peak or drop. In addition, a major issue highlighted by the authors is that FlowSense must wait for all flows completion to compute the link utilization, (as traffic statistics are collected at the end of a flow). This might lead to large delays for computing a link utilization.

OpenTM [20] focuses on traffic matrix by estimating the volume of traffic (bytes or packets) between every Origin-Destination (OD) pairs. Although this information can be easily accessed by requesting statistics on switches, assuming that proper rules have been installed prior, the main research questions of this paper are the selection of switches to query and the time interval between two requests in order to find a good trade-off between the accuracy and the maximum load on the switches. For instance, the last switch on a path of a flow is the most representative of the receiver because loss of packets can occur beforehand. However, such a strategy leads to a large load on this switch. Therefore, other strategies like uniform random choice or least load switch (worst strategy in terms of accuracy) have been investigated. The experiments highlight a small degradation of the accuracy (less than 3%)[20].

Unlike OpenTM, Payless[4] argues for an adaptive query rate which is defined through a timeout which is the time between two statistics requests for a unique flow. When the controller is getting back information about a flow through a *FlowStatisticsReply* message, the difference between the current and previous measures is calculated (difference in byte counts). If this difference is higher than a threshold, the timeout is reduced (divided by a tunable factor) and increased otherwise (multiplied by a tunable factor). The timeout value is also bounded to avoid too large or small values. Thanks to this process, a higher attention is given to big flows which weight more than small flows when evaluating the link utilization. In addition, multiple requests are grouped together (flows with a same timeout value) in order to limit the network overhead. It has been observed that Payless is able to estimate the link utilization with a small error, compared to a frequent periodic polling strategy, but saving up to 50% of messages. Besides, Payless proposes a global framework for SDN-based monitoring including a request interpreter to create high level monitoring policies, for example by user. An adaptive monitoring method is also described in [24] and is close to [4] but is applied to anomaly detection in mobile networks. The author uses a prediction model to compare how an indicator, like a counter, is evolving compared to a prediction based on historical values. Based on the result of the comparison, the granularity of the monitoring is adjusted both in space and time. OpenNetMon[21] is complementary to the previous approaches, especially by computing the packet loss, which might be representative of a collateral effect of an attack, like a Denial-of-Service attack. This is achieved by counting the number of packets at the entry and exit switch of a flow path.

Flexam [18] proposes, as an extension for OpenFlow, an additional action to sample packets on the data path and send the samples to the controller for advanced security analysis. While this is an interesting approach, it still requires a careful evaluation overhead (sampling process at the switch, number of sampled packets) before being potentially adopted in a SDN framework. Avant-Guard [17] adopts another angle for enabling powerful monitoring in SDN by adding functionalities on top of switches, which is a bit contradictory with the concepts of SDN. In fact, switches are in charge to filter only TCP connections which are successfully initialized before interacting with the controller to know how to handle it. In addition, a fine-grained monitoring is proposed by extending the rule definition with conditions (for example, only if the packet rate is higher than a threshold) to activate another rule or to notify the controller with or without including the payload of the packets.

4.2 Active monitoring

Active monitoring with OpenFlow relies on the controller's capacity to request a switch to send a packet through the *PacketOut* message. Figure 3(c) shows the necessity to install the rules to forward the probing packets in the network. Installing the rules in a reactive way would lead to more messages between the switch and the controller and additional delays as explained in section 4.1 whereas the controller, as the initiator of the packets, has the necessary knowledge to prepare the specific rules in advance. That is why it is recommended to configure the switches in a proactive way.

In [16], an active method is proposed for estimating the path delay by injecting packets at the first switch on the path and getting back it at the last switch. In fact, such a switch may use the default rule to send a *PacketIn* message to the controller. The delay estimation then takes in account the delay between the controller and switch, estimated as the half of the round trip time. Using such an approach, the probing packets can be specifically crafted as minimal as possible to reduce the bandwidth use by 81% compared to an ICMP ping (echo-reply). As shown by the simulation-based experiments, there is a slight offset between the estimated value and the real one. This can be due to the latency of running processes at the switch and controller side in addition to the network latency. Although such a deviation can be corrected assuming a calibration process in [16], the authors of OpenNetMon[21] show that using the control plane and a *PaketOut* message implies additional and variable delay because it has to go through the software scheduler of the switch. This yields to very inaccurate results unlike in [16]. These different conclusions might be explained by distinct experience configuration and environment: either based on simulation [16] or an a real testbed [21]. In the latter case, the solution consists in injecting packet from the controller to the first switch on a path through a dedicated VLAN without relying on the control plane. The promising results show a small deviation of around 0.15ms on a small testbed with four switches.

5. ADVANCED ANALYSIS

The works described in section 4 are dedicated to statistics which can be leveraged to detect anomalies like heavy hitters. However, the security often need advanced processes deeply investigated in the past. Advanced packet analysis including Deep Packet Inspection (DPI) would require to forward each incoming packet to the controller (*PacketIn*) entailing a large overhead on the controller and on the links between the switches and the controller. Therefore, it is unfeasible in practice. In [10], the authors adapt usual anomaly detection techniques to be enabled by OpenFlow. For example, to detect TCP scans, it is necessary to track the initialization of the connections when they are successful (SYN, SYNACK) or not (RST or timeout). Only these messages are forwarded to the controller (*PacketIn*), which is responsible for applying detection algorithms (abnormal number of unsuccessful connection attempts or large number of contacted IP addresses) before installing long-term rules to forward the flows if the connection is flagged as normal. The authors also combine such techniques (tracking the first packets) and passive monitoring in pull mode to build an entropy-based detector. Finally, they leverage a rule-based filtering method to inspect the first bytes of a packet. For instance, only the 100 first bytes of a TCP connections are relevant. Hence, once the corresponding packets are sent to the controller (*PacketIn*), the controller installs a rule to forward directly the remaining ones (*FlowMod*).

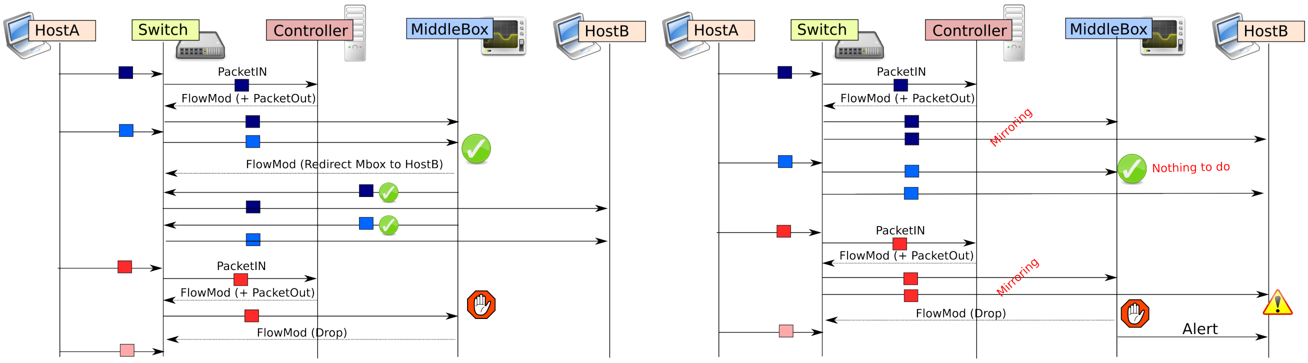
Therefore, a careful and reduced selection of packets may support an analysis where the controller acts as the main interface between the network and the security applications. Nowadays, there are also many security middleboxes to enforce security, especially deep packet inspection modules, to check if the content of a flow contains a malware signature, to detect spam, to analyze any application protocol like SIP [1], which can rely on hardware-based fast processing [13]. This needs to redirect the traffic towards the dedicated middleboxes. Although the controller could again play the role of the interface between the network and the middleboxes, it is clearly a bottleneck. Therefore, a much efficient approach would consider to install a rule (*FlowMod*) to redirect the traffic towards the middleboxes directly. For example, all incoming emails can be filtered by installing a rule matching the mail server address and whose the action is to redirect the packets to a Spam detector. Actually, two modes described in [3] are highlighted in Figure 4:

- Interception: the packets are forwarded exclusively to the middlebox. The latter will then send back them to the switch if no suspect content has been observed. As highlighted in Figure 4(a), the controller plays an important role since it is in charge of installing a rule to forward in the network the flows validated by the middlebox. The validation checking can be done in many ways. For example, it may simply consist in verifying if a packet comes from the middlebox by matching fields (Ethernet source address, dedicated switch port...). Such a way is fully compatible with OpenFlow as there is no need for any additional mechanism. If the middlebox considers the traffic as malicious, it can interact with the controller to install a drop rule.

- Mirroring: the packets are forwarded to the middlebox and the final destination in parallel. This is feasible with OpenFlow as multiple actions can be defined for a flow. In such a case, if the traffic is not considered as suspect, no specific rule has to be installed as it already exists (the controller might be requested to disable the mirroring to the middlebox). However, when a suspect behavior is detected, a drop rule has to be set as shown in Figure 4(b).

The interception induces long delays due to the middlebox analysis. Its usage can be questioned for real time application like streaming but is encouraged for others like email services. A major concern is that several packets might be accumulated at the middlebox before making a decision as shown in Figure 4(a). In such a case, the connection oriented protocols and applications would fail as the destination node is not responding anymore unlike with mirroring. Hence, a smart middlebox has to interact in a similar way than the final destination node (like a sandbox) before delivering the packets to it. On the contrary, the risk of the mirroring approach is to block the malicious traffic too late, which may thus need additional tasks to verify its impact and to apply counter-measures like malware disinfection. As a conclusion, the choice between the interception and the mirroring mode depends on the type of the application.

In [3], the authors describe a SDN-enabled forensic system for data centers by deploying a set of PVPs (Provenance Verification Points), middleboxes which have to verify the identity of the message senders and receivers and to store



(a) Interception

(b) Mirroring

Figure 4: SDN support for Middlebox-based analysis

them for further forensic analysis when requested. In such a scenario, OpenFlow switches are configured to mirror the traffic to the PVPs. OpenSafe [2] proposes a monitoring framework to filter and redirect traffic to counters and sinks. It includes a policy language and relies on OpenFlow to filter the traffic (by IP address, by port). In order to improve scalability, multiple schemes to share the load and so the traffic on multiple sinks are proposed like round robin or random. However, as highlighted by the authors, OpenFlow switches do not have the ability to achieve that since all actions of a rule are always applied. Therefore, the controller has to be always requested to decide on the final destination of a flow, which thus induces an overhead.

Although the previous approaches aim at forward the traffic to middleboxes, the middleboxes may modify packets, as for example application proxies. Hence, a SDN-controller could not be able to track an individual flow from the first to the last switch. Assuming that the goal is to observe an attack path from a known blacklisted IP address, if it goes through a proxy, the source address might be rewritten and so the flow cannot be followed until its final destination by the SDN controller, which is not aware of the middlebox process. Therefore, the middleboxes and the SDN controller have to cooperate together. This is why FlowTags [5] extends the OpenFlow architecture where each middlebox has to mark the packets. The experiments in the paper use the ToS/DSCP field of the IPv4 header to keep track of the sources of the messages. To have a coherent flow tracking over a network, FlowTags is integrated within OpenFlow by enabling communication between the middlebox and the controller, similar to the switch-controller communication.

Finally, Jafarian *et al.* [6] also leverage the ability of OpenFlow for dynamically rerouting traffic. However, the objective is not to forward the packets towards a specific location for being analyzed but to frequently change the location of a host in the IP address space (Moving Target Defense). Actually, an attack is usually launched after a first stage of probing in order to discover reachable hosts and services. Therefore, by changing the IP address of a host between these two stages (probing and attack), the attack would fail. To achieve that, each real IP address is assigned to a virtual one, which is frequently changed. The authors have implemented their proposal in a NOX controller which is responsible of keeping track of the matching between the virtual and real IP addresses. It has to install the rules to maintain previously established connections and dynamically modifying the IP addresses. The results show that during a worm propagation, such an approach can prevent up to 90% of the hosts in a class B network to be infected.

6. CONCLUSION

Using network programmability, the security applications can dynamically apply changes to the network configuration to rapidly adapt security policies, trigger counter-measures or request further analysis from a dedicated middlebox. The researchers have focused on OpenFlow as one of the most generally adopted standard for SDN. Besides, increasing security is necessary nowadays and it is legitimate to investigate new technologies for helping in such a task. However, OpenFlow was not designed for security and so cannot fully support easily all these functions, especially when it is necessary to track the full history of a flow (*e.g.* stateful firewall).

Firstly, our comparisons are based on qualitative aspects because quantitative comparisons are not possible and would not be fair. Indeed, most of the methods do no provide usable source code or datasets. Actually, there is a lack of common datasets as this would require too many features: the topology, the traffic flows, the SDN policies (*i.e.* the OpenFlow rules installed on switches) and eventually labeled attacks. Accessing such data is highly difficult which leads to simulation-based or small testbed-based evaluations. The evaluations usually consider small topologies (like a line of few switches), artificial traffic (iperf – iperf.fr) generated traffic, injected delays or loss of packets...). Therefore, even similar methods may lead to different conclusions as highlighted in the case of latency monitoring in Section 4.

Secondly, using SDN for security may tend to add additional tasks to these switches which is slightly contradictory with the paradigm. For example, while OpenFlow define counters, observing anomalies might lead to create numerous rules, for example one per each origin-destination pair unlike the forwarding policies which are prone to larger aggregates. Such an issue is addressed in [7] by dynamically modifying the granularity of flow rules to limit their number but it is only useful to monitor large aggregates whereas a stealthy attack is not necessarily voluminous.

As a final conclusion, OpenFlow and generally SDN are good enablers for network security but a careful design and evaluation for avoiding too much collateral overhead are required. Advanced security analysis cannot clearly rely solely on these technologies but may benefit from their support for highly adaptive mechanisms (middlebox mirroring, counter-measures deployment, source attack isolation...). It has to be carefully considered, especially regarding the induced overhead and the scalability, which are currently few considered. This can be partially explained by the lack of datasets as explained previously, which thus limits the assessment of an approach in a realistic environment.

Acknowledgement: this work was partially funded by MOVE and IDSECOM, two projects funded by the *Fonds National de la Recherche* in Luxembourg. **Note:** figures include wikimedia contents (visit http://commons.wikimedia.org for license information).

7. REFERENCES

[1] H. Abdelnur, R. State, and O. Festor. Advanced Network Fingerprinting. In *Recent Advances in Intrusion Detection – RAID*. Springer, 2008.

[2] J. R. Ballard, I. Rae, and A. Akella. Extensible and scalable network monitoring using opensafe. In *Internet Network Management Conference on Research on Enterprise Networking*, Berkeley, CA, USA, 2010. USENIX.

[3] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou. Let SDN be your eyes: Secure forensics in data center networks. In *Proceedings of the NDSS Workshop on Security of Emerging Network Technologies (SENT'14)*, Feb. 2014.

[4] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS)*, 2014.

[5] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *SIGCOMM Workshop on Hot Topics in Software Defined Networking - HotSDN*. ACM, 2013.

[6] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. In *Hot Topics in Software Defined Networks - HotSDN*. ACM, 2012.

[7] L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In *Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services – Hot-ICE*. USENIX, 2011.

[8] A. Lahmadi and O. Festor. Secsip: A stateful firewall for sip-based networks. In *International Conference on Symposium on Integrated Network Management - IM*. IEEE, 2009.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[10] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting traffic anomaly detection using software defined networking. In *Recent Advances in Intrusion Detection – RAID*. Springer, 2011.

[11] M. Moshref, M. Yu, and R. Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *SIGCOMM Workshop on Hot Topics in Software Defined Networking*. ACM, 2013.

[12] A. networks. Worldwide infrastructure security report (2013 report). Technical report, 2014.

[13] G. Ormazabal, S. Nagpal, E. Yardeni, and H. Schulzrinne. Principles, Systems and Applications of IP Telecommunications - IPTComm. chapter Secure SIP: A Scalable Prevention Mechanism for DoS Attacks on SIP Based VoIP Systems. Springer, 2008.

[14] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Comput. Surv.*, 39(1), 2007.

[15] R. Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. In *Proceedings of the Ninth Symposium on Data Communications, SIGCOMM*. ACM, 1985.

[16] K. Phemius and M. Bouet. Monitoring latency with openflow. In *Network and Service Management (CNSM)*, 2013.

[17] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *SIGSAC Conference on Computer & Communications Security - CCS*. ACM, 2013.

[18] S. Shirali-Shahreza and Y. Ganjali. Flexam: Flexible sampling extension for monitoring and security applications in openflow. In *SIGCOMM Workshop on Hot Topics in Software Defined Networking*. ACM, 2013.

[19] M. Suh, S. H. Park, B. Lee, and S. Yang. Building firewall over the software-defined network controller. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, 2014.

[20] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. Opentm: Traffic matrix estimator for openflow networks. In *International Conference on Passive and Active Measurement - PAM*. Springer, 2010.

[21] N. L. M. Van Adrichem, D. Doerr, and F. A. Kuipers. OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks. In *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2014.

[22] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *International Conference on Passive and Active Measurement - PAM*. Springer, 2013.

[23] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Conference on Networked Systems Design and Implementation - NSDI*. USENIX, 2013.

[24] Y. Zhang. An adaptive flow counting method for anomaly detection in sdn. In *Conference on Emerging Networking Experiments and Technologies - CoNEXT*. ACM, 2013.