Network Intrusion Detection And Prevention Middlebox Management In SDN

Wen Wang
School of Computer Science
McGill University
Montreal, QC, Canada
wen.wang4@mail.mcgill.ca

Wenbo He
School of Computer Science
McGill University
Montreal, QC, Canada
wenbohe@cs.mcgill.ca

Jinshu Su
School of Computer
National University of Defense Technology
Changsha, Hunan, China
sjs@nudt.edu.cn

***Abstract*—In traditional networks, it is difficult to manage the distributed detection and prevention nodes of IDS and IPS due to the laborious manual deployment and independent configuration. Software defined networking (SDN) provides a flexible approach to control the underlying network infrastructures efficiently. However, the OpenFlow flow table is too simple to provide complex functions with the match-action style processing. To support more functionalities, in this paper, we propose a middlebox management architecture with SDN – OpenMiddlebox, by extending OpenFlow to support middleboxes with ClickOS virtual machines (VM), so that programmable middleboxes could be deployed and managed in switches with fast booted ClickOS VMs flexibly. We then design automatic deployment and update schemes of network intrusion detection and prevention middleboxes with the centralized controller. The evaluation results show that OpenMiddlebox could manage the distributed middleboxes efficiently and is scalable to large networks, and the centralized control also improves the network intrusion detection and prevention accuracy.**

I. Introduction

In traditional networks, to detect and prevent intrusions in network traffic, administrators usually have to deploy multiple intrusion detectors at different locations in a network and then analyze collected network traffic locally or at a centralized node. Unfortunately, the widely applied IDS/IPS architecture meets a lot of barriers to manage the distributed nodes efficiently. Firstly, the distributed intrusion detection nodes have to support different configurations. As configurations depend on the network topology, manual and frequent-changing configurations are inevitable to make policies in distributed nodes effective and consistent. Secondly, intrusion detection and prevention algorithms which determine the protection accuracy are usually designed for certain attack scenarios. To make intrusion detection nodes effective, more and more protection protocols are expected to be implemented in each node. However, employing multiple protocols and frequent updating complicate a node and even degrade processing performance. Moreover, network devices in traditional networks usually have specialized protocol implementations and control interfaces, which makes automatic management complex.

Considering the inflexibility resulting from the complexity and proprietary of switches, there are growing interests in abstracting network functions from dedicated switches to software-based applications in SDN. With OpenFlow switches controlled by the centralized controller, applications implemented in the controller communicate with switches through control messages to manipulate flow tables, while switches do not need to implement protocol details. Thus, security policies could be installed as rules in flow tables by the controller, instead of manual and independent configurations depending on the network topology and infrastructures. Additionally, SDN has natural statistics features which are useful for intrusion detection analysis, so that the centralized controller gains more visibility of the global network traffic. Therefore, SDN seems to provide a more suitable architecture for IDS and IPS.

Recently, there are a lot of intrusion detection schemes [1], [2], [3] designed for SDN. However, these applications running on the controller pose critical bottleneck challenges for the centralized computation and communication. As the separation of control plane and data plane abstracts the functionalities of switches to controller, these applications usually require to collect a great quantity of traffic information from switches to the controller for further analysis. Therefore, there are large overheads for the centralized computation and the communication between the two planes. What's worse, if connections between the switches and the controller are lost, these security applications also fail.

The root of the heavy overhead for the centralized controller is the complete functionality abstraction. In SDN, switches do not implement any network protocol detail, and only process packets as flow tables instruct. However, the simple match-action processing oversimplifies flow tables in switches, as actions in flow tables are only to forward, drop or modify packets. Thus, switches are incapable to implement and execute complex programs to perform intrusion detection and prevention, e.g., deep packet inspection. Even though a lot of researches [4] install security policies as flow entries in flow tables, the flow table only provides coarse-grained packets filtering according to the simple action field. To make switches powerful to support more sophisticated functions while maintaining the flexibility of SDN, switches should support programmable actions besides the simple flow table.

To enable programmable actions of switches while managing distributed nodes of IDS and IPS efficiently, in this paper, we propose a middlebox management architecture with

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

SDN, by extending OpenFlow to support more sophisticated actions with ClickOS [5] VMs. These programs in VMs serve as middleboxes and are constructed with elements provided by ClickOS. Thus, a middlebox could be defined by the controller with a configuration file and then executed distributedly in VMs of switches. Each switch detects and prevents malicious activities with middleboxes locally, and reports suspicious alerts to the controller. The centralized controller is only responsible for planning the middlebox deployment, coordinating the alert reports, and updating expired middleboxes. The evaluation shows this architecture is able to manage middleboxes in switches efficiently, and the centralized collaboration and middlebox updating mechanisms improve the network intrusion detection and prevention accuracy.

The rest of the paper is organized as follows. Section II looks at the existing IDS/IPS applications and middleboxes in SDN. Section III describes the OpenMiddlebox management architecture in SDN. Section IV designs automatic detection and prevention middleboxes deployment and update schemes. Section V evaluates the proposed architecture and approaches. Finally, Section VI concludes the paper.

II. Background and Related Work

A. Network IDS and IPS In SDN

Traditional IDS and IPS primarily focus on static networks with certain topology and manual policy configuration. However, the rapid evolving network attacks require the detection and prevention approaches keeping updating in real time. To make the management of network infrastructures more flexible, SDN shatters the proprietary implementations and independent configurations of network infrastructures. A lot of researches have been carried out for network monitoring and protection with the centralized control. [6], [7], [8] provides traffic monitoring with flow statistics polled by a centralized controller for anomaly detection. Security enhancement applications [3], [9], [10], [11], [12] are also deployed on the controller with modular components or extra tools (e.g., sFlow [13]). As OpenFlow switches simply process packets according to action field of flow tables, the abstraction leaves all control logic to the centralized controller. Increasing centralized applications in the controller not only creates communication and computation overheads on the controller but also may be a single point of failure.

With the increasing volume of network traffic, it is impossible for centralized IDS approaches to get detailed traffic information to a centralized node for analysis due to the great overhead. Even though sampling is a widely applied approach, it decreases the accuracy because of missing a lot of traffic details. Unfortunately, OpenFlow switches are incapable to implement and execute programmable actions. [14] notes that OpenFlow offers very limited support for fine-grained network monitoring applications at the data plane.

B. Middlebox In SDN

To support middleboxes in SDN, [15], [16] steer traffic through the desired sequence of middleboxes including firewalls and IDS without mandating any placement or implementation constraints on middleboxes. In spite of the desired forwarding sequence of middleboxes, policies in middleboxes should also be carefully considered to be consistent and effective. OpenNF [17] managed both network forwarding state and internal NF state to coordinate race conditions.

Even though the SDN controller is able to direct packets through middleboxes with desired sequence and consistency, switches provide little capability for middlebox functions. [18] notes that even if an OpenFlow device is now rich of functionalities and primitives, it remains completely "dumb", with all the "smartness" placed at the controller side. Therefore, [18] proposes a viable abstraction to formally describe a desired stateful processing of flows inside the device with XFSM. [19] describes a reconfigurable match tables model which allows match tables in the data plane to be reconfigured. [20] enables end-hosts to coordinate with switches to implement a wide-range of network tasks, by embedding tiny programs into packets that execute directly in the dataplane. ClickOS [21] makes the data plane programmable with a tiny Xen-based virtual machine that can run a wide range of middleboxes.

III. System Architecture

A. Middlebox In Switch

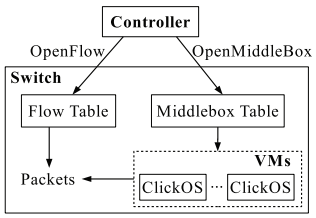


Fig. 1: OpenMiddleBox Architecture

To reduce the computation and communication overhead on the centralized controller, we use OpenFlow switches as distributed nodes for intrusion detection and prevention. As packets always have to go through a series of switches along their routing paths, switches are able to capture traffic variation and throttle malware spreading, which are essential for intrusion detection and prevention. To make switches more powerful and flexible, despite the simple match-action processing with the OpenFlow flow table, we extend switches to support IDS and IPS middleboxes with ClickOS [5]. ClickOS is a Xen-based tiny virtual machine that runs Click [22], and it can be quickly instantiated in $30ms$ with a compressed 5MB image. As Click equips with over 300 stock elements, these elements make it possible to construct middleboxes with minimal efforts. Moreover, we can easily extend this framework and construct new elements to support more middleboxes [21]. To set up a middlebox in ClickOS VM, the controller dispatches a Click configuration to related switches, which is essentially a text file specifying elements. Upon receiving the configuration file, the switch boots a VM for the middlebox based on the defined configurations. Therefore, middleboxes in VMs enable

switches to perform more complicated actions in addition to simple forwarding, modifying or dropping actions of flow tables. Meanwhile, middleboxes are isolated into multiple fast booted VMs, so that they do not interfere with each other during processing.

B. Middlebox Control

To support middlebox configuration, we design OpenMiddlebox control messages to manage middleboxes in switches with the controller. For the setup of a middlebox, the controller sends the Click configuration request to a switch with the OpenMiddlebox control message, so that the configuration is used to instantiate a middlebox VM. A middlebox entry is installed in the middlebox table at the setup of middlebox as Figure 1 shows. The entry in the middlebox table records the corresponding ClickOS VM and middlebox configuration. For the configuration changes, e.g., inserting rules in an IDS, as ClickOS can change configurations at runtime, the controller just needs to send new configurations to switches. The deletion of middleboxes is more straightforward by shutting down the corresponding VMs and removing the related middlebox entries. Middleboxes are isolated into VMs with restricted memory and CPU resources, and ClickOS accesses packets with a direct pipe between NIC and VMs [5]. Therefore, middleboxes in VMs do not affect the basic efficiency of data plane packet processing, while the OpenFlow protocol handles regular requests to manipulate the flow table as usual. In addition to the control of middleboxes at the centralized controller, middleboxes in switches may also require to communicate with the centralized controller for specific reports, e.g., an IDS middlebox would report alerts for detected intrusions to the controller for further analysis. The OpenMiddlebox protocol also supports this communication requirement by sending middlebox ID with the report which is understandable for the middlebox manager in the controller.

C. Middlebox Authorization

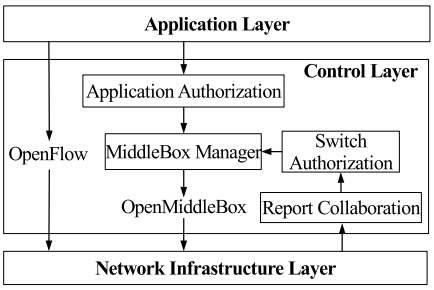


Fig. 2: Middlebox Authorization

1) Application Authorization: The main idea of SDN is to create a network abstraction layer, enabling applications running on top of the controller. With the open APIs for third-party applications, it makes the network more programmable and flexible. However, it also increases the risks of malware to compromise the network. The absence of the northbound API fails to decide the access permissions granted to network applications. As applications are able to manipulate switches, poorly implemented, misconfigured or malicious applications may modify flows erroneously. Especially for middleboxes in switches, abused configurations would totally collapse a switch and mess the network. Therefore, we should limit authority of applications to access middleboxes in switches. With a white list of authorized applications, only the middlebox configurations issued by the authorized application are processed, otherwise requests will be rejected to send any control message to manipulate middlebox tables.

2) Switch Authorization: Middleboxes run in switches independently and communicate with the controller occasionally. However, a malicious switch may alter the middlebox configuration to evade detection or send false reports to confuse the controller. Therefore, there is a need to check whether the switches are legitimate and remove the compromised. Initially, we treat all newly connected switches equally to be legitimate. When an OpenFlow switch tries to set up a connection with the controller, the controller checks its identity to decide whether it could be granted the permission for implementing middleboxes and records it in the granted list. If a middlebox in the switch always sends false reports conflicting with other middleboxes of the same type, it will be kicked out from the granted list based on the middlebox report accuracy in Section IV-B, and the corresponding middlebox in the switch are also destroyed.

IV. Intrusion Detection And Prevention Middlebox Management

With the OpenMiddlebox architecture, various middleboxes (e.g., IDS, firewall, NAT) could be constructed with elements in ClickOS. In this section, we show how to manage distributed network intrusion detection and prevention middleboxes with a centralized view.

A. Middlebox Deployment Distribution

To detect anomalous activities in the network, intrusion detection middleboxes composed of Click elements (e.g., IPRateMonitor, TCPCollector, Classifier) should be installed in multiple switches at different locations. In contrasted with IDS deployed at core switches in traditional networks, as the controller is aware of network topology and flow forwarding paths, the middlebox deployment distribution in SDN could be dynamically planned. Therefore, instead of installing and configuring middleboxes manually and independently in each switch, the controller could automatically install middleboxes in concerned switches for the designated network traffic. As packet processing in SDN is flow-based, the controller plans middleboxes distribution based on the flow matching criteria, such as middlebox detecting anomalies in flows with the same source IP address or destination port of flows. A middlebox m is installed in switches to detect abnormal activities of a subclass of flows $F(m) = \{f_1, f_2, ...\}$. A flow f always travels through several switches $S(f) = \{s_1^f, s_2^f, ..., s_d^f\}$ along the routing path to reach the destination. Thus, the switches in the

routing paths are able to capture flows' activities and used for intrusion detection. To avoid high false positive due to a single detection node, we install middlebox instances in multiple switches for m and then compare the detection results from the multiple switches to make a final decision. $S = \{s_1, s_2, .., s_n\}$ is the set of switches which are in the routing paths of flows in subclass $F(m)$. Therefore, the intrusion detection middlebox instances of m should be installed in a subset of S to capture most of flows in $F(m)$.

As switches always query the controller about the routing path of a new flow with $PACKET\_IN$, the controller is able to get each flow's routing path and the traffic distribution of the monitored flow class. Meanwhile, the controller is notified with $FLOW\_REMOVED$ messages when flow entries are deleted from flow tables in switches. Thus, the controller always has the knowledge of ongoing flows $F(s_i, m)$ related to middlebox m in switch s_i. Upon the arrival and removal of a flow f ($f \in F(m)$), the ongoing flow set $F(s_i, m)$ is updated if s_i is on f's forwarding path S(f).

$$F(s_i, m) = \begin{cases} F(s_i, m) \cup \{f\} & newflow(f) \wedge s_i \in S(f) \\ F(s_i, m) - \{f\} & delflow(f) \wedge s_i \in S(f) \end{cases}$$

Intuitively, we would like to install middleboxes in the switches which are able to capture a large number of flows other than those processing only a small amount in the monitored subclass, as a small percentage may lead to subjective view. By sorting switches s_i $(i = 1, 2, ..., n)$ in S as the descending sequence of monitored flow numbers $|F(s_i)|$, the first switches of sorted S cover most of flows in the subclass. We choose the first k switches $S(m) = \{s_1^m, s_2^m, .., s_k^m\}$ which cover at least p percent of flows in subclass $F(m)$ to install intrusion detection middlebox instances for middlebox m.

$$\min_k |\bigcup_{i=1}^{k} F(s_i^m, m)| \geqslant p \cdot |F(m)| \qquad (0 \leq p \leq 1)$$

The p controls the monitoring accuracy and the intrusion detection overhead, as the larger p is, the more switches are involved in intrusion detection, but the more flows are monitored for intrusion detection.

B. Collaborative Network Intrusion Detection

With multiple middleboxes instances running in multiple switches for each intrusion detection middlebox, switches perform intrusion detection in real time, and then report suspicious alerts to the centralized controller. The controller collaborates these alerts to make an intrusion decision, as collaboration usually achieves more accurate results.

Intuitively, if the majority of instances of middlebox m report the same alert, we are more confident that there is an abnormal activity, otherwise it is probably a false alert and we will ignore it. However, the middlebox instances in different switches process with different ongoing flow sets, and alerts generated by these switches also vary because of different monitored flows. To reflect the differences in monitored flows, we assign weights to alerts from different switches. The alert a_i^m of middlebox m in switch s_i^m is assigned with a weight $w_i^m(1)$, and the non-alert report (none report) also equips with a weight $w_i^m(0)$. The alert and non-alert weights $w_i^m(t)(t = 0, 1)$ of the switch s_i^m are defined as the product of alert/none-alert accuracy $\frac{na_i^m(t)}{na^m(t)}$ and the percentage of monitored flows in the monitored subclass $\frac{|F(s_i^m,m)|}{|F(m)|}$. The alert/non-alert accuracy $\frac{na_i^m(t)}{na^m(t)}$ indicates the accuracy confidence of the alert or none-alert, in which $na^m(t)$ is collaboratively decided alert($t = 1$)/non-alert($t = 0$) count and $na_i^m(t)$ is the count that alert/non-alert reports of middlebox m in switch s_i^m are consistent with collaborative decisions. The percentage of monitored flows $\frac{|F(s_i^m,m)|}{|F(m)|}$ means the flow set coverage confidence of switch s_i^m in the monitored subclass $F(m)$. We compare the confidence of true-intrusion $\sum_{i=1}^{k} w_i^m(1) \cdot a_i^m$ and non-intrusion $\sum_{i=1}^{k} w_i^m(0) \cdot (1 - a_i^m)$ to decide whether it is a true intrusion a^m. If the true-intrusion confidence is larger, it indicates a probable true intrusion.

$$a_i^m = \begin{cases} 1 & middlebox\ m\ in\ switch\ s_i^m\ reports\ an\ alert \\ 0 & else: no\ alert \end{cases}$$

$$a^m = \begin{cases} 1 & \sum_{i=1}^{k} w_i^m(1) \cdot a_i^m > \sum_{i=1}^{k} w_i^m(0) \cdot (1 - a_i^m) \\ 0 & else: no\ intrusion \end{cases}$$

$$na_i^m(t) = \begin{cases} na_i^m(t) + 1 & a_i^m = t \bigwedge a^m = t \\ na_i^m(t) & else \end{cases} \quad (t = 0, 1)$$

$$na^m(t) = \begin{cases} na^m(t) + 1 & a^m = t \\ na^m(t) & else \end{cases} \quad (t = 0, 1)$$

$$w_i^m(t) = \frac{na_i^m(t)}{na^m(t)} \cdot \frac{|F(s_i^m, m)|}{|F(m)|} \quad (t = 0, 1)$$

Algorithm 1 Alert Collaboration

1: Calculate a^m with $w_i^m(0), w_i^m(1), a_i^m$ $(i = 1, ..., k)$
2: **if** $a^m == 1$ **then**
3: decide a true alert
4: **end if**
5: update $w_i^m(0), w_i^m(1), na_i^m(0), na_i^m(1)$ $(i = 1, ..., k)$
6: **for** each switch $s_i^m \in S(m)$ **do**
7: $accuracy(t) = \frac{na_i^m(t)}{na^m(t)}$ $(t = 0, 1)$
8: **if** $accuracy(1) < 0.5 \ \wedge \ accuracy(0) < 0.5$ **then**
9: $S(m) = S(m) - s_i^m$, remove middlebox m in s_i^m
10: **end if**
11: **end for**
12: Update middlebox instance set of m

We design Algorithm 1 to collaborate alerts. Lines 1-4 calculate collaborative alert with weights and alerts from multiple instances of middlebox m. In spite of reducing false positive, the centralized collaboration also bring another merit to identify abnormal reporting activity of switches. As the controller has no idea about whether the switches report a truely detected intrusion or a false alert deliberately, the collaboration helps to identify the anomalous behaviors of switches. If a compromised switch always sends different alerts conflicting with others, it will result in low weight

$w_i^m(t)(t = 0, 1)$ due to the low accuracy of alert report, so that it has little impact on alerts collaboration. When the accuracy is lower than 50% (Lines 8-10) which means it is even worse than random alert report, the controller would remove the switch from $S(m)$ and stop the middlebox instance in the switch by shutting down the corresponding ClickOS VM.

The centralized collaboration ignores insignificant alerts with low confidence. However, as middlebox instances in multiple switches process different flows going through them, a switch may report a alert for a true small-regional anomaly which is invisible to other switches, especially at the beginning stage of malware explosion during which only a small part of a network is infected. To perform fine-grained monitoring, administrators could divide the monitored class $F(m)$ into multiple fine-grained classes and increase the monitored flow percentage p to avoid false negative.

C. Intrusion Prevention Management

1) Intrusion Prevention In SDN: When the controller determines a true intrusion with collaborative reports, it should take actions to install prevention approaches in switches to stop the abnormal activities. The prevention approaches could be security policies installed in flow tables, e.g., setting the action field of the flow entry as dropping. Although a flow entry could act as a security policy and drop packets of a malicious flow to throttle the propagation of malware, dropping an entire flow may lead to high false positive, because only a portion of packets contain the malicious pattern. Thus, the coarse flow prevention with the flow table is not an advisable choice. Moreover, the flow matching capability with the flow table is limited, so that it is unable to identify packets containing certain patterns in their payloads or malicious activities at session level. It is not preferred to extend the flow table with more security fields as matching with more field against a flow entry will decrease processing efficiency. Due to the limitation of the flow table, OpenFlow is not capable to perform fine-grained intrusion filtering.

The OpenMiddlebox remedies the prevention functions of the flow table. With middleboxes composed of Click elements (e.g., IPClassifier, IPFilter) to filter abnormal traffic matching the malicious pattern, these intrusion prevention middleboxes could be triggered by more complicated conditions defined by the programmable elements in addition to the simple matching field in flow entry. Thus, middleboxes are able to act against anomalies that evade defenses of the simple security policies in flow tables.

To avoid high false alarm rate of intrusion preventions, we distribute multiple preventions $P = \{p_1, p_2, ..., p_t\}$ at multiple locations $L(P)$ for intrusions detected by middlebox m. $L(P)$ is composed of switches that are able to capture malicious flows of the intrusion detected by middlebox m.

$$L(P) = \{s_i^m | s_i^m \in S(m), f \in F(s_i^m, m) \wedge f \in Malicious(m)\}$$

2) Intrusion Prevention Updating: When an intrusion is detected, experts analyze the intrusion pattern to generate the prevention solutions and then install preventions in switches to stop malicious activities. However, intrusion patterns always change as time goes and evolve to be new types of anomalies, so that previously installed prevention solutions may fail to detect the latest anomalies. Therefore, switches should be aware of the changes in intrusion patterns to update preventions. If a prevention solution does not detect any intrusion for a long time, the cases may divide. Optimistically, it means there is no intrusion threat at all. On the other hand, the intrusion pattern changes such that the malicious pattern previously defined in the prevention fails. For both these cases, the preventions require to be removed or updated in switches. Hence, only effective middlebox instances are kept and updated in switches, while redundant and expired preventions are removed to relieve the processing overhead.

According to the anomaly-based and signature-based IDS, the updates of preventions could be divided into two categories. As the anomaly-based intrusion detection learns normal traffic patterns and distinguishes anomalies from the normal, the normal model may vary as time goes, such that preventions may fail to capture and stop anomalies. Thus, the baseline models in anomaly-based prevention require to be updated. For the signature-based intrusion prevention, signatures used in prevention middleboxes may fail due to the changes of malicious signatures, especially for the polymorphic malware which keeps evolving to evade intrusion prevention. Therefore, the applied signatures in prevention middleboxes should be updated. As the controller maintains the global control of detection and prevention middleboxes, it compares the detection and prevention results to decide whether preventions fail to stop malicious activities. Intuitively, if a prevention p_j always fails to capture anomalies which trigger alerts of the corresponding intrusion detection middlebox, this prevention is ineffective due to the low prevention accuracy. Therefore, the update of intrusion prevention p_j depends on the defense accuracy compared with the alerts generated by the intrusion detection middlebox m.

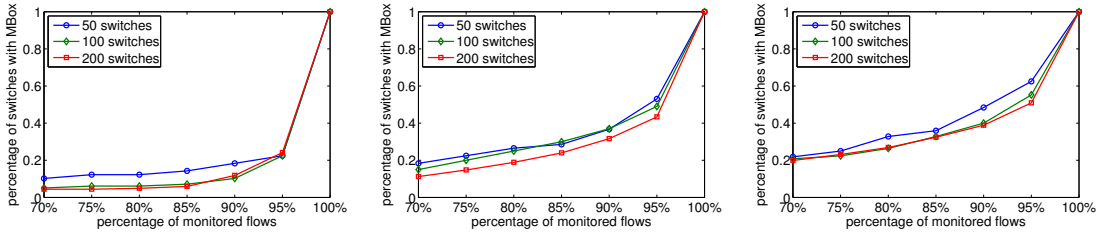
Accuracy is the key factor to evaluate the effectiveness of an intrusion detection and prevention system. For a prevention p ($p \in P$), we compare the defense result $d(p)$ of the prevention with alerts $a(m)$ generated by the corresponding detection middlebox m to determine the accuracy

$$accuracy(p) = \frac{N(d(p) \wedge a(m))}{N(d(p) \wedge a(m)) + N(\neg d(p) \wedge a(m))}$$

in which $N(d(p) \wedge a(m))$ and $N(\neg d(p) \wedge a(m))$ are the true positive count and false negative count compared with the detection middlebox m. To ensure the efficiency of preventions, we would like to retain preventions with high accuracy, and the ineffective preventions with low accuracy are reported to the controller to require update.

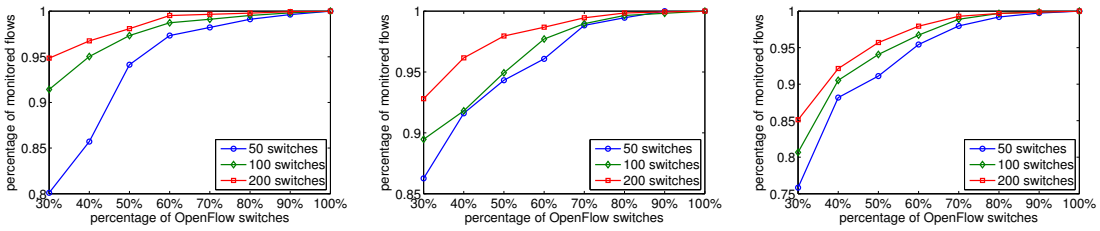
V. System Evaluation

With the centralized control and management of middleboxes, we evaluate the performance of middlebox distribution and accuracy of the intrusion detection and prevention. As the controller plans middlebox distribution management and



(a) Fat-tree (b) 2D mesh (c) 3D torus

Fig. 3: The percentage of switches used for monitoring



(a) Fat-tree (b) 2D mesh (c) 3D torus

Fig. 4: The percentage of monitored flows in hybrid SDN

collaborates middlebox reports according to network topology and traffic distribution, we evaluate the proposed approaches with three kinds of topologies: fat-tree, 2-D mesh, 3-D torus. The node location and link interconnection are different in these three topologies, so that the distribution and collaboration of intrusion detection and prevention middleboxes are also different.

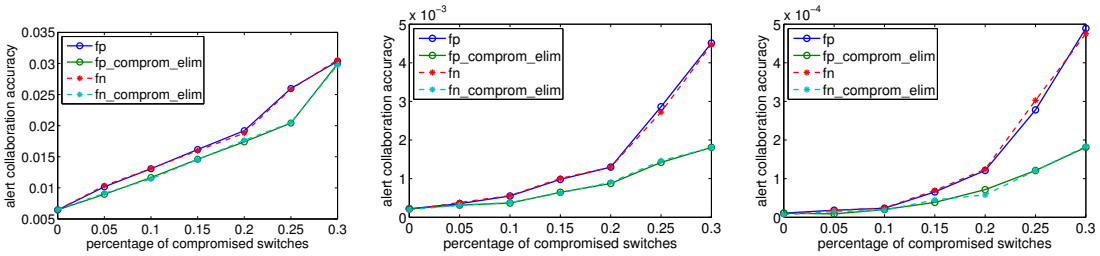
A. Distribution Of Middleboxes

Intuitively, to capture details of network traffic, more middleboxes should be deployed in different locations of the network. Meanwhile, these middleboxes are required to be able to monitor most of network traffic. Figure 3 shows the percentage of switches running middlebox instances to monitor the entire network when the traffic is equally distributed among hosts. In the three kinds of network topologies, with less than a half number of switches running middlebox VMs, these middleboxes are able to monitor most of traffic (95%) in the network. Especially for the fat-tree topology, the entire network traffic could be monitored with middleboxes in 20% switches in the network. It means the controller only needs to manage a few number of middleboxes to monitor the entire network traffic. The 3-D torus topology has more redundant routing paths, so that network traffic is probably routed through various different paths and evenly distributed among switches. Therefore, 3-D torus employs larger number of switches to monitor a certain percentage of network traffic compared with fat-tree and 2-D mesh in Figure 3c. We also note that, the larger the network is, the number of switches running middlebox instances accounts for less proportion of the network infrastructures, e.g., the network composed of 200 switches employs a less percentage of switches to monitor the entire network than a small network with 50 switches. Compared with the fairly distributed network traffic among nodes, when traffic concentrates on a small number of nodes, the switches on the critical routes to these nodes are able to capture most of the traffic. Thus, there will be less switches running middleboxes with the concentrated traffic.

The evolving of enterprise networks and data center networks from traditional networks to software defined networks is usually an incremental process. It is inevitable that hybrid networks operate with partial OpenFlow switches while others still run as non-OpenFlow switches. Fortunately, these partial OpenFlow switches are still able to capture most of network traffics. In Figure 4, when OpenFlow switches are randomly deployed in the network, at least 75% of traffic could be monitored with 30% OpenFlow switches in these three topologies. Due to the routing path diversity, the OpenFlow switches in 3D torus network capture less amount of traffic in Figure 4c compared with fat-tree and 2-D mesh topologies. The larger the network is, a certain percentage of OpenFlow switches could monitor more network traffic than small networks.

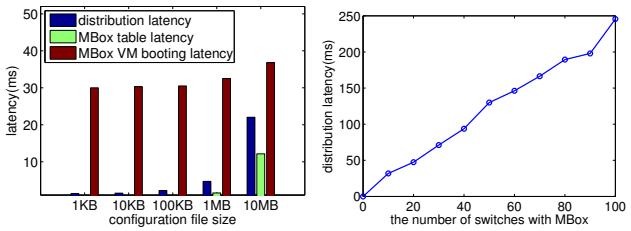
B. Scalability

The middleboxes in switches not only reduce the communication overhead between the controller and switches, it also relieves administrators from heavy labour workload by deploying and controlling these middleboxes automatically with the centralized controller. To show the efficiency of managing these middleboxes, we evaluate the performance of the OpenMiddlebox architecture with the scaling of the configuration file size and the network size respectively. As switches are connected directly to the controller, the middlebox management with the controller is independent from the



(a) Fat-tree (b) 2D mesh (c) 3D torus

Fig. 6: Alert collaboration accuracy with compromised switches



(a) Scalability with file size (b) Scalability with network size

Fig. 5: Scalability with file and network size

TABLE I: Collaboration accuracy

Topology		Switch alert accuracy						
		0.70	0.75	0.80	0.85	0.90	0.95	1
Fat-tree	FP	2.636%	0.645%	0.102%	0.007%	0	0	0
	FN	2.655%	0.649%	0.100%	0.007%	0	0	0
2D mesh	FP	0.291%	0.022%	0	0	0	0	0
	FN	0.294%	0.019%	0	0	0	0	0
3D torus	FP	0.036%	0.001%	0	0	0	0	0
	FN	0.035%	0.001%	0	0	0	0	0

network topology. We test a 2-D mesh network with Mininet [23], and each switch in the network connects to the controller with a 1Gb/s link.

As the configuration file only needs to define the element names and rules with integrated elements in ClickOS, the size of configuration file is usually small at the level of kilobytes. However, with the increasing number of security policies defined with elements IPFilter, IPClassifier, the configuration file could be as large as several megabytes. In Figure 5a, when the size of configuration file grows, the distribution latency also increases, and it takes over 20ms to send a 10MB file to a switch. Nevertheless, it is still acceptable for the overall lifetime of a middlebox, as the configuration file is only transferred at the beginning. In the OpenMiddlebox architecture, when a switch sets up a middlebox, it inserts a corresponding entry in the middlebox table and records the configuration on the disk, and then boots the middlebox in ClickOS VM. We notice that the middlebox VM booting takes about 30ms and does not increase a lot when the configuration file size grows. The overall middlebox setting up time is less than 100ms for a 10MB configuration file, thus the setting up latency is quite acceptable to relieve the centralized controller from frequent traffic information fetching.

Meanwhile, when the network size scales, the number of switches running middleboxes to monitor the network also grows, so that the overhead to distribute and manage middleboxes in switches is also expected to increase as Figrue 5b shows. It takes about 250ms to distribute a 100KB middlebox configuration file to 100 switches at once, which is still much shorter than the collecting and sampling frequency in centralized approaches which usually perform intrusion detection at the level of several seconds. Therefore, the distribution and management overhead of middleboxes is considerably acceptable. As Figure 3 and Figure 4 show that partial switches in the network are able to monitor most of the entire network traffic, the middlebox management architecture could be deployed in a large network with a part of switches running middlebox VMs to monitor and protect the entire network.

C. Alert Accuracy

Accuracy is usually a key criteria to evaluate IDS and IPS. In this section, we evaluate the accuracy of collaborative alert reports. We have to note that the detection and prevention accuracy greatly depends on the algorithms in middleboxes. We do not evaluate certain algorithms here as most algorithms could be adapted to be middleboxes in switches, and we only focus on the collaboration of multiple middlebox instances.

For networks composed of 100 switches, Table I shows that the centralized alert collaboration significantly improves the overall alert accuracy than intrusion detection accuracy of middleboxes in a single switch when the accuracy of each switch ranges from 0.70 to 1. As the network traffic distribution varies in different network topologies, the weights of switches are also different. Compared with 2D mesh and 3D torus networks, in the fat-tree topology, the core and aggregation switches usually process more traffic than access switches, which results in larger weights for higher layer switches. Therefore, core and aggregation switches have heavier impact in alert collaboration and lead the collaboration results. On the other hand, the collaboration results are likely to be mislead by the false positive and false negative of these switches. In 2D mesh and 3D torus topologies, the traffic distribution is much more fair among switches, especially for the 3D torus in which there

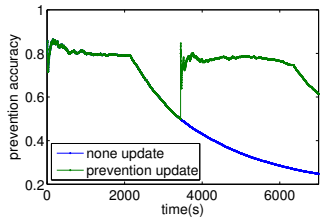


Fig. 7: Prevention accuracy

are more redundant routing paths for load balancing. The fair traffic distribution makes the weights of switches similar, thus the collaborative decision is more objective and impartial with lower false positive and false negative.

Furthermore, switches may be compromised and report false alerts deliberately to confuse the controller. Although the alert collaboration has reduced the impact of false alerts to some extent, the compromise elimination of switches further gets rid of compromised switches to bring down the false positive and false negative of alert collaboration in Figure 6. Therefore, the compromise elimination helps to improve the overall protection accuracy. Especially for the 2D mesh and 3D torus networks, switches have similar weights in these topologies compared with fat-tree network, such that the compromised switches are more easier to be recognized. For the fat-tree network, the core and aggregate switches have larger weights and tend to lead the alert collaboration results, so that it is difficult to identify compromise in these switches. The collaboration results also show that the false positive and false negative are quite similar and both stay at low level, because we have taken both the alerts and none alerts accuracy of each middlebox into the collaboration.

D. Prevention Updating

To test the validity of prevention updating, we simulate the process of malware evolving while the patterns defined in prevention middleboxes stay. As prevention updating depends on the accuracy of alerts reported by switches, we assume that middleboxes detect intrusions with accuracy 80% in the simulation. With malicious patterns changing as time goes, intrusion detection middleboxes learn the abnormal pattern in real time while the predefined patterns in prevention are static. Hence, the accuracy of prevention decreases with the evolving of the malicious in Figure 7, because the static patterns in prevention fail to match the evolved malicious. When the accuracy decreases as the malicious patterns change, if we do not update the prevention scheme, the accuracy keeps dropping. However, if we update or replace the prevention when the estimated accuracy is less than 50%, it is able to achieve high accuracy again at about 3500s in Figure 7.

VI. Conclusions

Considering the inflexibility of IDS and IPS architecture in traditional networks, we note that SDN provides a flexible architecture to control distributed nodes. However, the abstracted SDN data plane provides little support for complicated middlebox processing. To enable programmable middleboxes in OpenFlow switches, we propose a middlebox management architecture to control distributed intrusion detection and prevention middleboxes in switches. Middleboxes run as isolated ClickOS VMs in switches and are constructed with elements provided by ClickOS. The controller manages middleboxes with configuration files. We further design deployment and update approaches for intrusion detection and prevention middleboxes with the proposed architecture. The evaluation shows that the middlebox management architecture is able to deploy and control middleboxes in switches effectively and is well scalable to large networks.

References

[1] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *RAID*, 2011.
[2] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *LCN*, 2010.
[3] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," in *NDSS*, 2015.
[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu *et al.*, "Clickos and the art of network function virtualization," in *NSDI*, 2014.
[6] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *NOMS*, 2014.
[7] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch." in *NSDI*, 2013.
[8] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *CoNEXT*, 2013.
[9] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras *et al.*, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.
[10] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building robust firewalls for software-defined networks," in *HotSDN*, 2014.
[11] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong *et al.*, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013.
[12] S. Shirali-Shahreza and Y. Ganjali, "Flexam: Flexible sampling extension for monitoring and security applications in openflow," in *HotSDN*, 2013.
[13] "sflow," http://www.sflow.org/.
[14] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *CCS*, 2013.
[15] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao *et al.*, "Simple-fying middlebox policy enforcement using sdn," in *SIGCOMM*, 2013.
[16] B. Anwer, T. Benson, N. Feamster, D. Levin *et al.*, "A slick control plane for network middleboxes," in *HotSDN*, 2013.
[17] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl *et al.*, "Opennf: Enabling innovation in network function control," in *SIGCOMM*, 2014.
[18] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch," in *SIGCOMM*, 2014.
[19] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *SIGCOMM*, 2013.
[20] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazières, "Tiny packet programs for low-latency network control and monitoring," in *HotNets*, 2013.
[21] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with clickos," in *HotSDN*, 2013.
[22] E. Kohler, R. Morris, B. Chen, J. Jannotti *et al.*, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.
[23] "Mininet," http://mininet.org/.