Piggybacking Network Functions on SDN Reactive Routing: A Feasibility Study

Chang Liu[*], Arun Raghuramu[*], Chen-Nee Chuah[*], and Balachander Krishnamurthy[**]

[*]University of California, Davis; **AT&T Labs-Research

ABSTRACT

This paper explores the potential of enabling SDN security and monitoring services by piggybacking on SDN reactive routing. As a case study, we implement and evaluate a piggybacking based intrusion prevention system called SDN-Defense. Our study of university WiFi traffic traces reveals that up to 73% of malicious flows can be detected by inspecting just the first three packets of a flow, and 90% of malicious flows from the first four packets. Using such empirical insights, we propose to forward the first K packets of each new flow to an augmented SDN controller for security inspection, where K is a dynamically configurable parameter. We characterize the cost-benefit trade-offs of SDN-Defense using real wireless traces and discuss potential scalability issues. Finally, we discuss other applications which can be enhanced by using our proposed piggybacking approach.

CCS Concepts

•**Networks** → **Network security;**

Keywords

SDN; Network Security

1. INTRODUCTION

Software-defined networking (SDN) is widely considered to be the networking architecture of choice for future networks. The SDN controller has a global view of the network and can dynamically configure the forwarding rules in managed SDN switches. Network function virtualization (NFV) is another technology trend that is complementary to SDN, allowing network functions to run as virtual machines or containers running on the same physical hardware, e.g. a blade server.

One common operational mode of SDNs is reactive routing where the first few[1] packets of a flow will be forwarded to the SDN controller to determine which actions should be taken before the associated forwarding rules are installed in the SDN switch/router. This is useful for networks where logical topology is continuously evolving based on traffic statistics and policies. There is a unique, untapped opportunity here given that the SDN controller has a global view and can potentially *inspect* first K packets of each flow, where K can be a configurable parameter.

In this paper, we explore the potential of supporting new network services by *piggybacking network functions on SDN reactive routing*. For instance, inspecting the first few packets seen through reactive routing can be critical for early intrusion detection, or for classifying packets belonging to specific service types for QoS provisioning and accounting. As a concrete example, we provide an in-depth feasibility study of offloading signature-based intrusion prevention to the SDN controller by piggybacking on reactive routing- an approach referred to as SDN-Defense. Some potential benefits of SDN-Defense include: 1) earlier/faster detection and mitigation, i.e., if a malicious flow is detected at the SDN controller, it can install appropriate rules at all the switches to block such future traffic, 2) relieving load on traditional intrusion detection/prevention systems (IDS/IPS), and 3) more comprehensive coverage with global view of SDN controller, hence detecting threats not seen by traditional IDS/IPS middleboxes that only inspect traffic from local links they monitor at the network edge. Our contributions are threefold:

- We analyzed 296GB of WiFi traffic from a large US-based university campus and demonstrated that 73% of the malicious flows can be detected by just inspecting the first three packets of a flow, and 90%

SOSR '17, April 3-4, 2017, Santa Clara, CA, USA
© 2017 ACM. ISBN 978-1-4503-4947-5/17/04...$15.00
DOI: http://dx.doi.org/10.1145/3050220.3050225

[1]For a TCP connection, only the first packet is forwarded, whereas for a UDP connection, the first few packets arriving at the switch without routing information get forwarded.

of malicious flows from the first four packets.

- We propose, implement, and evaluate SDN-Defense, where SDN reactive routing is augmented with Snort rules to perform *first K-packet inspection.* We characterize the cost-benefit trade-offs of SDN-Defense using real WiFi traffic traces.
- We discuss other SDN services such as application identification and dynamic Traffic Dispersion Graph (TDG) generation, which can be enhanced with our proposed piggybacking approach.

We will discuss related work next (Section 2), followed by an overview of our framework (Section 3). Section 4 presents the cost-benefit analysis of SDN-Defense. Section 5.1 discusses controller scalability and Section 5.2 describes other potential piggybacking-based applications. We conclude the paper in Section 6.

2. RELATED WORK

2.1 Network intrusion detection

There is a large body of prior studies on network intrusion detection. Liao et al. [10] offer a comprehensive review of existing intrusion detection methodologies and systems. In this work, we utilized Snort [5], a widely deployed, open-source, signature based IDS to demonstrate our proposed offloading framework.

Papadogiannakis et al. [15], propose selective packet discarding — a best-effort approach which enables a NIDS to anticipate overload conditions and minimize the impact on attack detection. Instead of letting the packet capturing subsystem randomly drop arriving packets in case of overload, the authors propose to discard packets that are less likely to affect detection accuracy and focus on the traffic at the early stages of network flows. Unlike our work, the system proposed in [15] is for a traditional (non-SDN) network setting. Their solution faces all the limitations of an IDS operating in the traditional network, e.g. the IDS only sees traffic from local links. Furthermore, our proposed framework is general and can enhance other applications such as application classification as we will discuss in Section 5.

2.2 Intrusion detection in SDN

We now describe some attempts in the past focusing on implementing network intrusion detection systems in SDN. Syed et al. [14] propose that SDN can be used to effectively detect and contain network anomalies in home and home office networks. Giotis et al. [8] propose combining OpenFlow and sFlow to implement anomaly detection and mitigation in general SDN environments. Unlike these works which implement an anomaly based intrusion detection system, our case study involves running a signature-based detector (Snort) on the controller. Another important distinction is that both [14] and [8] utilize a limited and narrow set of synthetically generated attack traffic (e.g. port scan, DoS) to evaluate their system whereas we perform detailed analysis by utilizing real-world malicious traffic containing 44 distinct threat types seen in a university campus trace.

Yoon et al. [12] analyze the feasibility of implementing various SDN-based security functions. They implement the signature based NIPS/NIDS function on a popular SDN platform. Their proposed approach is based on reactive routing. In their system, the first packet of a new flow is sent to the controller and is then passed to NIPS/NIDS. NIPS/NIDS determines if the flow should go through packet inspection process and based on the decision, different forwarding rules are installed for the flow. However, there is no detailed trace-based evaluation of their proposed approach. In fact, our studies using real network traces demonstrate that inspecting just the first packet is often insufficient in determining if a flow is malicious or not.

An architecture for integrating Snort IDS with SDN for cloud environments is discussed in [11]. Specifically, they utilize XenServer and run Snort checks on VM generated traffic in the hypervisor management domain. The SDN controller then interprets the alerts generated by Snort and controls an OVS switch to take dynamic routing decisions. Unlike [11], the framework presented in this paper is more general and not tied to a particular deployment scenario. The piggybacking mechanism we present can also be easily extended to perform other network monitoring functions. Finally, the evaluation performed in their work is incomplete from a security perspective since they, like earlier work, only utilize a narrow set of synthetic traffic.

3. CASE STUDY: PIGGYBACKING IDS/IPS ON SDN REACTIVE ROUTING

In this section, we describe the idea of piggybacking IDS/IPS functions on SDN reactive routing and study its benefit and cost. There is a broad design space for incorporating these functions in SDN networks. At one end of the spectrum is the base case where SDN controller/switches are not involved and IDS/IPS middleboxes operate in the traditional way: only processing traffic from local links they monitor. On the other end of the spectrum, the complete IPS functions are running as applications within SDN controller. Alternately, IPS can be introduced as a VNF (Virtual Network Function) and co-located with the controller. Clearly, forwarding all traffic to the controller for inspection will incur significant overhead. Our goal is to explore the sweet spot in between by piggybacking on SDN reactive routing.

3.1 System Overview

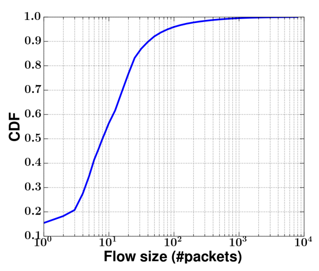
In reactive routing mode, the first packet of a new flow is sent to the SDN controller for routing information. The controller responds by installing a forwarding rule for this flow. Subsequent packets of the same flow are matched with the forwarding rule installed at the SDN switch/router and routed at line rate.

Piggybacking on the SDN reactive routing, we propose a framework called *SDN-Defense*, which inspects the first K packets of a flow forwarded to the SDN controller. In SDN-Defense, the controller delays installation of a flow's forwarding rule at the **edge** switch/router until the K_{th} packet of this flow is inspected (unless an earlier decision is made). Each of these first K packets are inspected by selected IDS/IPS rules (a design choice) at the controller. If the packet is benign but the controller has not seen all the first K packets, the controller will send this packet to the corresponding output port of its edge switch via a *packet_out* message without installing a forwarding rule for this flow. Then the packet is forwarded to its destination as normal. If the packet is detected to be malicious, the controller drops this packet and sets up a blocking rule at the edge switch to drop subsequent packets of this malicious flow (mitigation). If all the first K packets are determined to be benign, the controller installs a forwarding rule for this flow at the edge switch so that subsequent packets of the same flow will be forwarded at switch level.

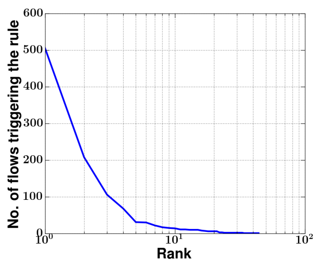
The choice of the K parameter in SDN-Defense depends on the detection performance and the cost incurred. We now characterize how K parameter affects the detection performance in detail.

3.2 Feasibility Study

We use campus WiFi traffic captured in May 2014 for our experiments. The 10.67-hour long trace is 296GB in size and contains over 269 Million packets corresponding to over 5 Million flows. We define flows as unique unidirectional 5-tuples (i.e. protocol, source IP, source port, destination IP, destination port). Figure 1a plots flow size distribution of this dataset. As shown in Figure 1a, 96% of the flows have no more than 100 packets. The average flow size is 53 packets. We run this WiFi traffic



(a) Cumulative distribution of flow size.



(b) Ranked distribution of No. of malicious flows seen per rule.

Figure 1

against open-source Snort detection rules[5]. The trace generates a total of 1770 TCP alerts from 44 different rules (with unique SIDs), triggered by 1145 flows. We rank the rules in decreasing order based on the number of flows triggering each rule. From Figure 1b, we can see that the top 4 most-frequently triggered rules detect more than 75% of the total malicious flows. We will focus on these 4 rules in our discussion later.

3.2.1 *Which packet triggers an alert?*

The question we seek to answer is: how deep do we need to look into a flow to make a decision whether it is malicious or benign? Towards this end, we divide a complete TCP connection into two unidirectional traffic flows: initiating direction traffic (i.e., from client to server) and responding direction traffic (i.e., from server to client). From client to server, the 1^{st} packet is a TCP SYN packet; the 2^{nd} packet is an ACK packet; the 3^{rd} packet is the first data packet and so on. From server to client, the 1^{st} packet is a TCP SYN-ACK packet; the 2^{nd} packet is the first data packet and so on.

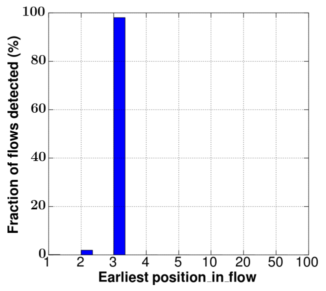
We define *earliest position_in_flow* as the earliest packet position in a TCP flow that triggers a specific security alert. Note that the packet position is counted within unidirectional traffic of a flow. For each of the top 4 most-frequently triggered rules, we analyze all the associated malicious flows and plot the distribution of earliest position_in_flow in Figure 2. Rule 24111, looking for a signature pattern matching against the HTTP header of a packet, is the most-frequently matched rule (encompassing 44.2% of malicious flows). Figure 2 shows that 100% of malicious flows triggering Rule 24111 can be detected within the first 3 packets. The same observation holds for both rule 32847 (ranked 3^{rd}, covering 9.26% of malicious flows) and rule 30260 (ranked 4^{th} covering 5.94% of malicious flows). All these rules look for patterns matching against HTTP header of a packet. Rule 16301, which ranks 2^{nd} (triggered by 18.2% of malicious flows), looks for a signature pattern in the packet payload. As seen in Figure 2, the distribution of earliest position_in_flow spreads out deep into the flow. However, 60% of malicious flows could still be detected within the first three packets.

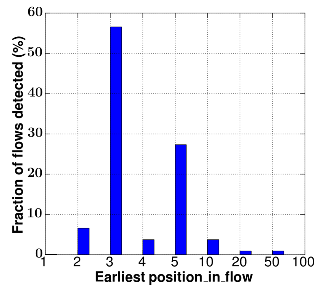
3.2.2 *Detection coverage vs K parameter*

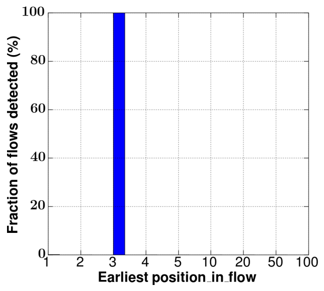
Our observations imply that various security attacks can be detected early in the flow. We define the metric *detection coverage* to quantify the overall success of detection vs the K parameter. The *detection coverage* is defined as the fraction of the number of malicious flows detected within the first K packets over the total number of malicious flows detected after running against the Snort rules. As we will see in Section 4, 73% of malicious flows can be detected within the first 3 packets, and 90% of malicious flows are detected within the first 4 packets of a network flow.

4. COST ANALYSIS

Is offloading security checks to the SDN controller feasible in practice? To answer this question and delineate the associated costs of performing inspection, we compare two cases: (1) SDN reactive routing (SDN-RR) and (2) SDN-Defense in terms of the following two performance metrics: (a) controller response time and (b) controller throughput. We will discuss how configuration parameters such as flow depth (i.e. K) affect the performance of SDN-Defense in Section 4.1, and perform detailed sensitivity analysis with respect to the


(a) SID 24111


(b) SID 16301


(c) SID 32847

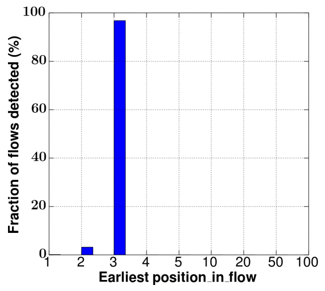
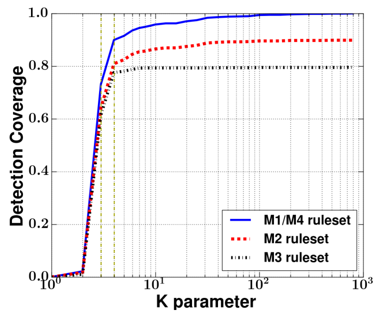
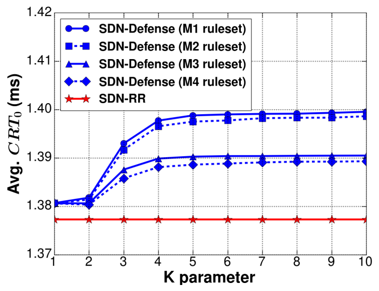

(d) SID 30260

Figure 2: Fraction of flows detected vs the earliest position_in_flow for most-frequently occurring rules.


(a) SDN-Defense detection coverage as a function of K and M_i.


(b) CRT_0 measurements under (i) SDN-RR and (ii) SDN-Defense

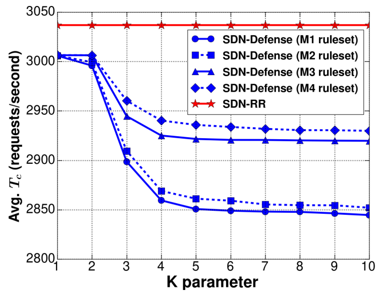

(c) T_c measurements under (i) SDN-RR and (ii) SDN-Defense

Figure 3

two cost metrics in Section 4.2 and 4.3, respectively.

4.1 SDN-Defense Parameters

As described earlier, K represents the flow depth examined by SDN-Defense. With a larger K, more malicious flows are detected by SDN-Defense. However, larger K means more packets are sent to the controller (adding extra load to it) and introducing a longer latency to the network. Therefore, there is a trade-off involved in choosing an appropriate K value for deployment. Further, let M_i denote the subset of Snort detection rules offloaded to the controller site. As in the case of K, the detection can be more comprehensive with a larger subset M_i. However, larger M_i will take longer for a packet to run through the detection engine, incurring higher overhead on the controller. In SDN-Defense, K and M_i are the two tunable parameters for balancing between cost and detection accuracy (benefit). The choice of which ruleset M_i would be offloaded to the controller depends on (a) the security policy, (b) traffic load, and (c) the traffic characteristics of the managed network. To perform a fair comparison, we utilize the rule subsets shown in Table 1 in our experiments. The $\%Coverage$ in Table 1 is defined as the percentage of malicious flows in our dataset detected by a given rule subset. Network managers can dynamically update and modify the choice of these rulesets offloaded to SDN-Defense based on the operational conditions.

M_i	**Description**	$\|M_i\|$	**%Coverage**
M_1	All Snort 2.9.8.3 rules [5].	12.3k	100
M_2	Highest priority rules.	11k	89.9
M_3	Rules scanning http headers.	1750	79.7
M_4	Rules triggered by WiFi traffic.	53	100

Table 1: Rule subsets M_i offloaded to the controller. M_4 can be obtained from historical knowledge.

Figure 3a shows how the detection coverage varies for different choices of K and M_i. Note that M_1 and M_4 share the same characteristic since M_4 includes all the rules which were seen in campus traffic. It is interesting to note that with the choice of $K = 4$, we can detect ~81% of malicious flows with the highest priority ruleset M_2, and ~78% of malicious flows with the http-header ruleset M_3. Further, when using the full ruleset M_1, we can detect up to 90% of all threats with $K = 4$. These results demonstrate the feasibility of offloading subsets of Snort rules to the controller while maintaining good detection coverage. We now elaborate on the impact of SDN-Defense on the controller response time.

4.2 Controller response time

In an SDN network where flow setup is performed reactively, the controller response time[2] directly affects the flow completion time. We define the metric, *zero-load controller response time* (CRT_0), as the controller's processing time for handling a single *packet_in* message

[2]The processing time between the switch sending a *packet_in* message to the controller and receiving a corresponding controller response message.

when there is no queuing delay (i.e., at minimal load). We use this metric to quantify the processing complexity of the controller.

To measure CRT_0, we modify the Cbench tool [2] to send real *packet_in* messages generated using campus traffic to the controller one by one. The Cbench tool emulates the communication between OVS and the SDN controller. The controller application is developed in POX [4]. We then record the time it takes to receive the corresponding response messages from the controller. To eliminate queuing delay from the measured response time, we maintain one *packet_in* request on the fly and wait for response before firing the next request. This helps us clearly delineate the delay introduced by SDN-Defense security checks at the controller.

Figure 3b shows the average CRT_0 for (a) SDN-RR and (b) SDN-Defense with different choice of K parameter and different rulesets. Note that CRT_0 is the average controller response time for a packet going through controller plus the average Snort processing time per packet. As shown in Figure 3b, the CRT_0 increases switching from SDN-RR to SDN-Defense at $K = 1$ due to the additional delay introduced for running packets through the detection engine. Furthermore, we observe a difference in CRT_0 when using different rulesets M_i. This is expected, as it takes a shorter time for a packet to run through the detection engine with a smaller number of offloaded rules and vice versa.

When K is small ($K = 1, 2$), the packets sent to the SDN controller are mostly handshaking packets, which are relatively short and these can only trigger lightweight non-payload rules offloaded to SDN-Defense. When K is larger ($K >= 3$), larger packets with payload data are sent to the controller and it takes a longer time for the detection engine to process these packets (i.e. pattern matching with payload rules). Note that CRT_0 increases with an increase in K, however, for $K >= 4$ there is very little growth in CRT_0. SDN-Defense introduces as little as 1.5% CRT_0 overhead compared to SDN-RR and achieves 81% detection coverage when offloading M_2, and 90% detection coverage with M_1 offload. Furthermore, the CRT_0 overhead for SDN-Defense is as small as ~0.94% when using smaller rulesets M_3 - this will provide up to 78% detection coverage. The operational choice of K and M_i should also take into account their impact on controller throughput, as we will discuss next.

4.3 Controller throughput

In this section, we examine the impact of SDN-Defense on controller throughput. We define controller throughput (T_c) as the maximum number of requests the controller can handle per unit time. To measure T_c, we modify the Cbench tool to encapsulate packets from campus traffic in OpenFlow *packet_in* messages. Then, we issue these *packet_in* requests to the controller at the maximum possible rate without packet drops. Finally, we record the number of response messages received from the controller in unit time and compute the throughput T_c. Note that although such traffic replay may not preserve the internal interaction between the packets of a TCP connection (e.g. a TCP SYN packet needs to be received by the server before it sends out the TCP SYN-ACK packet), it is sufficient to quantify the impact of SDN-Defense on the controller throughput.

Figure 3c shows the T_c measurements for (a) SDN-RR and (b) SDN-Defense under various choices of K parameter and different rulesets M_i. As expected, the throughput drops when switching from SDN-RR to SDN-Defense at $K = 1$. Further, T_c decreases with increasing K for any offloaded ruleset. However, it is important to note that the decrease in throughput due to SDN-Defense is as low as 3.0% when using $K = 3$ and the M_3 (http-header) ruleset. This configuration will provide us up to 61.5% detection coverage. Furthermore, the drop in throughput is as small as 5.8% when offloading the full Snort ruleset M_1 with $K = 4$ (this gives us 90% detection coverage). These results are promising and demonstrate the lightweight nature of SDN-Defense.

In practice, there is no "right" or "wrong" settings for configuration parameters K and M_i. Network administrators can choose desirable K values and M_i depending on security goals and performance requirements of the network where SDN-Defense is deployed. The operational choice of K can also be guided by the ruleset M_i being offloaded to the controller. For instance, if an administrator would like to offload only lightweight http-header rules M_3 to the controller, the K value can be set lower since http-header information is available earlier on in the flow. The choice of the ruleset M_i depends on the security policy, traffic load, and traffic characteristics as enumerated earlier.

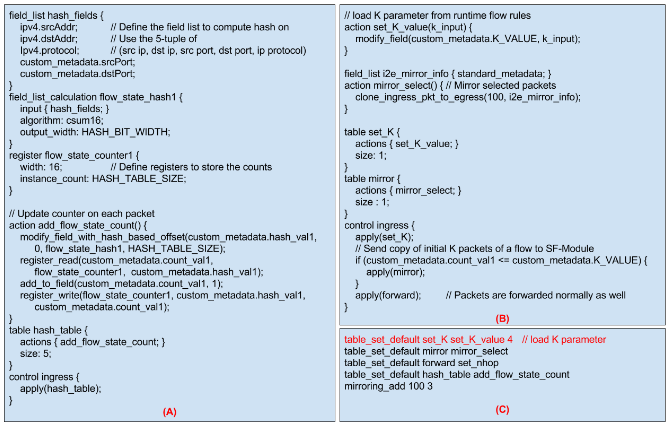
The characteristics presented in Figure 3 can serve as a guideline for administrators to help tune SDN-Defense parameters to find an operational sweet-spot. Furthermore, the flexibility of SDN can be exploited to adaptively change the K values and the ruleset M_i. For instance, the administrator can set a network policy to increase K if the network is lightly loaded and to a smaller M_i if the network load exceeds a given limit.

5. DISCUSSION

We now briefly discuss controller scalability issues and other applications which can benefit from the piggybacking approach. For a more detailed discussion of these applications and issues such as evasion by a malicious adversary, we refer the reader to our detailed technical report [13].

5.1 Scalability

SDN-Defense piggybacks on reactive routing to perform security checks on the first few packets sent to the controller. In practice, reactive routing is not widely deployed in wide-area networks, because (1) The controller becomes the bottleneck under large traffic load. (2) Having packets going through the switch-controller-



```
field_list hash_fields {
   ipv4.srcAddr;          // Define the field list to compute hash on
   ipv4.dstAddr;          // Use the 5-tuple of
   Ipv4.protocol;         // (src ip, dst ip, src port, dst port, ip protocol)
   custom_metadata.srcPort;
   custom_metadata.dstPort;
}
field_list_calculation flow_state_hash1 {
   input { hash_fields; }
   algorithm: csum16;
   output_width: HASH_BIT_WIDTH;
}
register flow_state_counter1 {
   width: 16;             // Define registers to store the counts
   instance_count: HASH_TABLE_SIZE;
}

// Update counter on each packet
action add_flow_state_count() {
   modify_field_with_hash_based_offset(custom_metadata.hash_val1,
       0, flow_state_hash1, HASH_TABLE_SIZE);
   register_read(custom_metadata.count_val1,
       flow_state_counter1,  custom_metadata.hash_val1);
   add_to_field(custom_metadata.count_val1, 1);
   register_write(flow_state_counter1, custom_metadata.hash_val1,
       custom_metadata.count_val1);
}
table hash_table {
   actions { add_flow_state_count; }
   size: 5;
}
control ingress {
   apply(hash_table);
}
                    (A)
```

```
// load K parameter from runtime flow rules
action set_K_value(k_input) {
   modify_field(custom_metadata.K_VALUE, k_input);
}

field_list i2e_mirror_info { standard_metadata; }
action mirror_select() { // Mirror selected packets
   clone_ingress_pkt_to_egress(100, i2e_mirror_info);
}

table set_K {
   actions { set_K_value; }
   size: 1;
}
table mirror {
   actions { mirror_select; }
   size : 1;
}
control ingress {
   apply(set_K);
   // Send copy of initial K packets of a flow to SF-Module
   if (custom_metadata.count_val1 <= custom_metadata.K_VALUE) {
      apply(mirror);
   }
   apply(forward);        // Packets are forwarded normally as well
}
                    (B)
```

```
table_set_default set_K set_K_value 4    // load K parameter
table_set_default mirror mirror_select
table_set_default forward set_nhop
table_set_default hash_table add_flow_state_count
mirroring_add 100 3

                    (C)
```

Figure 4: Sample P4 code with A calculating 5-tuple hash for incoming packets and updating counters; B compares current counter with K parameter and if less, copy of packet is sent to the mirroring port; C shows runtime table entries.

switch loop introduces additional end-to-end latency. There are several potential solutions to resolve the scalability issues here.

In the packet mirroring approach [16], the controller first proactively pushes rules into the SDN switches. These rules direct switches to perform normal packet forwarding while simultaneously instructing them to send a copy of the packet to the controller or a separate detection engine. The controller then decides when to stop mirroring by reactively installing a fine-grained flow entry with a higher priority, which overrides the mirroring rule. We note that using packet mirroring forwards packets as normal, hence negligible additional end-to-end latency is introduced with this approach. However, due to the limitation of OpenFlow, simple flow level logic such as identifying initial K packets of a flow (for later security processing), requires the involvement of the controller, potentially making the controller the bottleneck. In practice the controller may end up receiving more than K packets before the corresponding reactive rule to stop mirroring gets installed at data plane. This problem exacerbates with increasing traffic load.

Emerging switches that are programmable using languages like P4 [3, 1], provide new capabilities, such as flexible parsing, hashing over parsed fields, and supporting registers to maintain stateful flow records. Utilizing these new capabilities, P4-enabled switches can be programmed to identify the initial K packets per flow and mirror a copy of them to the controller. Example P4 code to achieve this is shown in Figure 4. It remains to be seen however, how widespread the deployment of P4 switches will turn out to be in reality. Alternatively, we can integrate flow sampling into SDN-Defense to accommodate large traffic load. For example, using IP-block based flow sampling ensures traffic from the important IP blocks are examined under large traffic load.

5.2 Other Applications

Application Identification: Bernaille et al. [6], [7] in their work, show that it is possible to perform early application identification with high accuracy (>90%) using only the first four or five packets of a network flow. Inspired by such results, we envision that a piggybacking based application identification service running on the SDN controller could achieve accurate and on-the-fly application classification. The features required for classification can be obtained in an online fashion with the arrival of each reactively routed piggybacking packet at the controller. Utilizing such a mechanism presents two clear benefits, (a) Network administrators can gain immediate and accurate visibility into the traffic mix of their network thanks to the global view of the controller, while eliminating problems occurring due to traffic blind-spots. (b) The dynamic control capabilities offered by SDN when coupled with online application identification can enable administrators to rapidly react to changes in traffic mix in their network.

Dynamic Traffic Dispersion Graph (TDG) Generation: TDG's are effective tools to answer 'Who talks to whom?' type questions, performing anomaly detection [9] and other network monitoring tasks. Traditionally, TDG's have been generated by performing expensive trace collection at the network edge followed by offline analysis. We propose utilizing the packets available via reactive routing for *free* to dynamically generate TDG's at the SDN controller. These TDG's can be generated periodically or on-demand with connectivity information obtained in the initial reactive routing packets. Thanks to the global view of the SDN controller, the TDG's generated by this methodology will offer a more comprehensive picture for network administrators and SDN applications which utilize TDG's. This approach takes away issues of traffic blind-spots associated with collection of traces at the network edge and minimizes data collection overheads while allowing for an online and dynamic way to generate TDG's.

6. CONCLUSION

In this paper, we explored the feasibility of piggybacking on SDN reactive routing to support network functions e.g., IPS. Our investigation with campus WiFi traffic revealed that upto 90% of malicious flows are detectable with examination of just the first 4 flow packets. Using these insights, we prototyped and evaluated a piggybacking based intrusion prevention system called SDN-Defense. We showed that SDN-Defense can achieve upto 90% detection coverage with a minimal (5.8%) drop in controller throughput by examining just the four initial packets of a flow. We also discussed design issues such as controller scalability and presented two other applications which can benefit from the proposed piggybacking approach. We elaborate on these applications and security issues such as evasion in our detailed technical report [13].

7. REFERENCES

[1] Barefoot networks :https://www.barefootnetworks.com/technology/.

[2] Cbench: an OpenFlow controller benchmarker. https://github.com/mininet/oflops/tree/master/cbench.

[3] Netronome: https://www.netronome.com.

[4] Pox: https://github.com/noxrepo/pox.

[5] Snort: https://snort.org.

[6] L Bernaille, R Teixeira, and K Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, page 6. ACM, 2006.

[7] Bernaille et al. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[8] Giotis et al. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62:122–136, 2014.

[9] Le et al. Traffic dispersion graph based anomaly detection. In *Proceedings of the Second Symposium on Information and Communication Technology*, pages 36–41. ACM, 2011.

[10] Liao et al. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[11] Xing et al. Sdnips: Enabling software-defined networking based intrusion prevention system in clouds. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 308–311. IEEE, 2014.

[12] Yoon et al. Enabling security functions with sdn: A feasibility study. *Computer Networks*, 85:19–35, 2015.

[13] C Liu, A Raghuramu, C-N Chuah, and B Krishnamurthy. Piggybacking network functions on sdn reactive routing: A feasibility study. https://www.dropbox.com/s/5gi0toqqkd89lt4/piggybacking-network-functions-v10.pdf?dl=0, 2016.

[14] S Mehdi, J Khalid, and S Khayam. Revisiting traffic anomaly detection using software defined networking. In *Recent Advances in Intrusion Detection*, pages 161–180. Springer, 2011.

[15] A Papadogiannakis, M Polychronakis, and E Markatos. Improving the accuracy of network intrusion detection systems under load using selective packet discarding. In *Proceedings of the Third European Workshop on System Security*, pages 15–21. ACM, 2010.

[16] Y. Wang, C. Orapinpatipat, H. Gharakheili, et al. Telescope: Flow-level Video Telemetry using SDN. In *Proc. of EWSDN*, 2016.