A Framework for Improving Network Resilience Using SDN and Fog Nodes

Amir Modarresi*, Siddharth Gangadhar*, and James P.G. Sterbenz*†‡
amodarresi|siddharth@ittc.ku.edu, jpgs@{ittc.ku.edu|comp.{lancs.ac.uk|polyu.edu.hk}}

*Information and Telecommunication Technology Center
Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
www.ittc.ku.edu/resilinets

†School of Comp. and Comm. (SCC) and InfoLab21
Lancaster University, LA1 4WA, UK
‡Computing, The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

***Abstract*—The IoT (Internet of Things) is one of the primary reasons for the massive growth in the number of connected devices to the Internet, thus leading to an increased volume of traffic in the core network. Fog and edge computing are becoming a solution to handle IoT traffic by moving time-sensitive processing to the edge of the network, while using the conventional cloud for historical analysis and long-term storage. Providing processing, storage, and network communication at the edge network are the aim of fog computing to reduce delay, network traffic, and decentralise computing.**

In this paper, we define a framework that realises fog computing that can be extended to install any service of choice. Our framework utilises fog nodes as an extension of the traditional switch to include processing, networking, and storage. The fog nodes act as local decision-making elements that interface with software-defined networking (SDN), to be able to push updates throughout the network. To test our framework, we develop an IP spoofing security application and ensure its correctness through multiple experiments.

***Index Terms*—Resilience, survivability; Future Internet, IoT; SDN, OpenFlow; Cloud and fog computing, OpenFog; Network security, IP spoofing, intrusion detection; Mininet network simulator**

I. Introduction

Cloud computing operates as the back-plane of a cloud-based Internet of Things (IoT) architecture to process and store data generated at the edge network. Furthermore, increasing the number of edge nodes due to the emergence of IoT devices leads to greater traffic in the core network. Although all generated traffic is not part of the Internet services that need fast response time, low delay and jitter are two necessary requirements for interactive and mission-critical services. Unfortunately, delay is not a predetermined attribute in the core network and transferring more traffic saturating the network impacts this attribute directly. Furthermore, the growth of IoT nodes and their computing and storage requirements stresses data center capacity; therefore, a solution is required to decrease these effects. Fog computing is a horizontal, and possibly multi-layer architecture to provide computing, storage, and networking capabilities close to the edge network [1]. While there is no standard model for fog computing, the OpenFog Consortium [2] has introduced a model for this architecture. Throughout this paper, we use the OpenFog architecture as the reference model for our study. OpenFog does not consider fog computing as a separate architecture from cloud computing, but rather recognises fog computing as a *cloud-to-things* continuüm [1]. In other words, fog computing is an extension of cloud computing. Based on this architecture, some services run in the cloud, while others are more apt to run in the fog.

Software-defined networking (SDN) [3], [4], [5] is another emerging technology that moves the control plane of network switches to a centralised controller. In this technology, switches are not responsible to generate and update forwarding tables, rather they receive this information from a controller in the network. For each missing destination in the forwarding table of a particular switch, it contacts the controller to get and update its forwarding table. Such an architecture enables programmability in the control-plane architecture. Centralised control is another SDN attribute that facilitates network management. We use SDN and fog nodes in this paper to decide, redirect, or drop traffic based on incoming rate and other attributes of the flow. The close proximity of the fog nodes to the edge leads to retrieving accurate information about each flow and its corresponding node. The fog nodes send the proper information to the controller to issue redirection or other commands to a specific node. While it is easily possible to impose new policies in SDN on the fly, it is not practical in conventional (non-programmable) switches. This solution can be adopted for blocking network traffic during an attack or redirecting traffic to another possible path when the edge network is under a challenge, improving network resilience.

We define *resilience as the ability of the network to provide an acceptable level of service in the face of faults and challenges to its normal operation* [6], [7]. As defined in ResiliNets [7] disciplines, our model to express network resilience, security is a major subdomain of resilience and is part of a larger field called trustworthiness. Security is defined as the property of a system and the measures taken such that it protects itself from unauthorised access or change, subject to policy [8]. Security can also be considered to be related to

978-1-5386-0671-1/17/$31.00 ©2017 IEEE

self-protection, which is one of the core pillars of resilience.

This paper describes how fog computing and SDN can work together to improve the resilience of the network. Fog nodes have the ability to perform local packet-processing operations and send the end result through control packets signalling the SDN controller for necessary actions to be performed on the network. As part of this work, we introduce a framework for such integration to take place. To show our framework in action, we concentrate on a particular example service, IP spoofing prevention. The fog node helps detect the spoofed packet and passes on a *dropflow* message containing the spoofed source IP address to the controller. The controller, upon receiving the message, drops the flow containing the corresponding parameters for a fixed duration of time. This framework can be easily extended to accommodate complex applications such as traffic management, traffic monitoring, or parental control.

The rest of this paper is organised as follow. In Section II, we review the new architectures and models in both fog computing and SDN. In Section III, we introduce the framework of our model. Section IV comprises our simulation experiments and the results. Finally, we conclude our paper in Section V.

II. Background and Related Work

To the best of our knowledge, Cisco adopted the term *fog* [9] for the first time in its IoT reference architecture. In this model, illustrated in Figure 1, fog computing has been located in the third layer of a seven-layer architecture equivalent to *edge computing*. According to this model, the function of fog computing is converting network data into information suitable for higher-level processing and storage. The first objective of the fog layer is performing information processing close to the edge network. However, Cisco mentions that this layer performs packet-by-packet information processing and there is no awareness about *sessions* or *transactions*. Therefore, this processing is limited to evaluating data based on some criteria, formatting and reformatting, expanding and decoding, summarisation, and assessment to compare with some thresholds or alerts [9]. After this reference architecture, the concept of fog computing has been used with different names including *edge computing*, *cloud at the edge*, or *mobile cloud computing*; however, there are some differences among all of these terms. For example, OpenFog has clearly expressed that fog computing differs from edge computing. The fog works with the cloud while edge computing is an exclusion of the cloud limited to few layers [1].

Cloud at the edge has been proposed as a solution by introducing private clouds and mini-clouds at the edge [10]. The idea of resource sharing is used to expand the layer among nodes in this architecture.

Cloudlet [11] is a similar solution to *cloud-at-the-edge* by using private nodes in public areas between the edge network and the cloud to reduce network traffic and delay in cellular networks. By installing resource-rich network nodes that can run many simultaneous virtual machines, users at the edge

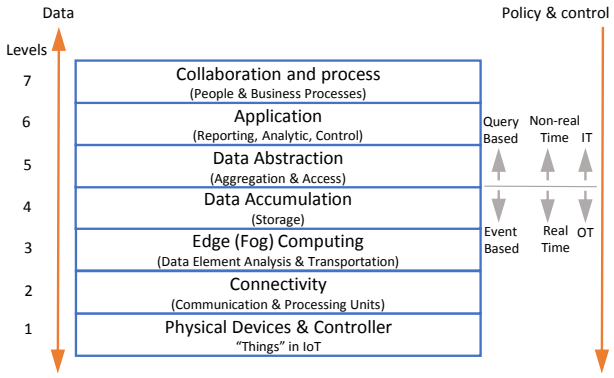

Fig. 1. Cisco reference architecture for IoT

request cloud services on these machines. Virtual machines are created upon the user request and destroyed upon service completion. Wireless LAN is used when available in this model instead of cellular network protocols such as 3G/4G to reduce delay and network traffic in the cellular networks.

Another similar concept has been introduced for cellular networks, called *mobile edge computing* (MEC) [12]. In this model, MEC servers are attached to the base stations to perform user requests. Once the MEC servers fulfill user requests, the results return to the user without sending any request to the cloud, otherwise the the user request is sent to the cloud. This architecture has the same purpose as cloudlets, while cloudlets do not use the cellular network when the wireless (non-telephony) network is available.

The OpenFog reference architecture (RA) [1] is currently the most complete model to represent fog computing. This architecture is based on a set of core principles including security, scalability, openness, autonomy, RAS (reliability, availability, and serviceability), agility, hierarchy, and programmability. These attributes are required to implement a horizontal-layer, distributed system with processing, storage, control, and network facilities as the fog layer. Figure 2 illustrates an abstract representation of this architecture. The OpenFog RA is based on a *view* and *perspective*: A view represents a structural aspect of the architecture recognising various stakeholders' view, while a perspective is a system attribute that needs to be considered across all layers. The current OpenFog RA represents three views including software, system, and node. The software view is the three top layers of RA from the application-services layer to the node-management layer. The node view contains two bottom layers including the protocol-abstraction layer, and sensor, actuator, and control layer. The system view represents a platform and is a combination of one or more node views integrated with other components to create a platform.

Security is one of the main concerns of fog computing and has been researched recently in SDN. SnortFlow [13] is a Snort-based [14] intrusion prevention system that been proposed for cloud environments. Depending on the attack

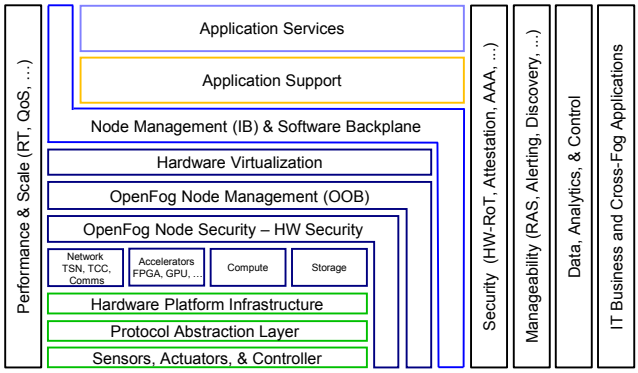

Fig. 2. OpenFog reference architecture [1]

vectors, SnortFlow can dynamically be reconfigured in real-time. The architecture is based on Xen cloud-based environments and OpenFlow switches. A total of three modules exist for the SnortFlow server: a SnortFlow daemon for collecting data from Snort agents, an alert interpreter to parse alerts, and finally, a rule generator that is responsible for generating rules to be pushed to the OpenFlow devices.

Anomaly detection and mitigation systems for SDNs have been proposed [15] that are an architectural design of a framework that promotes anomaly detection and mitigation. The approach is combined with sFlow [15] for flow-based data gathering. Three major components exist in this framework: a collector, an anomaly detection module, and anomaly mitigation. The collector is responsible for gathering flow-related information and this data is produced by the anomaly detection module every few seconds. The anomaly detection module is capable of using any kind of anomaly detection to identify the attacker that is then passed on to the anomaly mitigation module to insert the corresponding flow entries required to mitigate the attack.

A multipath load balancer has been proposed for DDoS (distributed denial of service) attack mitigation [16], performed in two steps: layer-7 load balancing based on DNS/NAT balancing, while layer-4 load balancing splits traffic between paths in the network. The algorithm used for redirection is three-fold: first get the load information from the network, second override the routing with a static Bellman-Ford algorithm, and finally split traffic paths that are either overloaded or have other alternate paths available.

Our solution builds a framework that integrates both fog computing and the security aspects provided by SDN. Fog computing provides the ability to run processing and store data closer to the end users, while these decisions can be broadcasted over the entire network using SDN technology. Details on the framework that helps achieve this is provided in the following section.

III. System Design

A. Fog Attributes of Interest

The proximity of the fog nodes to the edge networks leads to many advantages including recognition of accurate attributes of a flow. Furthermore, considering various attributes of the OpenFog RA makes fog a dependable layer between the edge networks and the cloud. Consequently, the processing power centralised in the cloud is distributed throughout this hierarchical architecture. On the other hand, many *things* in IoT systems, including sensors, do not have sufficient resources for intensive processing. Lack of processing power and energy consideration for battery-operated things limit edge networks to carry out vital operations such as security or optimal decision making. This shifts processing to the fog nodes, the first resource in the network to perform various operations such as context-aware decision making, enforcement of access control for traffic management, or encryption for security. We consider three attributes of the OpenFog RA for our study including security, scalability, and autonomy:

- Security is one of the pillars of the OpenFog RA, considered as a multi-layer security mechanism starting from hardware to any software installation on a particular node. Since fog nodes are the first line of defense for some simple edge networks such as wireless sensor networks, they should be secure in both hardware and software; the ultimate result is a secure path from the edge to the cloud.
- The scalability attribute of the OpenFog RA defines the scalability inside a node and extension to the fog layer. In other words, a node can improve its capability by adding extra hardware or software, or the whole layer can scale up by adding extra nodes. This leads to improving performance by decreasing response time, dependability by adding redundant nodes, and security by using software or hardware accelerators for cryptographic processing.
- The autonomy attribute of the OpenFog RA increases the dependability of the architecture in the face of external challenges such as cable cut or natural disaster. This is due to the autonomy of decision making during challenges. This attribute distributes the centralised decision-making in a cloud architecture without any other layer in-between the fog nodes in various levels of the fog architecture. The autonomy attribute can be expanded to other areas, including autonomy of resource discovery, autonomy of security, and autonomy of operation. Autonomy of resource discovery leads to autonomous decision making for resource sharing and management among other nodes. Autonomy of security allows a node to decide about security issues including blocking malicious traffic that may pass through the node or updating virus scanner databases. Finally, the autonomy of operation increases the ability of a node to support local decision making, including proper response for an actuator in the edge network [1].

We consider these attributes as part of our model architecture.

B. SDN Role in Framework

We use these properties of a fog node to propose our model to increase the resilience of the edge network with lightweight processing. The idea is to use simple and cost-efficient components to improve the overall resilience of the network. In our model, we use SDN switches and simple hosts with limited processing power as a fog node, instead of conventional high-end switches and full-function firewalls. The main intent to use SDN is the flexibility in programming and centralised management of multiple switches. Each SDN switch receives its forwarding table from the controller in the network. Furthermore, one controller is enough to manage multiple switches [17]. If a switch cannot find a path to a particular destination, it sends a request to the controller to receive an updated forwarding table, including the path to the requested destination. The controller is responsible for running routing algorithms to keep the paths updated. We add processing power in the form of a VM to transform each SDN switch into a fog node, in order to be capable of executing lightweight algorithms. Since switches have full visibility to their own subnet, this offers an opportunity to block many attacks, including IP spoofing with simple processing such as comparison of IP addresses. Firewalls installed in the core network have difficulty blocking an IP spoofing attack as they must consider other aspects of a particular flow, including from other layers. In contrast, a switch at the edge network can easily detect the IP prefix of its own network. In this case, any packet that has a different source address than the local network is considered as a spoofed packet.

C. Framework Design

Ideally, fog nodes are devices with processing, networking capability, and storage. As fog nodes have not been realised yet, we consider a fog node as a virtual machine connected to a switch with the processing and storage unit shown as a separate entity.

Our framework is depicted in Figure 3. The network consists of the various hosts shown at the bottom of the figure and includes the fog nodes, which are the integration of an SDN switch and a VM. This assumption holds that the fog node has access to the data that passes through. Each switch in a fog node is connected to the SDN network through the SDN controller, which acts as the main component of the network's control plane. The controller is responsible for pushing flow table entries to the switches.

The end hosts can send packets to each other through the switches, while the SDN controller provides routing information when it is required. The fog node, in turn, inspects any network data that passes through its connected switch. The fog node has access to the entire packet, including the payload, when it is required to make decisions based on the requested network service. The decisions are then relayed through the controller using specially crafted `packet-in` messages, with

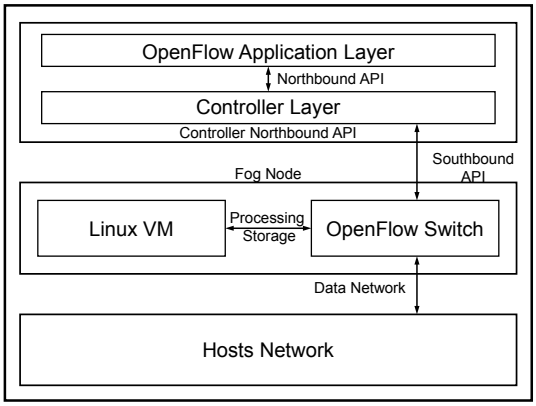

Fig. 3. Fog SDN Architecture Diagram

the action to be performed included in the payload that is then pushed to all switches in the network. The payload can be in the form of a JSON message for standardisation of message exchange between the controller and the fog node. The messages deliver key parameters of the various flow entry attributes that the controller sets to modify flow entries.

Let us consider an IP spoofing experiment with realised through our framework. Spoofed packets originate from a network with a different source address of the attacker and if left unchecked, travel through the network. When the spoofed packets reach the fog-connected switch, the fog VM has a process running that inspects the header of the packet and decides whether the packet is spoofed or not. Legitimate traffic is allowed to pass through to the controller without any action from the fog node, with the controller installing flow rules for the packets to reach the destination. In case the packet is determined to be spoofed, special `packet-in` messages are sent to the controller with the payload formatted as a JSON message. This contains key features depicting the action to be performed such as drop flow, the source IP address, and the destination IP address. The message is received by the controller and the drop action is performed for a specific period of time. The pseudocode for implementing this scenario in a fog node is shown in Algorithm 1.

D. Service Realisation through the Framework

In this section, we outline some of the various services that can be realised through this framework. We consider a few that can be developed with minimal changes to our framework.

A simple traffic management system is possible by monitoring the volume of traffic and the utilisation of the CPU in the switch, as well as packet drop rates in the ingress and egress buffers. In this case, if there is another interface available and buffer overflow occurs for a particular interface, the switch can inform the controller to add an alternate path to the forwarding table. This feature increases the dynamic manageability of the network. However, this condition needs a degree of network context-awareness. In order for a node to be self-aware, it must be able to monitor itself; in our architecture, the fog nodes promote this awareness. Broadly

Algorithm 1 IP spoofing application based on framework

```
1: procedure PACKET SNIFFER
2:     for each packet traversing the network
3: Sniffing phase:
4:     sniff packet occuring on eth0
5:     save the source IP of packet for further processing
6: Analysis phase:
7:     compare packet IP with network of host network
8:     if IP not in network then
9:         spoofed packet
10: procedure IP SPOOFING FRAMEWORK
11:    run packet sniffer on fog node
12:    if packet spoofed then
13:        generate packet-in messages to controller
14:        payload contains action to be performed
15:        controller executes necessary action
16:    else
17:        flow allowed
```

speaking, self-awareness is categorised as either public or private [18], [19]. Public self-awareness refers to knowledge acquired from the environment related to the consequences of a system action. Private self-awareness refers to knowledge internally available to the system. Since a fog node is installed close to the edge network, it has a better view of the local network and is able to make decisions optimally for its own network. Consequently, the decision for traffic management is likely to be more accurate based on the condition of the edge network and neighbor switches.

LAND (local area network denial) attacks [20] in which a malicious node sends a TCP SYN packet with the same source and destination of the target node are also recognisable in this model. In such an attack a node sends packets continuously to itself. This type of attack is easily detectable by a simple comparison when such requests enter the LAN network switch.

If a switch has enough processing power, it can perform decision making to stop malicious traffic causing various types of denial of service (DoS) attacks. For instance, a fog node can gather statistics about daily traffic for further analysis. Using this information, events such as flash crowds and other anomalies can be detected. To this effect, learning algorithms can be used to diffrentiate good and malicious traffic and measures taken accordingly [21].

URL filtering and parental control can be another use of our framework. While switches and routers process layers 2 and 3, URL filtering needs deep packet inspection. If DNS requests are monitored at the switch level, a proper result for filtering can be gained.

Furthermore, having fog nodes at the edge network benefits interactivity and response time, particularly for mission-critical and interactive systems in which edge networks contain limited-resource sensor nodes.

We would like to justify why are model is preferred to other existing solutions. The optimal explanation comes from the main attributes of fog computing and the SDN environment. As mentioned previously, a fog node can obtain more accurate information from the edge network if it is installed in the proper place. This information may be more beneficial if it is shared with other fog nodes. Our solution for sharing this information is using SDN controllers. Since SDN provides a logically centralised management plane for a large network, signalling can be propagated through the network efficiently, with programmability to take appropriate action.

IV. Evaluation

This section describes the setup for our simulations and the various experiments conducted to validate our framework.

A. Simulation Setup

In order to evaluate our model, we emulate a topology as illustrated in Figure 4 using Mininet [22]. Mininet can create a virtual network in the Linux kernel, instantiate OpenFlow switches, and run application code on bare metal or virtual machines. It has the ability to develop experiments with OpenFlow [3], [23] and SDN. Accessing the host kernel allows Mininet users to add their experimental code to the network. In our topology, end hosts at the lower level of the network are able to generate legitimate or malicious traffic. A Python script generates traffic that controls when and what type of traffic should be injected into the network. In order to implement spoofed packets, we change the source IP address to another address that is not part of the local network. Furthermore, malicious traffic can be mixed with legitimate traffic destined to the other parts of the network.

In addition to the hosts, a set of SDN OpenFlow switches is connected to an SDN controller (OFC), from which the forwarding table along with the action related to each flow is obtained. We use POX [24], which is an OpenFlow open-source Python-based platform for SDN control applications, to which we add our code to implement new policies to the controller.

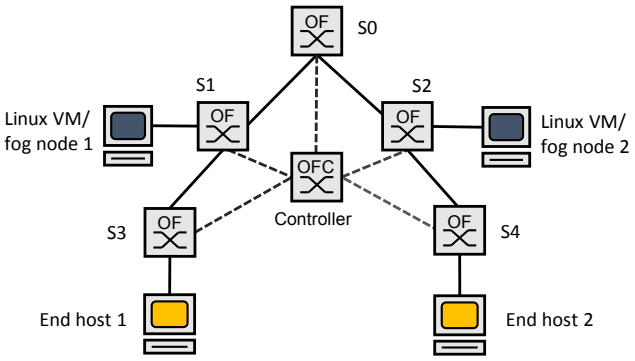

Fig. 4. Simulation topology

The last piece of the topology is the fog nodes that are implemented as Linux VMs, connected directly to the OpenFlow switches. In this case, a copy of all traffic passing through the switch is forwarded to the fog node where the decision making is performed. Therefore, fog nodes have visibility to

TABLE I
SNIFFER VS. NO SNIFFER PACKETS EVERY SECOND

	Second 1	Second 2	Second 3	Second 4
No sniffer packets	45	45	45	45
Sniffer packets	45	0	0	0

their own network. We implement a simple packet sniffer in Python and install it in the fog nodes, similar to the technique that intrusion detection systems (IDS) use to monitor traffic for irregularity. Furthermore, a direct request to the controller can stop malicious traffic or reroute it to a null port. In this case, the fog node that has detected malicious traffic sends a `packet-in` message containing flow information and drop actions. After processing the message in the controller, it updates the forwarding table and pushes to all switches. Receiving the updated table, switches drop the identified flow. This action remains in the forwarding table for a limited amount of time and then is removed. Therefore, it prevents the forwarding table from being overrun by attackers that change the source IP address constantly. If the same attack persists, the fog node sends a new request to the controller. Although the fog node is not inline with the traffic, its ability to send the requests and proximity to the switch permits the fog node work as an intrusion prevention system (IPS). Therefore, the risk of having a long delay to allow the malicious traffic pass the switch is negligible.

B. Experiments

We perform three experiments to validate our IP spoofing application. We utilise the topology specified in Figure 4 for the experiments.

1) IP Spoofing Application Validation: As part of our first experiment, we validate our IP spoofing detection in the presence of only spoofed traffic. For the first part of the experiment, we do not make use of the sniffer application and monitor the spoofed packets for the length of the simulation period. IP spoofed packets are generated from host 1 and travel through the fog node and the controller to host 2; Scapy [25] is used to generate packets. For the second part of the experiment, we introduce the sniffer when the experiment starts, which is run on the fog node 1.

The results are shown in Table I. In the case where there are no sniffer packets, we see a steady rate of spoofed packet generation, sampled at every second for the entirety of the simulation. When the sniffer is introduced, we see that the spoofed packets drop after the first second. This is due to the sniffer detecting the spoofed packet and sending control signals to the controller requesting it to drop the flow with the necessary flow parameters.

2) IP Spoofing Validation in midst of Legitimate Traffic: As part of our second experiment, we validate the working of our IP spoofing application in the midst of legitimate traffic, flowing between host 1 and host 2 for the entirety of the simulation. Spoofed packets are introduced at time 20 s of the simulation for the duration from host 1. The IP sniffer is installed in fog node 1. We consider two cases of study for this experiment: the first when no IP sniffer is employed and the second when an IP sniffer listens to flow information in the network.

The results are provided in Figure 5. The number of packets transmitted over time is plotted against the simulation time for both cases. We see that in the first no-sniffer case, a spike occurs around 20 s. This is due to the extra packets being generated by the spoofing application. In the second case, we observe that the spoofed packets get blocked by the IP sniffer application beginning at 20 s, showing that the sniffer does not affect any legitimate traffic destined for the same host.

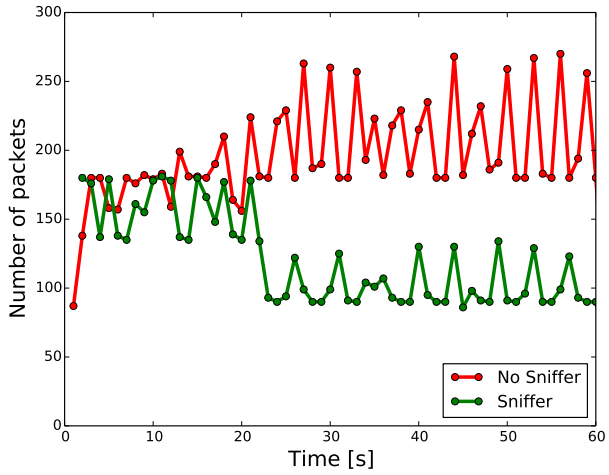

Fig. 5. Sniffer vs. no sniffer in midst of legitimate traffic

3) IP Sniffer Action for Differing Timeout Values: For the final experiment, we focus on the action of an IP sniffer when the duration of the forwarding table entries are low. The forwarding table entries have two types of duration parameters. The idle timeout parameter is a value that triggers when no flows are present for the specified period of time. Hard timeout, on the other hand, is a hard stop after which flow entries expire. When the hard timeout value is low, there are concerns that the spoofed packets will start to reappear. In this experiment, we set a value of 20 s for both the idle and hard timeout and check how the sniffer reacts to this case.

The results are presented in Figure 6. We see that when the sniffer is not in action, there is a surge in the number of spoofed packets starting at simulation time 20 s when the spoofed packets start to transmit. This trend is seen until the end of the simulation. On the other hand, when the sniffer is started, we notice spoofed packets occurring at 20 s and 21 s of the simulation for the first iteration. These spoofed packets are then dropped by the IP sniffer for an idle timeout and hard timeout of 20 s. At the end of 40 s, we see spoofed packets coming back only to be dropped at that instant, showing that the IP sniffer works continuously for the entirety of the simulation.

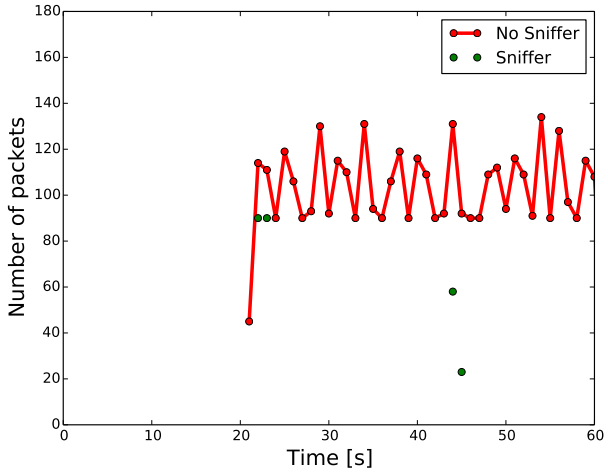

Fig. 6. Sniffer vs. no sniffer for spoofed packets

V. Conclusions

In this work, we have defined a new framework that includes fog nodes integrating with SDNs to improve the resilience of networks. The framework utilises fog nodes that are attached to OpenFlow switches and have the ability to inspect data packets that pass through the network. Services can then be developed based on the scenarios. Messages are then relayed to the controller to install flow rules accordingly across the network. To test the framework, we make use of a simple IP sniffer application that is installed on the fog node. Experiments show the proper working of the detector application, both with and without legitimate traffic. As part of our future work, we intend developing more services such as traffic management systems, and machine learning based intrusion detection on top of the framework.

References

[1] OpenFog Consortium Architecture Working Group, "OpenFog reference architecture for fog computing." https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf, 2017.

[2] OpenFog Consortium Architecture Working Group, "OpenFog." https://www.openfogconsortium.org/, 2017.

[3] Open Networking Foundation, "ONF website." https://www.opennetworking.org/, 2017.

[4] K. Kirkpatrick, "Software-Defined Networking," *Commun. ACM*, vol. 56, pp. 16–19, Sept. 2013.

[5] N. McKeown, "Software-Defined Networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[6] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[7] J. P. Sterbenz and D. Hutchison, "ResiliNets: Multilevel resilient and survivable networking initiative." http://wiki.ittc.ku.edu/resilinets, 2016.

[8] C. Landwehr, "Computer security," *International Journal of Information Security*, vol. 1, no. 1, pp. 3–13, 2001.

[9] Cisco, "The Internet of Things Reference Model." http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf, 2014.

[10] L. M. Vaquero and L. Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 27–32, Oct. 2014.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, Oct 2009.

[12] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile Edge Computing: A Taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*, pp. 48–54, Citeseer, 2014.

[13] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "Snortflow: An Openflow-based Intrusion Prevention System in Cloud Environment," in *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pp. 89–92, IEEE, 2013.

[14] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, (Berkeley, CA, USA), pp. 229–238, USENIX Association, 1999.

[15] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "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.

[16] M. Belyaev and S. Gaivoronski, "Towards Load Balancing in SDN-networks during DDoS-Attacks," in *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International*, pp. 1–6, IEEE, 2014.

[17] Casado, Martin and Garfinkel, Tal and Akella, Aditya and Freedman, Michael J and Boneh, Dan and McKeown, Nick and Shenker, Scott, "SANE: A Protection Architecture for Enterprise Networks.," in *USENIX Security Symposium*, vol. 49, pp. 137–151, 2006.

[18] C. Goukens, S. Dewitte, and L. Warlop, "Me, Myself, and My Choices: The Influence of Private Self-Awareness on Choice," *Journal of Marketing Research*, vol. 46, no. 5, pp. 682–692, 2009.

[19] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, "A Survey of Self-Awareness and Its Application in Computing Systems," in *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pp. 102–107, Oct 2011.

[20] G. A. Marin, "Network security basics," *IEEE security & privacy*, vol. 3, no. 6, pp. 68–72, 2005.

[21] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.

[22] "An Instant Virtual Network on your Laptop (or other PC)." http://www.mininet.org, July 2010.

[23] 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.*, vol. 38, pp. 69–74, Mar. 2008.

[24] "The POX Controller." https://github.com/noxrepo/pox, July 2010.

[25] P. Biondi, "Scapy Project." http://www.secdev.org/projects/scapy/, 2011.