Dynamic Orchestration of Security Services at Fog Nodes for 5G IoT

Vashish N. Imrith*, Pasika Ranaweera†, Rameshwar A. Jugurnauth‡, Madhusanka Liyanage§
*‡Department of Electrical and Electronics, University of Mauritius, Mauritius
†§School of Computer Science, University College Dublin, Ireland
§Centre for Wireless Communications, University of Oulu, Finland
Email: *vashish33@gmail.com,†pasika.ranaweera@ucdconnect.ie, ‡r.jugurnauth@uom.ac.mu , §madhusanka@ucd.ie,
§madhusanka.liyanage@oulu.fi

***Abstract*—Fog Computing is one of the edge computing paradigms that envisages being the proximate processing and storage infrastructure for a multitude of IoT appliances. With its dynamic deployability as a medium level cloud service, fog nodes are enabling heterogeneous service provisioning infrastructure that features scalability, interoperability, and adaptability. Out of the various 5G based services possible with the fog computing platforms, security services are imperative but minimally investigated direct live. Thus, in this research, we are focused on launching security services in a fog node with an architecture capable of provisioning on-demand service requests. As the fog nodes are constrained on resources, our intention is to integrate light-weight virtualization technology such as Docker for forming the service provisioning infrastructure. We managed to launch multiple security instances configured to be Intrusion Detection and Prevention Systems (IDPSs) on the fog infrastructure emulated via a Raspberry Pi-4 device. This environment was tested with multiple network flows to validate its feasibility. In our proposed architecture, orchestration strategies performed by the security orchestrator were stated as guidelines for achieving pragmatic, dynamic orchestration with fog in IoT deployments. The results of this research guarantee the possibility of developing an ambient security service model that facilitates IoT devices with enhanced security.**

***Index Terms*—IoT, Fog Nodes, IDPS, Security Services, Scalability, Performance, Orchestration**

I. Introduction

The Internet of Things (IoT) emergence made a big impact on daily life with the arrival of micro and nanodevices. These devices inheriting autonomous intelligence are used to simplify personal lives by improving the efficiency of digital infrastructure. The extent of IoT applications ranges from households, business environments, agricultural sites, sporting events to automobiles [1]. Typically, IoT systems consist of sensors, actuators, networking nodes, intermediary data storage, and processing centers, interfacing devices, and remote clouds. Most of the IoT devices are miniature and battery energized appliances that embed low-level processing and memory components [2]. Thus, reliable and highly scaled service functions cannot be guaranteed by them. Such appliances are employed as remote sensory devices and wearables that are commissioned to extract data and monitor the subjected entities. Extracted data from these sensors are imperative for not only immediate decision making but for identifying patterns, predictive analysis, and forecasting solutions to improve the effectiveness of the smart services [3].

Typically these sensory devices are connected to an IoT gateway that forms a Wireless Sensor Network (WSN). A number of connected gateways depend on the extent of the smart environment/system. As these WSNs are exerting an enormous amount of critical data from the smart environments, securing the transmission channels is in the best interest of achieving an accurate and seamless smart service. The resilience of these devices, including the gateways, however, is lesser for different types of security attacks due to their remote nature. IoT sensors could be tampered by physical means. There is a higher tendency for the injection of malicious nodes among sensors in WSNs. The household components such as CCTV cameras and doorbells are hackable for gaining habitual and personal information [4], [5]. Botnets are the most recent threats that emerged for emanating Distributed Denial of Service (DDoS) attacks on WSN based IoT gateway nodes for disrupting their services [6].

These vulnerabilities in IoT sensors are creating different security requirements that could not be mitigated by a singular security function. Launching diverse security functions in a WSN composed of IoT devices and a gateway is questionable due to their resource constraints. However, with edge computing initiatives such as fog computing, an edge infrastructure is envisaged to deployed among the IoT systems [7]. Thus, this research is intending to explore the possibility of launching multiple-security services at a resource-constrained edge node.

A. Contribution

In this paper, we aim to investigate the feasibility of launching multiple Intrusion Detection and Prevention System (IDPS) tools at a resource-constrained edge node. We are proposing an architecture suited for provisioning security services for IoT devices, centralized by a fog node. This approach enables a user to acquire a preferred security service that is capable of ensuring security and privacy aspects regardless of the service which was originally subscribed to. Moreover, the dynamic nature of the service provisioning fog based platform improves the scalability of IoT applications. The main contribution is the orchestration strategies proposed for

managing the security services in the edge node. In order to implement multiple security services, light-weight virtualization techniques are followed. Moreover, state-of-the-art IDPS tools are configured with the edge device to form the testing environment. We are validating the proposed structure by emulating malicious traffic flows and testing the performance of the edge node.

Rest of the paper is organized as follows. Section II elaborates the background technologies used for this investigation. Prevailing literature are summarized in Section III. Section IV and Section V are proposing and validating the IoT dynamic security architecture. Section VI concludes the paper.

II. Background

A. Fog Computing Edge Nodes

Introduced by Cisco in 2012, Fog computing is a major edge computing paradigm that augments cloud computing services to mobile devices through a decentralized service platform [8]. In this approach, an edge node, referred to as a fog node, is dispensing resources that are limited to the main cloud with proximate accessibility. The fog concept is formalized with the three functional planes of cloud, fog, and IoT/user levels [9]. Unlike other edge computing paradigms, fog network is extended from fog-to-fog connections. Thus, latency endured with fog applications is very low [1]. Service deployment at the fog infrastructure is achieved by virtualization technologies. Due to the proximate placement of fog nodes to the IoT nodes restricts the resources available for them. Thus, fog nodes that are localized to IoT stratum are typically considered as resource-constrained edge nodes.

B. Docker

Docker is a prominent light-weight virtualization technique that is capable of launching at computing platforms with alleviated resources [10]. Its ability to execute multiple service instances with minimal resources enable its integration to emerge virtualization-based systems. Docker is formed as a client-server model, where docker daemon is handling the user requests interfaced by a docker client. The extracted images from the docker registry are executed as containers within the docker host. Each container is given an ID and is capable of committing the status of the container as a newly forged image. These containers are connected with a default bridge network. The overhead is remarkably lesser as 70 MB for a Ubuntu-based container. Thus, deploying multiple containers at an edge device is no longer an arduous task.

C. System on Bare Metal (SoBM) Vs. System on Docker Containers (SoDC)

SoBM is a situation that particular software or a tool is directly running on the edge device. In contrast, the SoDC concept allows running the tools in Docker containers where the Docker engine is orchestrating the light-weight virtualized entities. The wide adaptability of docker as means of launching diverse services with miniature resource-constrained devices, IDPS systems are one such service that researches have considered to embed into containers. Experiments conducted by [11] clearly shows that, apart from network utilization, other factors such as packet processing speed, prompted security alerts, and RAM utilization resembles in both cases. CPU utilization and a number of dropped packets are lesser with SoDC compared to SoBM. Thus, utilizing the SoDC approach for launching diverse functions with fog nodes would allow the entire edge system to control resources optimally. Moreover, as the dockerized environment is operating in the application level, the flexibility, scalability, adaptability, and interoperability attributes are guaranteed.

III. Related Work

Boudi et al. in [11] contributed towards the verification of feasible deployment of resource-constrained edge devices. Their assessment of validating the SoDC systems at edge nodes was the foundation for this research initiative. In this paper, a performance evaluation was conducted employing a Raspberry Pi3 model as an edge device measuring the processed packets, network utilization, CPU load, RAM utilization, number of security alerts, and number of dropped packets when suricata [12], [13] was operating. Though the authors conclude that overhead on the Docker platform is very light, only one docker container was tested with this approach.

Islam et al. in [14], the author, experimented in regards to one issue face, which stated that several IoT devices in use caused high amounts of data where critical system functionalities must be ensured during the access of network. The paper brought edge functions to the local level as virtualized and dynamically deployed components utilizing local hardware capacity. They implemented a local edge networking prototype based on local microservices, called nano-services. The latter was implemented using Docker container and deployed using Docker Swarm-based orchestration. They focused on the optimization of resources of the proposed nano-services. They noticed that multi-stage builds based images show the best performance for the reason that it includes only run-time dependencies. The paper was based on the feasibility study of the virtualized nano-services at the local level of the IoT edge network. The reduction of resource consumption was due to the replacement of base image sizes with multi-stage builds which actually reduced from hundreds of MB's to tens of MB's. They used Alpine image through a run-time container size that was squeezed to a few hundreds of kilo byes with the cost of larger unique container sizes. Due to the multi-stage containers, they were able to reduce the nano-services deployment time, which also resulted in the reduction of initiation time.

Sfirzin et al. in [15] focused on provisioning virtualized security service in resource-limited edge nodes by leveraging lightweight virtualization technologies. The analysis of the paper gave an overview of the container-based security solutions. Thus it provided viable guidelines towards the orchestration of security at the edge. According to the results, the overhead introduced by the containerization for security functions is very light. The Docker container had 100 percent control over

the network interface even though only one Docker container was running the performance evaluation.

Tripathi et al. in [16] investigated the possibility of employing Raspberry Pi as an Intrusion Detection System against cyber-attacks at home environments. The security functions of the proposed model included a honeypot, packet analyzer, and a firewall. The tools Snort [17], Barnyard2, Pulledpork, MySQL, Ruby, Apache2, Cowrie, and Tshark were configured to form the testing environment.

IV. Proposed Architecture

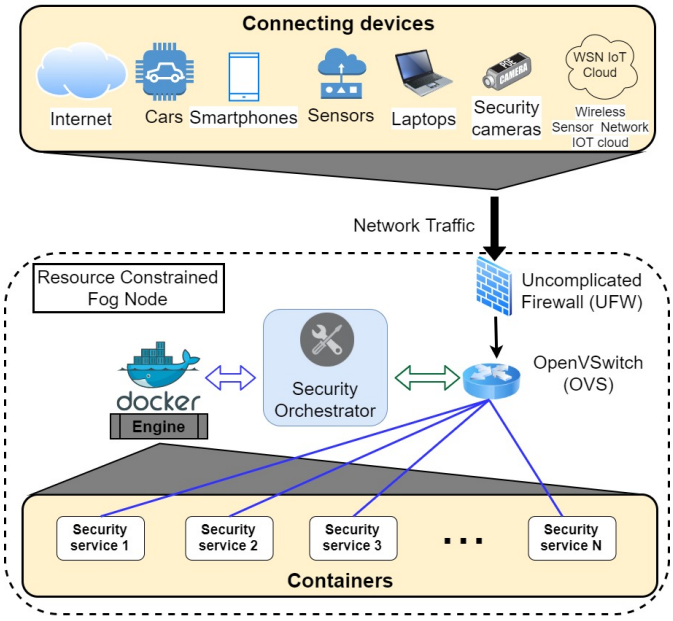

Fig. 1. Dynamic Security Provisioning Architecture for Resource Constrained Fog Nodes

In this paper, we intend to propose an architecture to resource constraint edge nodes, as depicted in Fig. 1. Various security functions hosted as containers are running in the dockerized environment of the edge device. Docker engine is responsible for the creation, retrieval, and termination of containers as in a typical docker implementation. Security Orchestrator (SO), however, is capable of auto-configuring containers in accordance with the intrinsic security service type. Thus, SO acts as an autonomous entity that instructs the Docker engine on running, committing, and stopping security service containers based on the demand. Besides, service and performance statistics of the dockerized environment is monitored by the SO.

It is obvious that the heterogeneous IoT based devices connecting to a fog node require diverse security functions. According to the OpenFog reference architecture presented in [18], a separate layer exists for node management or orchestration. This layer is consisting of hardware virtualization, security, and node management in terms of network, computing, storage, and accelerator resources. Thus, our proposed solution is adhering to the OpenFog architecture (i.e. node management layer) that presents a way to implement orchestration along with security assurance. In our proposed architecture, we are focusing on IDPS systems that are capable of monitoring network intrusions. When multiple devices require provisioning of security services at the same time, it is imperative to distinguish the virtual security instances for efficient processing. Thus, we are proposing to establish a container network that is capable of forming an Internet Protocol (IP) based network. The exiting docker bridge network is limiting its scalability and interoperability. Therefore, we intend to employ OpenVSwitch (OVS) [19] as a virtual router for the fog node. The OVS is creating its bridge network that assigns IP addresses for each container. The security services are distinguished by their assigned IP addresses thereon. Moreover, OVS is auto-configured by the SO for updating the list of security service containers and their IPs. An Uncomplicated Firewall (UFW) [20], [21] function is deployed at the network interface of the fog node. This UFW restricts unauthorized intrusions towards the system.

A. Security Orchestrator (SO)

As explicated earlier, SO is the main entity that manage the processing in the edge node. However, this edge node is performing other services apart from facilitating IDPS based security services. Thus, a maximum amount of resource limit should be maintained by the SO for security service related provisions. In addition to orchestrating docker containers, we propose the following strategies to be governed by the SO.

1) Multiple Device Support: In a circumstance where multiple devices are to be dispensed with security services at the same time, fog node should differentiate the ingress traffic flows. It would be an arduous task to achieve this with a single IDPS instance. Thus, multiple instances of IDPS tools should be launched in parallel to serve the varied flows directed to the fog node. The SO is autonomously tunneling the distinct traffic towards the IDPS containers. The IDPS tool running on the container would be based on the user preference and requested security level. However, in a situation where resources are limited, SO will assign a low resource consuming tool overriding user preference for seamless operation.

2) Performance Optimizing: As an orchestrator, optimizing the resource utilization for maximizing the output is an imperative requirement. Since our fog based edge nodes are provisioning multiple security services in parallel, monitoring the performance metrics in correlation to resource utilization is a task for the SO. SO should include a mechanism to threshold the affordable resource limits for particular containers/ instances based on their performance. If the performance of the particular service instance is weak (i.e., packet drops are higher), service should be assigned to a different tool intending better performance metrics. Moreover, if the ingress data rates are higher and the fog node is on the brink of overloading, SO should transfer the security service to a light resource-consuming service instance.

3) Classification of Ingress Traffic: The emerging diversified services and applications are conveying different types

of traffic through edge devices. These traffic types represent Ultra High Definition (UHD)/HD/ CCTV, Voice over Internet Protocol (VoIP), Augmented Reality (AR), massive Machine Type Communication (mMTC), and Industrial IoT (IIoT) applications that require fog nodes to manage their remote IoT work sites. Each traffic type demands different security levels and responsiveness in terms of alerts. Therefore, classifying the traffic according to their service would lead to better heterogeneity at the fog node. One way of achieving this is to classify the traffic according to their Transmission Control Protocol (TCP)/ User Datagram Protocol (UDP) port numbers. Once the classification is pursued, different traffic classes represented by the port numbers could be forwarded to either single or multiple security instances, chained following the Service Function Chaining (SFC) approaches [22].

4) Dynamic allocation of resources for service instances: In our architecture, service instances are launched as docker containers. These containers could be allocated with a controlled limit of the host system processor and memory. If the existing resources are inadequate for completing the executing process, resources of the subjected container could be improved without running an additional security instance. This scenario would be a lesser resource pruning than running multiple instances.

V. Implementation and Experiment Results

A. Experimental Test Bed

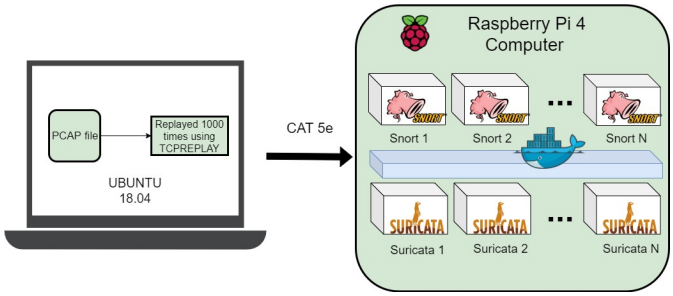

Fig. 2. Experimental Testbed

The testbed was developed with a Raspberry pi 4 Computer model B and a laptop having an Ubuntu 18.04 OS. As illustrated in Fig. 2, Raspberry pi was connected to the traffic emulating laptop with a CAT 5e Ethernet cable. Both devices were operated in offline mode to restrict the packets flowing from other sources to the shared interface. To enhance the accuracy of the experiment, the above practices were maintained throughout the experiment. The TABLE I shows the specification of the testbed appliances in terms of hardware and software. Note that the Raspberry pi uses the USB bus 2.0 chip and the LAN chip with a network interface card. The maximum achievable rate for the USB console is 100 Mbps.

B. Network Traffic Emulation

The alerts were generated from pcap files that were publicly available from [15]. The 1st pcap file contained larger files [23] where the 2nd pcap was containing small packets [24]. These pcap files were inclusive of malware and possible viruses. The intention was to check the rules of Snort and suricata and the number of resources that were being consumed. The pcap files have a more significant impact on the CPU. The pcap file that contains small packets will cause the CPU to have a higher workload compared to one that has larger packets. Small packets have the benefits of better response time and less error rate. Though, the overhead is higher and causes more CPU usage. If the number of packets per second is high, the receiver has to work a lot to accept all of them. Since the latter needs to work quickly, it produces CPU interrupts. Therefore, using two different sized pcap files, we expect to get different results.

TABLE I
Specifications of the Experimental Testbed

	Laptop	Raspberry pi 4
CPU	core i3 2.2GHz	Quad Core Cortex A72 1.5GHz
RAM	4GB DDR4	4GB LPDDR4
OS	Ubuntu 18.04	Raspbian Debian Buster
Connectivity	100 Mbps Cat5e	

C. IDPS Rules

Rules play a vital role in an IDPS. The most crucial part comes when the user has to choose the number of rules that are required. The rules are available on the snort websites. The registered rules contain around 12,000 rules. The number of rules used here was kept as default for the registered rules. We did not want to have the set of rules edited since our goal was to test the maximum resource usage of snort and suricata. We have used suricata 5.0.0 and snort 2.9.15 with registered rule-set.

D. Testing Scenarios

Testing scenarios were designed to validate the orchestration strategies of the SO. In this scope of the research; however, we are verifying the multiple device supporting strategy only.

1) Performance of NIDS tools with data rate of traffic: In order to identify the optimum performance of the two tools, they were exposed to an incoming emulated traffic flow that ranges from 10 Mbps to 80 Mbps. Results for Snort and Suricata are displayed in Fig. 3 and Fig. 4 respectively. According to the results, optimum data rates for snort and suricata are 55 Mbps and 30 Mbps, respectively.

2) Comparing the performance and resource utilization with multiple instances of the same IDPS tool: In this testing scenario, the effect of running multiple containers in parallel with the same IDPS is experimented to determine the performance and resource metrics for suricata and snort.

a) suricata: According to Fig. 5, CPU and RAM utilization are accumulating gradually. The alert percentage and drop rate do not vary significantly. The metric performance changes in Fig. 6 is also insignificant. However, resource parameters are incrementing with the data rate.

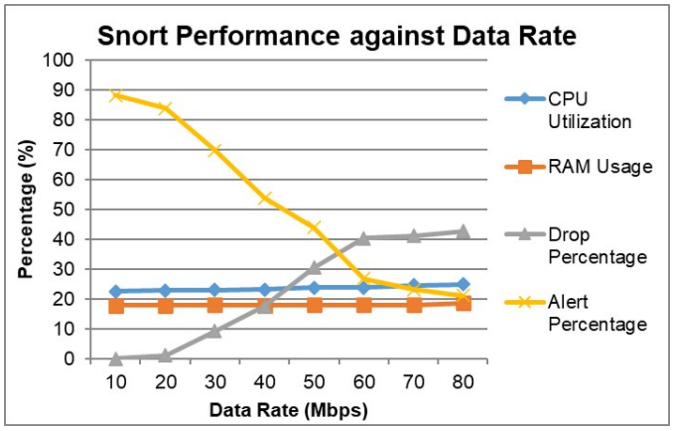

Fig. 3. Snort Performance compared to Data Rate

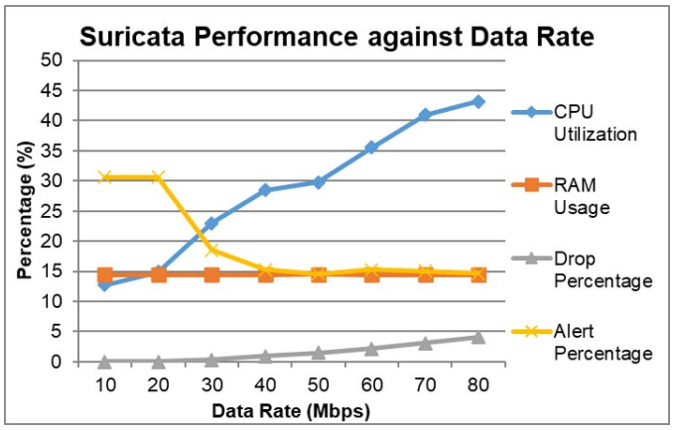

Fig. 4. suricata Performance compared to Data Rate

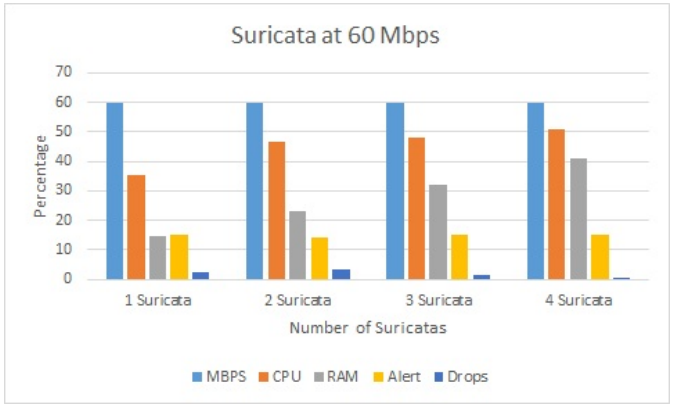

Fig. 5. Statistics with different suricata instances at 60 Mbps

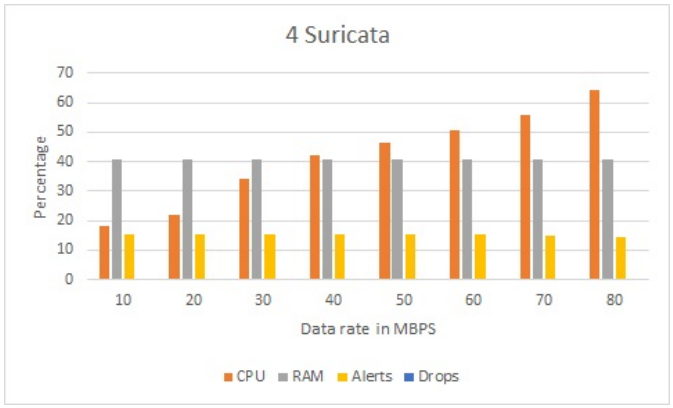

Fig. 6. Statistics with 4 suricata instances with varied data rates

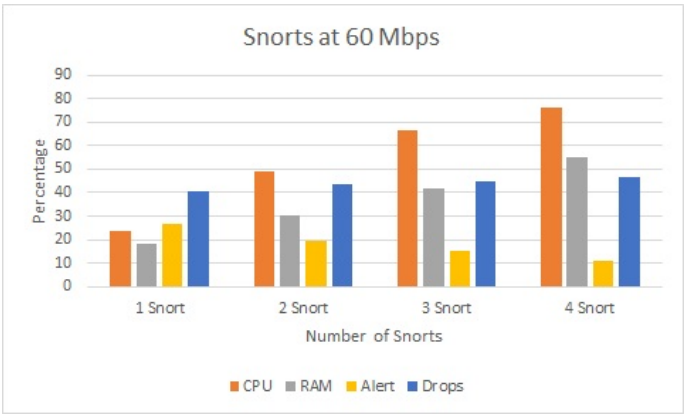

Fig. 7. Statistics with different snort instances at 60 Mbps

b) snort: Fig. 7 depicts the statistics with multiple Snort instances. Even though the CPU and RAM are conspicuously incremented over data rate, alert percentage reduces while the drop rate is approximately consistent. This shows different results than suricata's performance. Thus, we can observe that detection of malicious alerts with snort is highly reliant on the CPU allocation. Fig. 8 shows that alerts are reduced with the increasing data rate.

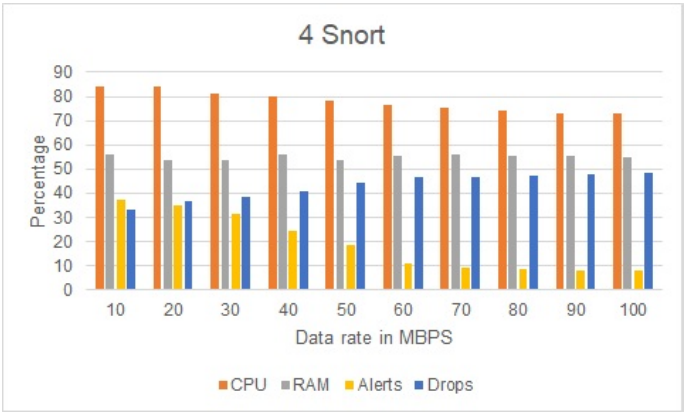

Fig. 8. Statistics with 4 snort instances with varied data rates

VI. Conclusion

In this paper, we proposed a novel architecture adaptable for launching dynamic security services at the edge. The edge infrastructure was launched as a fog node where multiple security services were able to launch simultaneously. The orchestration strategies stated in the paper holds the most significant contribution to the research community, as such a scheme was not yet published for resource-constrained edge devices. We propose the orchestration strategies of multiple device support, performance optimization, classification of ingress traffic, and dynamic allocation of resources for service instances. We have managed to validate the first strategy by running multiple security services in a light-weight virtualized environment. Our strategies are ideal for optimizing any edge infrastructure that provisions not only security services, but also the other types of service that can be launched for a remote work-site. Use cases such as smart agriculture, IIoT, and intelligent transportation systems require edge nodes deployed at distributed and remote locations. In such circumstances, our approach of launching multiple services dynamically via a light-weight virtualized environment that attribute various orchestration strategies for enhancing the service execution would be vital for realizing them pragmatically. However, the functions and capabilities of the orchestrator (i.e. SO) are dependent on the nature of the service type. Though, the four governing strategies proposed in this paper are applicable for all the use cases. We are intending to develop and validate the other orchestrating strategies in our future work.

Acknowledgement

This work is party supported by European Union in RESPONSE 5G (Grant No: 789658) and Academy of Finland in 6Genesis Flagship (grant no. 318927) projects.

References

[1] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on Multi-Access Edge Computing for Internet of Things Realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[2] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive iot data and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017.

[3] M. Liyanage, I. Ahmad, A. B. Abro, A. Gurtov, and M. Ylianttila, *A Comprehensive Guide to 5G Security*. John Wiley & Sons, 2018.

[4] D. Quick and K.-K. R. Choo, "Iot device forensics and data reduction," *IEEE Access*, vol. 6, pp. 47 566–47 574, 2018.

[5] M. Liyanage, A. Braeken, P. Kumar, and M. Ylianttila, *IoT Security: Advances in Authentication*. John Wiley & Sons, 2020.

[6] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[7] P. Ranaweera, A. D. Jurcut, and M. Liyanage, "Realizing multi-access edge computing feasibility: Security perspective," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2019, pp. 1–7.

[8] P. Zhang, J. K. Liu, F. R. Yu, M. Sookhak, M. H. Au, and X. Luo, "A survey on access control in fog computing," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 144–149, 2018.

[9] M. Aazam, S. Zeadally, and K. A. Harras, "Fog computing architecture, evaluation, and future research directions," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 46–52, 2018.

[10] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*. IEEE, 2015, pp. 130–135.

[11] A. Boudi, I. Farris, M. Bagaa, and T. Taleb, "Assessing lightweight virtualization for security-as-a-service at the network edge," *IEICE Transactions on Communications*, vol. 102, no. 5, pp. 970–977, 2019.

[12] K. Wong, C. Dillabaugh, N. Seddigh, and B. Nandy, "Enhancing suricata intrusion detection system for cyber security in scada networks," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2017, pp. 1–5.

[13] W. Park and S. Ahn, "Performance comparison and detection analysis in snort and suricata environment," *Wireless Personal Communications*, vol. 94, no. 2, pp. 241–252, 2017.

[14] J. Islam, E. Harjula, T. Kumar, P. Karhula, and M. Ylianttila, "Docker enabled virtualized nanoservices for local iot edge networks."

[15] A. Sforzin, F. G. Mármol, M. Conti, and J.-M. Bohli, "Rpids: Raspberry pi ids—a fruitful intrusion detection system for iot," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016, pp. 440–448.

[16] S. Tripathi and R. Kumar, "Raspberry pi as an intrusion detection system, a honeypot and a packet analyzer," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. IEEE, 2018, pp. 80–85.

[17] B. Caswell and J. Beale, *Snort 2.1 intrusion detection*. Elsevier, 2004.

[18] O. C. A. W. Group *et al.*, "Openfog reference architecture for fog computing," *OPFRA001*, vol. 20817, p. 162, 2017.

[19] J. Stringer, "Openvswitch without open vswitch: The api and its users," 2017.

[20] M. Horton, L. Chen, and B. Samanta, "Enhancing the security of iot enabled robotics: Protecting turtlebot file system and communication," in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 662–666.

[21] P. A. Carter, "Installation on heterogeneous operating systems," in *Pro SQL Server 2019 Administration*. Springer, 2019, pp. 101–125.

[22] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479–2489, 2017.

[23] A. D. Pinto. (2019, Jan) Tricotools. [Online]. Available: https://github.com/NozomiNetworks/tricotools

[24] R. McRee, "W32/sdbot infected machine." [Online]. Available: http://holisticinfosec.org/toolsmith/files/nov2k6/toolsmith.pcap