An SDN/NFV-Enabled Architecture for Detecting Personally Identifiable Information Leaks on Network Traffic

Sharleen Joy Y. Go*, Richard Guinto†, Cedric Angelo M. Festin‡,
Isabel Austria§, Roel Ocampo¶, and Wilson M. Tan‖
†Samsung Research Philippines
*‡‖Department of Computer Science, §¶Electrical and Electronics Engineering Institute
University of the Philippines - Diliman
Email: *sygo@up.edu.ph, †rf.guinto@samsung.com, ‡cmfestin@up.edu.ph,
§isabel.austria@eee.upd.edu.ph, ¶roel.ocampo@eee.upd.edu.ph, ‖wmtan@dcs.upd.edu.ph

***Abstract*—The widespread adoption of social networking and cloud computing has transformed today's Internet to a trove of personal information. As a consequence, data breaches are expected to increase in gravity and occurrence. To counteract unintended data disclosure, a great deal of effort has been dedicated in devising methods for uncovering privacy leaks. Existing solutions, however, have not addressed the time- and data-intensive nature of leak detection. The shift from hardware-specific implementation to software-based solutions is the core idea behind the concept of Network Function Virtualization (NFV). On the other hand, the Software Defined Networking (SDN) paradigm is characterized by the decoupling of the forwarding and control planes. In this paper, an SDN/NFV-enabled architecture is proposed for improving the efficiency of leak detection systems. Employing a previously developed identification strategy, Personally Identifiable Information detector (PIID) and load balancer VNFs are packaged and deployed in OpenStack through an NFV MANO. Meanwhile, SDN controllers permit the load balancer to dynamically redistribute traffic among the PIID instances. In a physical testbed, tests are conducted to evaluate the proposed architecture. Experimental results indicate that the proportions of forwarding and parsing on total overhead is influenced by the traffic intensity. Furthermore, an NFV-enabled system with scalability features was found to outperform a non-virtualized implementation in terms of latency (85.1%), packet loss (98.3%) and throughput (8.41%).**

I. Introduction

With the growing reliance on technology, the frequency and impacts of data breaches are expected to rise in the coming years. The Breach Level Index reports that the total number of records compromised within the first half of 2018 is 72% more than the figure for 2017 [1]. Social media industry accounts for 76.20% of the data breaches in 2018, while identity theft is the most prevalent data breach type. Additionally, according to a study conducted by IBM, the average cost of a data breach in 2018 is 6.4% larger than the cost for the previous year [2]. The aftermath of a mega breach involving 1 to 50 million records can yield an average cost of $40 to $350 million.

In an attempt to ward off such incidents, authorities are imposing stronger obligations on organizations to secure personal data. Implemented on May 2018, the EU General Data Protection Regulation (GDPR) [3] is designed to give users more control over their personal information. The GDPR grants a user with the rights to (1) access his personal data without any charge, (2) learn about the operations performed on his data, (3) restrict or halt the processing of personal data, and (4) in the event of a data breach, be informed of the incident within 72 hours. To assure data privacy for individuals, businesses have to amend their customary data handling practices, which can be very costly. Moreover, such policies protecting users by increasing the liabilities of a breached organization do not provide actual means for organizations to withstand data theft attacks.

In response, considerable efforts have been devoted to developing solutions for preventing unintended disclosure of personal data. The study presented in [4] focused on detecting and categorizing private information leakage in Twitter. The approach in [5] tracks information flows in mobile applications to uncover data leaking paths. In [6], a Privacy Capsule model is introduced to restrict the access of mobile applications to private information and prevent data spills to third parties. Built on Android's Virtual Private Network (VPN) APIs, the mobile traffic inspection tool in [7] intercepts outgoing network packets to locate personal data. Extending the security features of Android, BlackDroid [8], performs plaintext and ciphertext leak detection by setting data feature labels and guarding the Internet outlet point of mobile devices.

Operating on network traffic alone, the solutions presented in [9] and [10] are platform independent and easily deployable. ReCon [9] is a machine learning-based tool which provides an interface to block or overwrite Personally Identifiable Information (PII) leaks. In contrast, the approach of Liu et al. [10] applies a set of regular expressions and dictionary-based rules to discover different types of PII strings contained in protocol fields.

Despite the variety of solutions proposed to address the problem of privacy leaks, past endeavors have primarily concentrated on the accurate identification of personal data. Given that leak detection involves filtering large and highly variable quantities of data, the feasibility of an approach depends not only on accuracy but also on efficiency. Therefore, existing techniques must be efficiently implemented to ensure applicability in real-world setups.

The core idea behind the concept of Network Functions Virtualization (NFV) is to separate network functions (NFs) from dedicated hardware. By having virtual network functions (VNFs) run on portable virtual machines (VMs), NFV offers increased flexibility and scalability for lower installation and maintenance costs. As a consequence, a number of studies concerned with enhancing network security adopt the NFV architecture for seamless integration of security services [11]. Another emerging paradigm, Software Defined Networking (SDN) is characterized by the decoupling of the forwarding and control planes. Unlike in traditional networks, the switches directly receive forwarding instructions from an external SDN controller having complete awareness of the network topology and available resources. By improving the efficiency of the routing process, SDN shares a common goal with NFV: to optimize the performance of virtual networks. Hence, the two complementary technologies are typically deployed together.

In this paper, we propose a leak detection/prevention system by employing the PII recognition strategy of [10] and leveraging the capabilities of SDN and NFV. Contrary to previous studies which formulated methods for detecting privacy leaks, the focal point of our work is shifted towards designing a practical architecture for the purpose of leak detection. With the incorporation of NFV, the number of PII detector (PIID) instances can be scaled on demand. We acknowledge the simplicity and relevance of the network traffic-based PII identification strategy presented in [10] and thus, utilize the same ideas in the implementation of our PIID VNF. In addition, dynamic routing in SDN permits the forwarding of incoming traffic to a selected PIID instance and enables a load balancer to distribute the workload among the PIIDs present.

The main contributions of our work are listed as follows:

- An SDN/NFV leak detection system architecture based on the PII identification scheme in [10]
- In OpenStack, the architecture is implemented and experiments were ran to analyze the overhead introduced by a PIID middlebox and to demonstrate the benefits of scaling

The remainder of this article is structured as follows: Section II provides a general description of the proposed PII leak detection architecture and an overview of the architecture in OpenStack environment. In Section III, experiments are conducted to assess the performance of the proposed SDN/NFV-based architecture. Finally, our work is concluded in Section IV.

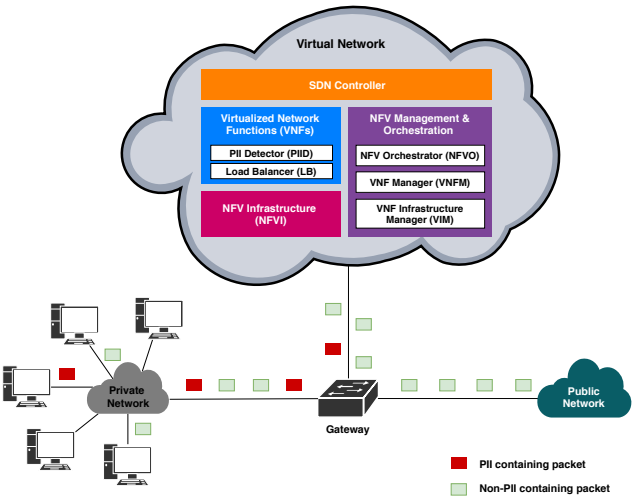

Fig. 1: SDN/NFV PII leak detection system architecture

II. System Description

A. General

The general architecture of our PII leak detection system is illustrated in Figure 1. To protect a private network from data leaks, packets from the private network, upon reaching the gateway device, are forwarded to a virtual network for PII filtering. Within the virtual network, PII detector (PIID) and load balancer (LB) VNFs, deployed on the NFV Infrastructure (NFVI), are maintained by an NFV Management and Orchestration (MANO) component. The NFV MANO is composed of three functional blocks: (1) NFV Orchestrator (NFVO) for on-boarding NS and VNF packages, NS lifecycle and global resources management, and authorization of NFVI resource requests; (2) VNF Manager (VNFM) for VNF lifecycle management; and (3) Virtualized Infrastructure Manager (VIM) for NFVI compute, storage, and network resources management.

When a new HTTP flow enters the virtual network, the SDN controller coordinates with a load balancer to select and assign a PIID instance to handle the packets from the particular flow. Since our PIID VNF is built on the techniques presented in [10], it can only detect plaintext PII strings embedded in HTTP fields. After inspection, packets found to contain PII are marked by the PIID; and depending on the desired system behaviour, a PII-containing packet may be instantly discarded by the PIID instance. Working under the assumption that all HTTP connections are persistent, data flows are identified by a 4-tuple of source and destination IP addresses and TCP port numbers. For this reason, an alternative reaction of the PIID instance is to forward the PII-containing packet to the SDN controller in order to trigger installation of a flow to drop succeeding packets from the data leaking flow. It is worth mentioning that for non-persistent HTTP connections, installing a single flow with the 4-tuple as

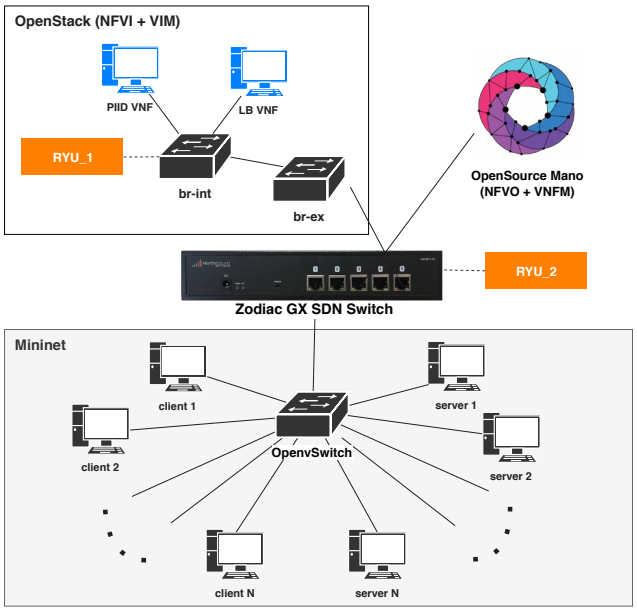

Fig. 2: Implementation of the proposed PII leak detection architecture

match fields would fail to terminate the connection as multiple port numbers are utilized to request for multiple resources from a single website. On the other hand, non-PII containing packets are handed back to the gateway device and transmitted to the public network as intended.

B. Implementation Specific

Figure 2 depicts the realization of our proposed PII leak detection architecture in OpenStack environment. Our testbed is composed of four primary components: (1) a DevStack machine equipped with an Intel Core i5-3570K processor with 8 gigabytes of RAM; (2) an orchestrator running the lightweight OpenSource Mano (OSM) [12] software stack; (3) a SuperServer 5019GP-TT on which Mininet hosts generating HTTP traffic reside and (4) a Zodiac GX SDN switch. With reference to the ETSI NFV architecture, the role of OpenStack corresponds to the NFVI and VIM; while OSM functions as both the NFVO and VNFM.

The PIID and LB VNFs are packaged as QCow2 images and uploaded via Glance API. In OSM, NS and VNF descriptor files are onboarded and a VIM referencing the DevStack deployment is created. The performance management module of OSM can facilitate periodic extraction of VM metrics given that the desired metrics are specified on the VNF descriptor files.

For the SDN element, we opted to employ dual controllers: RYU_1, inside the OpenStack cluster, is responsible for service function chaining; while RYU_2 external controller is tasked to block the HTTP flows found to leak PII. Thus, by having PII containing packets dropped early on the hardware switch instead of the software switch inside the OpenStack cluster, forwarding overhead is reduced.

III. Performance Evaluation

A. Experimental Setup

The goal of this section is twofold: (1) to analyze the forwarding and parsing overhead introduced by a PIID middlebox and (2) to demonstrate the scaling feature of an SDN/NFV architecture. For these purposes, the experimental scenario described in Table I is constructed. With reference to Figure 2, the topology is comprised of N=6 pairs of clients and servers. The experimental scenario is designed to be divided into 7 time intervals wherein at the beginning of each interval, the network load is varied by starting or terminating HTTP connections between pairs of clients and servers. Under the same experimental scenario, the following four sets of experiments are conducted: For case 1, no middlebox is present and HTTP packets are directly sent from clients to servers. In case 2, a middlebox acting as a passthrough sits between the pairs of clients and servers. All packets sent by the clients are directed to this middlebox which forwards the sniffed packets to the destination server. Compared to case 2, the middlebox in case 3 also performs parsing to discover PII leaks embedded in HTTP packets. Employing the same leak detection logic in case 3, case 4 also involves running a background script to monitor the CPU utilization of the PIID instances and initiate scaling mechanisms as necessary.

B. Results

This subsection is divided into two parts. In the first part, cases 1-3 are compared to quantify the overhead caused by adding a middlebox in the system. On the other hand, the second part solely looks into cases 3 and 4 to realize the advantage of scaling the number of PIID instances according to demand. To assess the performance of the HTTP flows in each experiment case,latency, packet loss and throughput are considered.

Furthermore, data collected from 30 runs of each case are presented in two ways. The interval-based analysis is provided to reveal the effects of varying the network load on the three metrics. In contrast, the second approach proceeds with a more general approach and examines the same metrics but taken over an entire experiment run.

Interval#	Event	PIID Load (kBps)
1	Flow# 1 is started at 18 kBps.	18
2	Flow# 2 is started at 40 kBps.	58
3	Flow# 3 is started at 32 kBps.	90
4	Flow# 4 is started at 25 kBps.	115
5	Flow# 5 is started at 40 kBps; and soon after, Flow# 6 is also started at 18 kBps.	173
6	Flows# 3 & 6 are terminated.	123
7	Flows# 2 & 4 are terminated.	58

TABLE I: Experimental scenario

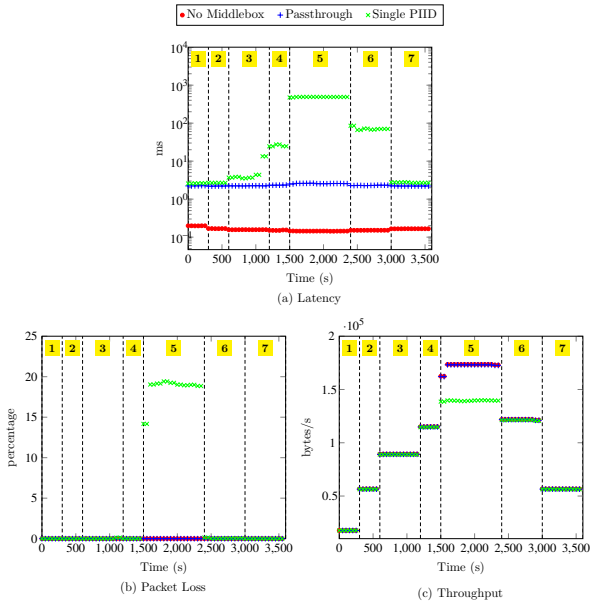

Fig. 3: Interval-based analysis of the overhead caused by a PIID middlebox

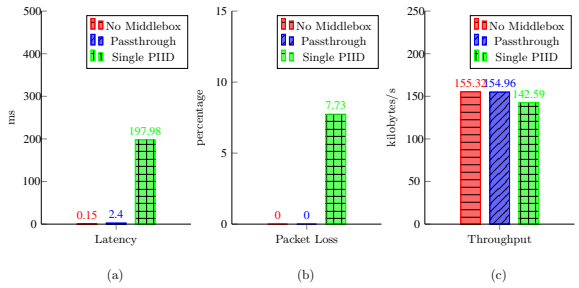

Fig. 4: General analysis of the overhead caused by a PIID middlebox

a) Quantifying Overhead:

- Interval-based Analysis

The data gathered from 30 runs of each experiment case were processed according to the following steps. For each run, the average of metrics associated to all active flows are taken within 100-second time windows. Afterwards, the metrics associated to the same interval across 30 runs of the same experiment case are aggregated eventually yielding the data points shown in Figure 3.

During the first interval, a single flow is present; and the network load is 18 kBps. Latency in case 2 is 11.3 times larger than case 1 while that of case 3 is only 1.17 times larger than case 2. Currently, the overhead induced by a PIID middlebox is dominated by the cost of forwarding. The increase in network load during the second time interval resulted to no significant change in the 3 metrics. At the beginning of the third time interval, the additional 32 kBps resulted to a slight increase in the latency of case 3. As the network load increases, the difference between the latencies of cases 2 and 3 continues to grow while transitioning to the fourth interval.

At the fifth interval, two flows are introduced yielding a total network load of 173 kBps. While increasing the network load produced a slight increase in case 2's latency (Figure 3a), packet loss (Figure 3b) and throughput (Figure 3c) during the fifth interval remained the same as the fourth interval. On the contrary, a drastic change is evident in all three metrics of case 3. Latency increased by a factor of 17.8; while throughput decreased by 19.3% due to packet loss of nearly 20%. Moreover, latency in case 2 is only 17.6 times larger than case 1 whereas that of case 3 is 192 times larger than case 2. Unlike the observation during the first interval, parsing overhead is now more significant than the cost of forwarding. As flows are terminated at the next intervals, the three metrics in both setups improve.

- General Analysis

The graphs in Figure 4 present a comparison among the three cases on the basis of the data flows' performance metrics averaged over the entire experiment duration. The forwarding overhead in case 2 only caused a 2.4 ms increase in latency and no significant changes in packet loss and throughput. On the other hand, the parsing overhead in case 3 caused a 7.73% increase in latency, packet loss, as well as 12.4 kBps decrease in throughput.

The magnitude of overhead induced by installing a PIID middlebox is affected by both the incoming traffic rate and the leak detection operations executed. Based on the results, forwarding, as opposed to parsing, has a greater influence over total overhead under low traffic. Conversely, the reverse is true when network load is high.

b) Demonstrating Scaling Mechanisms:

- Interval-based Analysis

The per-interval average latency, packet loss and throughput in cases 3 and 4 are depicted in Figure 5. The graphs were obtained by the same operations as done for Figure 3 except that the association of metrics to flow# is made apparent. Details regarding the scaling actions performed during the experiment are provided in Table II.

During the first two intervals, the network load did not cause the CPU utilization of the PIID VM to exceed the maximum threshold of 60%. Therefore, no additional instance is created and all the resulting metrics in cases 3 and 4 are equal during these intervals.

At the third time interval, introducing a 32 kBps flow caused the CPU utilization of PIID# 1 to exceed 60%. Being the largest flow assigned to PIID# 1, flow# 2 is reassigned to the new PIID instance. With the utilization of PIID# 2 lower than PIID# 1, the latency of flow# 2 becomes lower than the latencies of flows# 1 and 3. Due to delays in metric collection, this is only visible at the latter half of the third interval in Figure 5d.The other metrics,

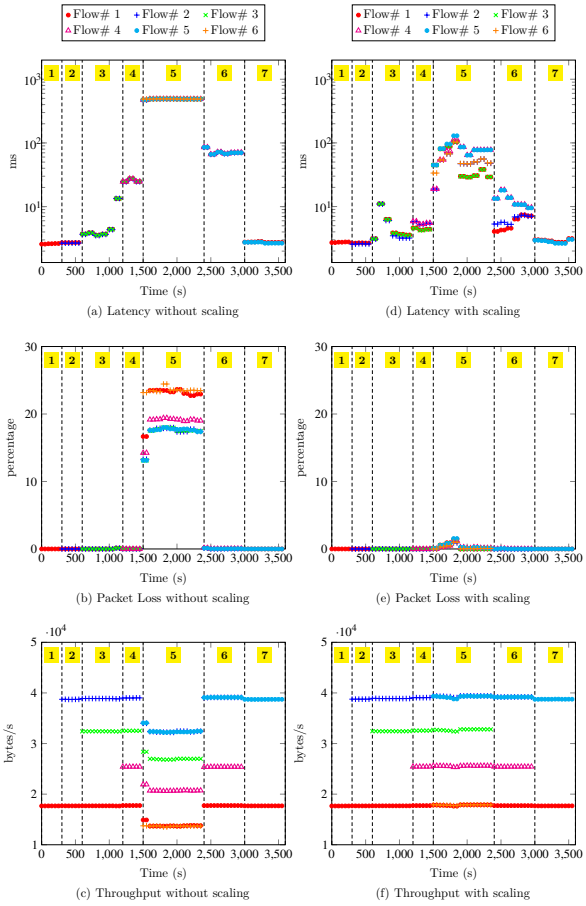

Fig. 5: Interval-based comparison of PIID systems with and without scaling capabilities

however, display no evident change between cases 3 and 4 as the network load is still considerably low.

During the fourth interval, flow# 4 is started and in case 4, is assigned to PIID# 2. As the load in PIID# 2 becomes greater than that of PIID# 1, flows# 2 and 4 which are assigned to PIID# 2 experience higher latencies in comparison to flows# 1 and 3 as shown in Figure 5d. Nevertheless, the latencies of all flows during the fourth interval of case 4 (Figures 5d) are lower than their corresponding values in case 3 (Figure 5a).

With the introduction flows# 5 and 6 at the fifth interval, the network load reaches 173 kBps. In case 3, the single PIID instance becomes overwhelmed resulting to a significant rise in latency (Figure 5a) and packets loss (Figure 5b) and drop in throughput (Figure 5c). When scaling is enabled, flows# 5 and 6 are assigned to PIIDs# 1 and 2, eventually causing the CPU utilization of both instances to surpass the maximum threshold. Since the load balancer prioritizes reducing the load of the instance with a higher utilization, flow# 5 is first selected for redirection to PIID# 3. Furthermore, flow# 4, the second largest flow assigned to PIID# 2, is also redirected to PIID# 3. Flow# 2, the largest flow assigned to PIID# 2, was not selected because doing so would result to PIID# 3 having an 80 kBps load which is close to 90 kBps, the load when a PIID# 1 was found to exceed the maximum threshold. By saving the aggregated rate of flows assigned to a PIID instance found to be overloaded, the load balancer can estimate the capacity of a PIID instance and avoid flow reassignments which would cause overloading. Similar to the observation in the third time interval, the scale up mechanism was not triggered instantly. Notice that in the fifth interval of case 4 improvement in latency (Figure 5d), packet loss (Figure 5e) and throughput (Figure 5f) only occurs after some time. Within this interval, employing 3 PIID instances resulted to significant improvements in all the flows' metrics.

During the sixth interval, the termination of flows# 3 and 6 reduced the load on PIIDs# 1 and 2 in case 4. Although the load on PIID# 3 remained the same, the latencies (Figure 5d) associated to flows# 4 and 5, assigned to PIID# 3, also improved. This behaviour is caused by having all flows traverse the same physical link connected to the OpenStack machine housing multiple PIID instances. Additionally, the CPU utilization of PIID# 1 drops below the 20% minimum threshold. Since the utilization of PIID# 2 is lower than PIID# 3, the flows assigned to PIID# 1 are reassigned to PIID# 2. Accordingly, the latency of flows# 1 and 2 increases after the redirection of flow# 1 from PIID# 1 to 2 as evident from Figure 5d. At the final interval, the network load drops to 58 kBps. With scaling enabled, PIID# 2 becomes under utilized and its flows are redirected to PIID# 3. In both cases 3 and 4, a single PIID instance is used to process flows# 5 and 6. Likewise, the metrics in both cases becomes equal.

- General Analysis

Interval#	Scaling Event	PIID#: Load before scaling	PIID#: Load after scaling
1	-	1: 18	-
2	-	1: 58	-
3	Create PIID #2. Redirect flow# 2 from PIID# 1 to 2.	1: 90	1: 50, 2: 40
4	-	1: 50, 2: 65	-
5	Create PIID #3. Redirect flow# 5 from PIID# 1 to 3 and flow# 4 from PIID# 2 to 3.	1: 90, 2: 83	1: 50, 2: 58, 3: 65
6	Flows assigned to PIID #1 are reassigned to PIID #2. PIID #2 is deleted.	1: 18, 2: 40, 3: 65	2: 58, 3: 63
7	Flows assigned to PIID #2 are reassigned to PIID #3. PIID #2 is deleted.	2: 18, 3: 40	3: 58

TABLE II: Summary of scaling events

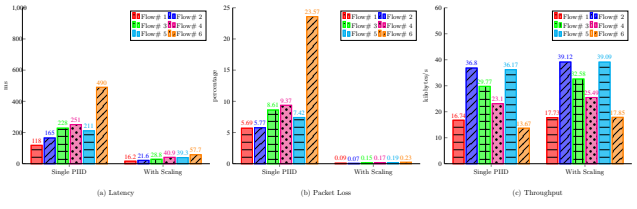

Fig. 6: General comparison of PIID systems with and without scaling capabilities

The graphs in Figure 6 compares cases 3 and 4 in terms of the 6 flows' performance metrics averaged over the entire experiment duration. All of the metrics resulting from case 4 is far better in comparison to case 3. Latency (Figure 5a), packet loss (Figure 5b) and throughput (Figure 5c) improved by an average of 85.1%, 98.3% 8.41% respectively.

Scaling capabilities are essential for the feasibility of middlebox-based solutions. As demonstrated, increasing the network load can quickly overwhelm a single PIID middlebox resulting to performance degradation.

IV. Conclusion

With the ever-growing collection of sensitive information stored on the Internet, the consequences of data breaches ranges from temporary financial losses to long lasting reputational damages. Realizing the risks associated with such incidents, numerous techniques to detect and prevent privacy leaks have been developed. With leak detection involving scouring masses of data, the feasibility of a solution relies not only on accurate identification of private information but also on efficient task execution. In this paper, SDN and NFV principles are leveraged to design a practical architecture for the purpose of PII leak detection. Our system consists of a MANO responsible for instantiating PII detectors and load balancers, an OpenStack cluster on which VMs are deployed and multiple Mininet hosts for generating HTTP traffic. Dual SDN controllers also promote seamless routing of traffic among VNF instances.

To quantify the induced overhead, the system's performance was analyzed under varying network loads and intensiveness of packet processing. From the results, we conclude that forwarding accounts for a larger proportion of the overall overhead when network traffic is fairly low; the opposite becomes true under sufficiently high traffic. Because there are no network appliances dedicated to PII detection, we instead emulated one by turning off the scaling capability of our implementation and compared its performance with our scaling-capable system. Results suggest that employing the SDN/NFV-based architecture effectively decreased latency and packet loss by 85.1% and 98.3% respectively, while increasing throughput by 8.41%.

References

[1] "Data breach statistics." https://breachlevelindex.com/.

[2] "Ibm study: Hidden costs of data breaches increase expenses for businesses." https://newsroom.ibm.com/2018-07-11-IBM-Study-Hidden-Costs-of-Data-Breaches-Increase-Expenses-for-Businesses.

[3] "2018 reform of eu data protection rules." https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[4] H. Mao, X. Shuai, and A. Kapadia, "Loose tweets: An analysis of privacy leaks on twitter," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, (New York, NY, USA), pp. 1–12, ACM, 2011.

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *SIGPLAN Not.*, vol. 49, pp. 259–269, June 2014.

[6] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: Preventing information leaks by mobile apps," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, (New York, NY, USA), pp. 399–411, ACM, 2016.

[7] Y. Song and U. Hengartner, "Privacyguard: A vpn-based platform to detect information leakage on android devices," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, (New York, NY, USA), pp. 15–26, ACM, 2015.

[8] Y. Zhang, Y. Wang, D. Wang, R. Zhang, and Q. Zhou, "Blackdroid: A black-box way for android plaintext and ciphertext privacy leaks detecting and guarding," in *2013 3rd International Conference on Consumer Electronics, Communications and Networks*, pp. 178–181, Nov 2013.

[9] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "Recon: Revealing and controlling pii leaks in mobile network traffic," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, (New York, NY, USA), pp. 361–374, ACM, 2016.

[10] Y. Liu, H. H. Song, I. Bermudez, A. Mislove, M. Baldi, and A. Tongaonkar, "Identifying personal information in internet traffic," in *Proceedings of the 2015 ACM on Conference on Online Social Networks*, COSN '15, (New York, NY, USA), pp. 59–70, ACM, 2015.

[11] T. Alharbi, A. Aljuhani, H. Liu, and C. Hu, "Smart and lightweight ddos detection using nfv," in *Proceedings of the International Conference on Compute and Data Analysis*, ICCDA '17, (New York, NY, USA), pp. 220–227, ACM, 2017.

[12] "Open source mano (osm)." https://osm.etsi.org/.