Enhancing security of SDN focusing on control plane and data plane

Barbora Čelesová, Jozef Vaľko, Rudolf Grežo, Pavol Helebrandt
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Bratislava, Slovakia
{xcelesova, xvalkoj1, rudolf.grezo, pavol.helebrandt}@stuba.sk

***Abstract*— Software-defined networks (SDN) have appeared as effective network technology, which is able to support the dynamic nature of future network functions and intelligent applications. On the other hand, the progress of the SDN is limited by various security threats. Analyzing the centralized nature of SDN, we found multiple potential vulnerabilities, which the attacker may use. Our solution covers a wider area, not just data plane, but also control plane security. The traffic, which is flowing through a data plane, could include various security threats. To detect them, we utilize OpenFlow possibilities and Machine Learning (ML) concept for the proposed Network Intrusion Detection System based on Deep Neural Network (NIDS-DNN). The solution can extract network statistics from OpenFlow switches (OF switches) and process them with DNN. The result is to warn about an attack on the data plane and to prevent malicious users from harming the network. For early detection of DoS/DDoS attacks aimed at controller, we present our solution – Specter, which changes the approach to the flow processing prioritization. Using priority queues ensures a better quality of service for legitimate users. To the best of our knowledge, our work is the first solution, which couple intrusion detection in the data plane with protection against DoS attacks in control plane.**

Keywords—Software defined networks, controller security, intrusion detection system, deep neural network, priority queues

I. INTRODUCTION

SDN has achieved a lot of attention in both industrial and academic sphere. It is a new network architecture framework for networking, which decouples the network control plane from the data plane at physical topology. SDN introduces centralization of network control and establish the ability to program the network. On the other hand, the progress of the SDN is limited by various security threats [1]. The aim of this work is to analyze the security in software defined networks which is important research area. We focus on security analysis of the SDN architecture and corresponding layer threats and countermeasures.

The architecture of SDN consists of three planes – application, control and data plane. The separation of the control and data plane means that all control logics are now defined in the control plane, and the data plane is responsible only for data forwarding. That makes network management much easier and further allows the network to scale and adapt to increasing demands. Despite the advantages of SDN, it is very important to understand security issues arising from its use [2].

The controller uses a suitable protocol to manage OF switches. Nowadays, it is the most widely used OpenFlow protocol, which can be considered as a standard. OpenFlow was firstly developed at Stanford University and now its specification versions are proposed by Open Networking Foundation (ONF) [3]. This protocol has many techniques how to process the management of the OF switches by the controllers.

ML is emerging today and it is implemented into many aspects of our lives. Pictures processing, face recognition or even network intrusion detection and prevention systems based on ML are the topics of many pieces of research [4, 5, 11]. By using ML, we can develop systems which can be more flexible and cost-effective than traditional approaches. These systems can be programmed to detect known attacks, but more interestingly, it could detect the unknown so-called zero-day attacks which have not occurred before. Furthermore, in SDN, there is also a room for detection, but also prevention against such attacks. This protects the network and its users from malicious activity.

In this paper, we propose a solution to address security threats to both data and control planes of SDN. We aim to deliver a complex security system that will be able to detect malicious traffic on the data plane and ensure correct controller behavior and improve resistance against flood attacks. If a controller is under attack, the whole network may be affected. OF switch cannot forward packets without rules that are provided by controllers. If there are no rules to forward packets in the OF switch, it will not do it at all.

II. STATE OF THE ART

Security has recently become an eminent open research problem in SDN. Thus, there are many interesting publications. We focused our analysis on the field of securing the data and the control plane. We found the most relevant of these solutions. Authors of [5] use DNN to detect anomalies in the network. When detecting attacks, they have chosen six basic features such as duration, protocol type, source bytes, destination bytes, number of connections to the same host, and to the same service. The NSL-KDD dataset [6] was chosen to train their neural network. They achieved accuracy at value 75.75%, which is not the best result, but it is caused by the choice of features. On the other hand, Niyaz et.al. [7] aimed their work at DDoS attack-detection. They based their solution on multi-vector attack detection in SDN environment. Their proposed system is able to detect individual DDoS attack class with an accuracy of 95.65%.

Most of the existing solutions, which are focused on DoS/DDoS against the controller, limit request rate to the controller by dropping overflowed request resulting in dropping legitimate requests. The latest OpenFlow mechanism applies short buffer size of OF switches and dropping redundant requests before sending them to the controller. Using first-come-first-serve mechanism results in a high drop rate of legitimate flow requests when the controller is under attack. To protect the controller from DoS and DDoS attacks better than standard first-come-first-serve (FCFS) methods, Wei et. al. proposed a solution named

978-1-7281-2827-6/19/$31.00 ©2019 IEEE

FlowRanger [8] that protects SDN from data-to-control plane flooding attacks. FlowRanger uses priority-based buffer to handle flow requests based on their likelihood to be malicious. The Packet_In requests to the controller are served through a job prioritization process. Prioritization uses a ranking algorithm to identify malicious users according to past requests to the controller. If the request came from a trusted source, it is served with higher priority. On the other hand, a malicious request is served with lower priority. We found this solution as an interesting theoretical proposal in which by adapting multiple approaches, we can reach improvement of security in SDN.

III. Emerging Challenges

The key to the intelligence of the artificial neural networks is in the data on which these networks are trained. The better data, the better performance we can get from the neural network. Surely, we could say the same to the characteristics of DoS/DDoS attacks on the controller. Both of them are described below.

A. Datasets

The datasets are used to store this data. We have analyzed various historical datasets and also modern ones. The most widely used dataset for the anomaly detection systems has been KDD CUP 99 and its modification NSL-KDD dataset [6] since its creation in 1999.

Both above-mentioned datasets are considerably old. On the other hand, UNSW-NB15 dataset [9] was created to address modern low footprint attacks. The main motivation of the authors was the fact that old datasets cannot effectively simulate the behavior of contemporary networks. The authors used IXIA PerfectStorm tool to create a hybrid of normal and malicious network traffic. Anomalous traffic is divided into 9 classes of attacks - Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. This dataset was used in several studies, whether in [4] for solution evaluation or in [10] for the purposes of the comprehensive survey of using ML techniques for intrusion detection. The authors made their captured packets publicly available as long as processed connections in the form of CSV files.

B. Demonstration of the DoS/DDoS attacks on the controller

In the DoS/DDoS attack, the main goal is to make controller functionality unavailable to legitimate users. An attacker tries to do this by exhaustion of memory or other computing resources. When we mix normal and attacking traffic together, it is quite difficult to differentiate between them. According to the OpenFlow specification, if an OF switch does not know how to handle a new packet, it will first store this packet in its Flow Buffer and then send a Packet_In message to the controller to request for instructions. Due to this fact, the controller has to deal with an enormous amount of Packet_In messages during a DoS attack. This may cause exhaustion of resources and processing of normal traffic might become difficult [12].

IV. Architecture

The main idea of this work is to create a complex security solution for modern SDN environments. The controller is the single-point-of-failure. Thus, we propose the solution to protect it against DoS attacks. Furthermore, we utilize its overview of the underlying infrastructure to detect and prevent various attacks in the data plane. Therefore, our solution is divided into 2 main modules:

- Control plane security
- Data plane security

Control plane security deals with the logic of the controller. It is aimed at processing *Packet_In* requests and prevents malicious users to make effective DoS attack on the controller by generating a large number of unique flow requests. The controller also collects flow statistics from the OF switches and evaluates them using ML in order to deliver data plane security. The proposed architecture is in Fig. 1.

A. Control plane security – Specter

By increasing control plane security, we ensure a better quality of service for legitimate users by changing the approach to the flow processing from FCFS to prioritized. We will distinguish users by IP address and everyone will be given a percentage of its legitimacy. This will be dynamically recalculated according to its behavior during the operation of the network. Then, user requests will be stored to priority buffers and then by using the scheduling algorithm, they will be served. Based on previous statements, we proposed the following three stages the packets have to pass through.

1) Stage 1 – Confidence award

The most important part of the whole Specter solution is the rating algorithm. It computes the confidence value of a requester. If a request comes from a new user, the initial value will be assigned to him. Based on the controller resources, the list of the requester has to be periodically modified and kept up-to-date. If the number of total requests in a time slot exceeds the threshold for that user, their requests will be dropped immediately, and the list of rejected requests will be updated. Additionally, when the confidence value is lower than a computed threshold for a malicious user, the requester is categorized as an attacker. Consequently, OF switches will be informed to drop any further packets from that user.

2) Stage 2 – Sorting to queues

Depending on the confidence value which was assigned to request, it may be now sorted to one of the queues. Our solution will contain multiple queues and depending on their priority, the request will be served. In order to avoid overflowing, if the sum of all queue's lengths exceed the maximum allowed length; controller drops the request from the tail of the lowest priority non-empty queue.

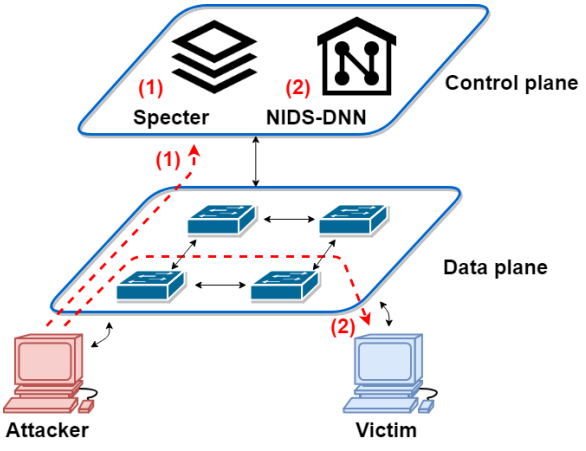


Fig. 1. Complex architecture

3) Stage 3 – Serving requests

After the request is appended to one of the queues, it is waiting to be served. FlowRanger uses a weighted round-robin scheduling algorithm to process the requests. We have improved this algorithm by adding absolute highest priority to the requests stored in the buffer with the highest priority. It means that in the first place, all requests stored in the buffer with the highest priority are served. After that, queue weights are computed according to the length and the priority of the remaining queues.

B. Data plane security – NIDS-DNN

This part of the solution is a built-in module of the SDN controller. It is detecting the anomalies in the network traffic with DNN. The DNN uses the statistics provided by OF switches to the controller in certain periods of time to classify the particular flows as normal or anomalous traffic.

The DNN module is implemented inside of the SDN controller as we can see in Fig. 2. The main functionality includes:

1. The controller gathers the information about the traffic from the OF switches using OpenFlow protocol tools.
2. The traffic in the OF switches is classified into several matching tables.
3. DNN is the module running within the controller and detecting malicious activity in the data plane.
4. The proposed system raises an alarm in case of anomalous activity is detected and blocks the malicious user.
5. The operator is able to set threshold values to tune the sensitivity of the DNN.

1) Flow tables

The controller installs the rules into flow tables of the OF switches. Since resolving the flow is done top to bottom, usually the last entry of the flow table is a path to the local controller. First packet of every new connection go through the controller. Thus, this packet can be saved and processed by NIDS-DNN.

Our solution uses several flow tables to store the various statistics about the traffic. These tables are host count table, service count table, and switching table. These tables are hierarchical, so the packet goes through all tables. After receiving the packet, the OF switch updates the counters in the first table and sends the packet to the second table. The same procedure is executed in the second table and the OF switch continues processing the packet until it finds the egress port.

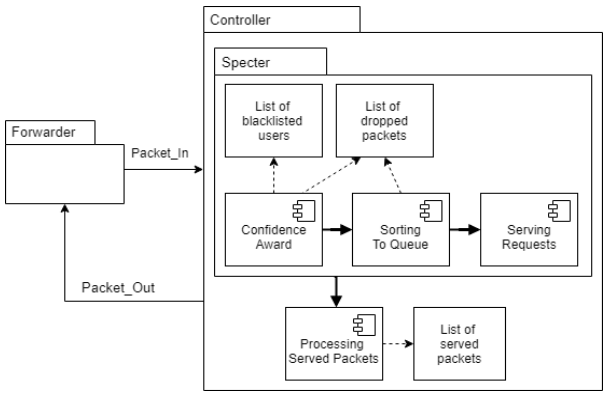


Fig. 2. Diagram of Specter components

2) DNN module

Our proposed controller keeps the identification of OF switches, called datapaths. The controller sends these datapaths to the DNN module. The DNN module triggers action processing by calling corresponding controller handling functions.

After gathering the datapaths, the DNN module is then able to get statistics from every OF switch. With the single request, we can obtain only statistics from one OF switch and from one ingress port. Consequently, the DNN module has to request the port statistics. We need these statistics to know which ports are active so we can request flow statistics from them. Received active datapaths and port numbers are then used for flow stats requesting. As a result of this, the DNN module will have an overview of the underlying network which will be refreshed in the configurable time interval.

3) Data processing

Received flows must be pre-processed before their evaluation. Data will have to be parsed from the raw form to the one suitable for DNN input. This procedure is divided into 5 steps:

1. Parse raw data to flows – extracting only information that is interesting for us.
2. Remove duplicates – removing duplicates about the same flow from different OF switches.
3. Add extended statistics – computing extended features like host and service count.
4. Merge flows into connections – source to the target and vice versa flows are paired into connections.
5. Packet_In processing – add statistics from first packets routed through controller into created connections.

The extended features in step 3 are host and service counts. Host count is the number of connections targeting the same host in the last x seconds. Service count is the number of connections targeting the same service and destination address in the last x seconds.

V. Implementation

In the current stage, we have created a proposed solution with two modules, which are implemented using Ryu controller framework. SDN data plane is emulated by Mininet.

A. Specter

Specter is called when a new Packet_In appears on the controller. Using the rating algorithm, it evaluates the sender and it sets its trusted value. According to this value, it stores Packet_In message into one of the priority queues. Meanwhile, a thread is running in the background to check if a request is in a priority queue. If so, it uses our modified weighted round-robin algorithm and processes this packet. Processing the packet means returning the packet to the default Ryu function *packet_in_handler*. Fig. 2 illustrates the components of Specter.

1) Module – Confidence Award

The function *_packet_in_handler* is called whenever the Packet_In message arrives at the controller. At this point, function *confidence_award* is also called. In the beginning, a source IP address is extracted from the packet with its time of arrival. This information is used as a key to the dictionary and then the source IP address is saved. Then, it is necessary to process the rating algorithm over this packet. We have implemented configurable default confidence value and

default threshold value in case we do not have any record about Packet_In source IP address. Otherwise, we take into consideration its previous behavior. In the meantime, we store actual Packet_In counters in the time slot and then use this value when comparing it with its threshold. During one time slot, we process only limited value of requests per user. All other requests are rejected and we decrease the confidence value of the user. In case, that users do not send a huge number of packets but there are many users sending just a few packets; single user thresholds are not exceeded. However, the controller seems to be under attack, so we have to recalculate user confidence value. On the other hand, when confidence value is less than a threshold for the malicious user, we consider this user as an attacker and this user is blacklisted. Then controller installs new flow to OF switches which denies them to process an incoming request from this user. After packet confidence value is recalculated and updated, it calls the *sorting_to_queues* function in Sorting to Queue module.

In the meantime, the standalone background process periodically clears a list of Packet_In counters and updates confidence values according to user appearance in the current timeslot. At the end of each timeslot, we also update thresholds for users based on the number of received Packet_In messages in the current timeslot.

2) Module – Sorting to Queue

The primary purpose of this module is to store the Packet_In to the corresponding priority buffer. In order to know where to save it, it is necessary to calculate the index. The formula for this calculation is as in (1).

It uses the min and max confidence values. If *actual_buffer_length* of all buffers is less than *total_buffers_length* specified by a network administrator, we can append this Packet_In at the end of the corresponding *priority_buffer*. However, if the *actual_buffer_length* of all buffers is equal or greater than *total_buffers_length*, it is necessary to reject the Packet_In from the non-empty lowest *priority_buffer*. After that, we can append Packet_In to the corresponding *priority_buffer*. At the end of this, we release *sem_priority_buffer*. This terminates the thread which started when the Packet_In arrived at the controller and called function *_packet_in_handler*.

3) Module – Serving Requests

The function serving requests is a stand-alone thread that is still running in the background and checks whether there are any requests stored in the priority buffers that might be sent for further processing. For our solution, we modified scheduling algorithm – weighted round-robin. The change over the weighted round-robin is that we process all packets from the buffer with the highest priority first.

In practice, processing begins when there is at least one packet in the *priority_buffer*. We calculate the weights for all buffers except the one with the highest priority and we find the actual length of each buffer. If there are packets in the highest priority row, all of them are processed before packets from other buffers. Next, we process packets from the buffers with the higher priority to the lowest priority. The number of processed packets from the corresponding buffer appertains to the weight we calculated at the beginning of the function. If the weight is greater than the number of packets in the buffer, only packets in this buffer are processed. We use the command pop to serve packet from the buffer. Once this process is completed, we check again if there are any stored packets in the priority buffers. If not, the thread waits for 10 μs before checking the buffers again.

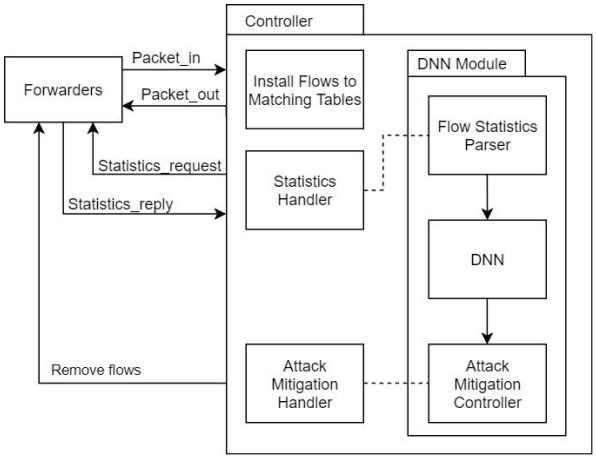


Fig. 3. Diagram of NIDS-DNN components

After serving a packet from a priority buffer, it is sent to function *create_flow*. This function is an original continuation of *_packet_in_handler*. This is the logic (that) we took from Ryu and it is just modified to process packets, whether it is flooding or sending to an output port.

B. NIDS-DNN

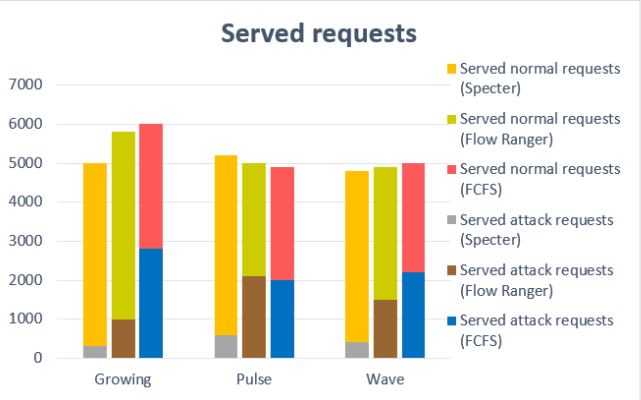
NIDS-DNN is able to mitigate the malicious traffic in the data plane. Firstly, the controller processes requests from OF switches and it installs flows into particular matching tables of OF switches. Secondly, the DNN module calls the functions of the controller and it is in charge of statistics requesting. Received statistics are sent from the controller into the DNN module. This raw data is processed into a form suitable for DNN. Our work now continues by implementing DNN and Attack mitigation components. Finally, the DNN will make a prediction whether the traffic represented by statistics is malicious or not. In case of an attack in the data plane, the DNN module mitigates the attack by calling the controller's function for flow table modification. The components of NIDS-DNN are depicted in Fig. 3.

1) Controller modification

We have used Ryu controller application, the simple switch for traffic forwarding. Furthermore, we have implemented 3 matching tables described in Section IIV. In order to be able to request port and flow statistics and later process the replies, it is necessary to implement handlers for these functions.

Port stats request function is used for gathering the active ports of the OF switch which is defined by its datapath. Port stats reply handler is there for saving active port numbers of every OF switch. The DNN module then collects this data from the controller by using the Queue Python module, which implements needed locking semantics.

$$index = round\left(\frac{confidence\_list[src\_ip] - min\_confidence\_value[0]}{max\_confidence\_value[0] - min\_confidence\_value[0]} * number\_of\_queues\right) \quad (1)$$



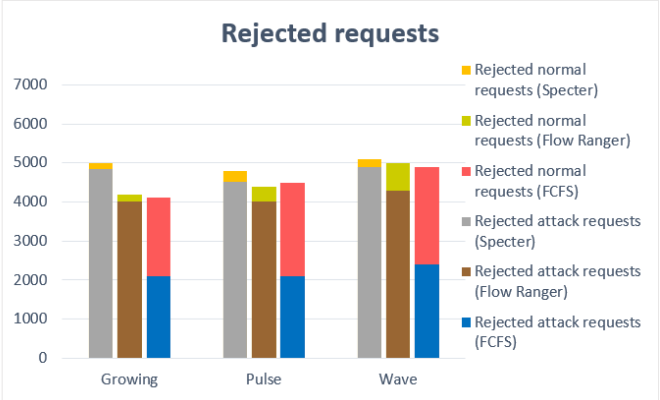


Fig. 4. Experiment comparison

Flow stats request function works similarly to the port stats request. The only difference here is that we have to specify the ingress port from which we want to obtain statistics. DNN module can trigger the flow stats request for every active port of every OF switch. The flow stats reply handler only unpacks the received raw data, puts them into a dictionary with correlation to OF switches and ports, and finally sends the data via Queue to the DNN module.

2) Traffic classifier

The DNN module is the brain of our solution. It triggers the controller's functions, process the received flows, and evaluate them using the neural network. This module is a stand-alone class in our solution and runs in the separate thread from the controller. It is started separately if needed. As input parameters, it gets the connections to the controller and also to the shared Queue. The Queue is used for passing the data between the controller and the DNN module.

The functionality of the module includes getting the connection to the OF switches in the data plane and then periodically requests the port stats from every OF switch, flow stats from every active port, parses the received data and feeds them into the neural network. After evaluating the result, possible action against malicious traffic is performed. By default, the aforementioned procedure is repeated every 10 seconds.

The classification is made using DNN trained on the UNSW-NB15 training set. The model consists of 12 input, 32 and 16 hidden respectively, and 1 output neuron. The goal of the model is to classify the traffic in the data plane as normal or malicious. This model with custom weights and used scaler is loaded and used for the evaluation of the connections in real time. The resulting prediction is then used as an input for Attack mitigation components.

VI. Evaluation

In this section, we evaluate the performance of both – Specter and NIDS-DNN module. Firstly, we describe the required setup followed by achieved results.

A. Specter Evaluation Setup

To evaluate the performance of Specter, we decided to simulate conditions similar to those used by FlowRanger authors. Thus, we implement our own traffic generator of three different DoS attacking workloads pattern – namely growing, pulse, and wave. For this purpose, we used Mausezahn, which is a fast traffic generator. Each simulation lasts for 50 seconds. It consists of normal traffic with average value of 100 requests per second, and it is distributed by the Poisson distribution. This traffic is injected with an attack request, where the peak load is set to 150 requests and the lowest to 50 requests.

We consider an SDN network with 100 users in a ratio of 80 legitimate and 20 malicious users. The maximum buffer length of the controller is set to 200. This buffer is divided into 3 priority queues. This evaluation is focused on the ratio of served and rejected normal and attack requests. Our results are compared with results of FlowRanger as well as FCFS.

B. Specter Evaluation Results

To get more accurate evaluation results we performed multiple simulations. Fig. 4. shows overall results. FCFS scheduling algorithm uses only one buffer that is noticeable at all attacking workloads patterns. It does not have any mechanism to differentiate normal and attack requests. On the other side, FlowRanger and Specter are capable of detecting and thus rejecting DoS attacks, which mitigate its impact. Specter achieved better performance, serving 15.53% more legitimate requests and rejecting 4.8% more malicious requests compared to FlowRanger. Detecting attacks, which simulate normal users, is still big challenge and provide room for improvement for both FlowRanger and Specter.

C. NIDS-DNN Evaluation Setup

We have evaluated our NIDS-DNN in terms of accuracy, sensitivity, specificity, and precision. To calculate these values, we used a confusion matrix. Here, True Positive (TP) is the number of attack records correctly classified, True Negative (TN) is the number of normal records correctly classified, False Positive (FP) is the number of normal connections classified as attacks, and False Negative (FN) is the number of attacks classified as normal traffic.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

$$Sensitivity = \frac{TP}{TP+FN} \quad (3)$$

$$Specificity = \frac{TN}{TN+FP} \quad (4)$$

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

TABLE I. NIDS-DNN EVALUATION STATISTICS

Statistics	Values
Accuracy	80.73%
Sensitivity	79.27%
Specificity	81.63%
Precision	41.11%

D. NIDS-DNN Evaluation Results

The evaluation can be divided into 2 parts: theoretical and practical. Firstly, we trained our model with the UNSW-NB15 training set. It contained 37 000 normal and 45 332 attack connections. The training was done during 100 epochs and batch size of 10 with Adam optimizer and learning rate at value 0.001. For regularization, we used a dropout rate of 0.2. The result was 93.17% accuracy and 18.56% loss. For theoretical evaluation, we used testing set provided by the authors of the dataset. The testing set contained 56 000 normal and 119 341 attack records. The resulting accuracy was 91.79% and loss 24.14%.

Secondly, the practical evaluation was done by replaying the same packets that were used to create UNSW-NB15 dataset. We used tools provided by the program *tcpreplay*. The random groups of packets were picked as samples and replayed into our SDN testbed. It contained 3 - three OF switches in a row and traffic was split into server-to-client and client-to-server parts. In order to simulate real connections, one part of the traffic was replayed from OF switch 1 towards OF switch 2 and the second part from OF switch 3 to OF switch 2. This way, we got information about all flows in OF switch 2 which could be collected and evaluated by NIDS-DNN. The resulting statistics are shown in Table I. The accuracy of the proposed model on simulated packets is quite high but the low precision rate means that used model is likely to create false alarms.

VII. CONCLUSION

In this paper, we set out to enhance the security of both control and data planes in SDN. This was achieved by design and implementation of two modules for mitigation of DoS/DDoS attacks - Specter for control plane, and NIDS-DNN for data plane.

Advantages of these modules compared to basic SDN without any advanced security, as well as existing solutions, were demonstrated by results of our evaluation. Based on these promising results, further development with a focus on the interconnection of these two modules into a more unified solution for combined control plane and data plane security.

Specter, by its functionality, provides early DoS/ DDoS attack detection that also ensures more secure execution of NIDS-DNN module. On the other hand, behavior gained from NIDS-DNN might improve confidence value assignment in Specter. Accordingly, a synergy of unified solution can boost SDN security even further.

In Specter, we still have the potential to improve our rating algorithm. With a focus on detecting slow DoS attacks on the controller, it is possible to gain considerable improvement in security. A comparison of the performance with other scheduling algorithms is an important part of our future work as well.

NIDS-DNN could be more accurate and precise if it could operate with more features. In this phase, it takes into consideration seven features which are computed based on flow statistics. In the future work, we will include more extended features from the UNSW-NB15 dataset and fine tune our model to get even better performance.

ACKNOWLEDGMENT

This project was partially supported by the Eset Research Centre. The authors would also like to thank for financial contribution from the STU Grant scheme for Support of Young Researchers.

REFERENCES

[1] BIAN, Shanshan, ZHANG, Peng and YAN, Zheng. A Survey on Software-Defined Networking Security. In: Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2016. pp. 190-198. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] KANDOI, Rajat and ANTIKAINEN, Markku. Denial-of-service attacks in OpenFlow SDN networks. In: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. IEEE. 2015. pp. 1322-1326.

[3] Open Networking Foundation. [Online] [Cited: 19.02. 2018.] https://www.opennetworking.org/software-defined-standards/specifications/.

[4] IDHAMMAD, Mohamed, AFDEL, Karim and BELOUCH, Mustapha. Dos detection method based on artificial neural networks. *Int J Adv Comput Sci Appl (ijacsa)*. 2017. pp. 465-471.

[5] TANG, Tuan A., et al. Deep learning approach for network intrusion detection in software defined networking. *In: Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on. IEEE.* 2016. pp. 258-263.

[6] TAVALLAEE, Mahbod, et al. A detailed analysis of the KDD CUP 99 data set. In: Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE. 2009. pp. p. 1 - 6.

[7] NIYAZ, Quamar, SUN, Weiqing and JAVAID, Ahmad Y. A deep learning based DDoS detection system in software-defined networking (SDN). *arXiv preprint arXiv:1611.07400.* 2016.

[8] WEI, Lei and FUNG, Carol. FlowRanger: A request prioritizing algorithm for controller DoS attacks in Software Defined Networks. In: Communications (ICC), 2015 IEEE International Conference on. IEEE, 2015. pp. 5254-5259.

[9] MOUSTAFA, Nour and SLAY, Jill. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *In: Military Communications and Information Systems Conference (MilCIS), 2015.* s.l.: IEEE, 2015. pp. p. 1-6.

[10] MISHRA, Preeti, et al. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys & Tutorials.* 2018.

[11] LATAH, Majd; TOKER, Levent. Artificial intelligence enabled software-defined networking: a comprehensive overview. IET Networks, 2018.

[12] SHU, Zhaogang, et al. Security in software-defined networking: Threats and countermeasures. Mobile Networks and Applications. 2016. pp. 764-77