Securing Networks using SDN and Machine Learning

Dragos Comaneci*, Ciprian Dobre*

*Faculty of Automatic Control and Computers
University Politehnica of Bucharest, Bucharest, Romania
E-mails: dragos@comaneci.ro and ciprian.dobre@cs.ro

***Abstract*—Software Defined Networking (SDN) holds the key for building networks that can adapt effectively and efficiently to ever changing conditions: traffic flows, network policies, security constraints, etc. Although it has this power, defining security policies that consider the different scenarios and applications running on the network can be an overwhelming task, even when using high level abstraction languages based on reactive programming.**

In this paper we present a system that alleviates this complexity by using machine learning traffic flow classification techniques and defining high level SDN policies based on the derived flow classes. We employ both supervised learning techniques based on pre-trained models for different types of traffic, and unsupervised learning, where we cluster together different traffic flows. Finally, after classifying the flows, a flow grouping algorithm determines which flows are generally seen together in the same time frame. For supervised learning we use C4.5 decision tree classifiers with features per flow such as inter-packet arrival time, packet size, packet count, flow tuple. For the unsupervised case we use the k-means algorithm on the same group of features.

After gathering the traffic flow information derived via machine learning, we explore how it can be integrated into an SDN controller, and provide an overview of the required hardware and software architecture. From a security standpoint we explore how we can leverage such information for network anomaly detection, botnet detection and traffic re-routing to a network honeypot.

***Index Terms*—Software defined networking; supervised learning; traffic classification; secure networks.**

Security is an aspect of computer networks that is becoming of paramount importance as more and more critical systems and devices are being connected to the network. To this end, traditional signature based intrusion prevention/detection systems are encountering scalability problems because of the increasing attack space [1], [2] and the security industry as a whole is itself reorienting towards machine learning based techniques for detecting attacks [3].

The present work explores how different machine learning techniques and networking technologies can be composed together in order to create a system that can tackle security and networking problems in an adaptive and scalable way. We show how such a system can be organized and built, what readily available components can be used, how they will interact with each other and what types of security solutions can be built on top of the system. We also present the test-bed we have constructed for this system and evaluate system performance as well as classification accuracy for the supervised learning portion on a mix of emulated traffic. By creating an adaptive network control system and detecting anomalies in a network, suspicious traffic can be automatically directed through a series of network appliances that perform deeper inspection by scanning the redirected flows for malware or network attacks.

In order to create such a system, we can leverage the current industry movement towards Software Defined Networking [4] which has been gaining great attention in recent years due to its ability to leverage existing network infrastructure in order to provide better overall network utilization and application specific traffic optimization. SDN achieves these goals by separating the system in charge of making decisions about where traffic is sent (the control plane) from the underlying systems that forward the actual traffic (the data plane). Security issues are also an active area of research for SDN as outlined by a recent survey [5] and the current work complements the work done in this area.

The most basic use of machine learning in the context of network traffic has been for *traffic flow classification*. This type of classification (i.e. identifying the protocol/type of traffic being sent on a certain flow) has been developed as an alternative to traditional deep packet inspection (DPI) and port matching techniques.

Before we begin our discussion about machine learning techniques for network traffic classification, let us first define the concept of *a traffic flow*. In our work, a flow in a TCP/IP based network is a logical communication abstraction characterized by a 5 tuple identifier, namely:
<source port, destination port, source IP, destination IP, transport protocol>

Machine learning can be used to process and classify traffic flows based on externally observable properties of such as: number of packets in a flow, flow duration, packet size, inter packet arrival time, flow size in bytes, etc. Based on these properties, additional learning features can be computed by also considering the average or the maximum of the observed values for a certain property.

Machine learning techniques can be classified roughly into three categories: supervised learning, unsupervised learning and semi-supervised learning, each differentiated by the amount of pre-labeled valid training data that is provided to the algorithm. A notable solution for supervised learning traffic classification that we leverage is Diffuse [6], a traffic classification and control extension to the IPFW tool. Diffuse uses C4.5 decision trees [7] and Naive Bayes [8] machine learning algorithms in order to train its model. The classification models themselves are obtained offline via Weka [9], which is a Java based machine learning framework.

The current work is organized as follows. In Section I we start off by exploring related work similar to our system as well as the concepts and techniques we build upon. In Section II we present an overview of how our system is designed, followed in Section III by an analysis of three security related scenarios. Afterwards, we present details about the architecture of our system in Section IV. In Section V we detail our experimental setup, and in Section VI we present preliminary results. Finally, Section VII outlines conclusions.

I. Related Work

The approach of using machine learning in order to manage networks has been first been proposed in [10], in the form of a knowledge plane for the Internet. There, authors outline the necessary patterns (edge involvement, global perspective, compositional structure, unified approach and cognitive framework) needed to build a self-adapting network. Problem domains like fault diagnosis and mitigation, automatic (re)configuration, overlay networks and intrusion detection are also suggested as being ripe for applying such a system.

A recent direct descendant of the proposed knowledge plane concept comes in the form of knowledge-defined networking [11], where the concept is merged with SDN and network analytics. The feasibility of knowledge-defined networking is explored for the use cases of routing in an overlay network, resource management in an NFV (network function virtualization [12]) scenario, knowledge extraction from network logs and short and long-term network planning.

Yet another approach of using machine learning in managing networks is presented in [13] where the concept of Cognition Based Networks (COBANETS) is established. Compared to the present work, COBANETS expounds the use of a generative deep neural net for generic feature identification using unsupervised learning and then proceeds to tailor the trained network using supervised learning for particular network optimization tasks. While the COBANETS approach is more generic, at the time of this writing it is still under research thus we cannot compare and contrast the two approaches.

Finally, an older system similar in scope to our own (that pre-dates the emergence of the SDN ecosystem) is ANEMA [14]. ANEMA is used for autonomic QoS/QoE management in multi-service IP networks and requires the specification of an action policy, a goal and a network utility function by a network administrator. It also requires an agent (based on Linux traffic control [15]) to be installed on each network device in order to enforce the globally defined policies. Compared to this system we are leveraging the SDN ecosystem for actual network control while traffic feature collection and extraction can either be built inside the SDN switches and exposed via per flow meters or by having a separate module that sees the actual packets that are forwarded by the switch.

While the goal of creating an adaptive network control system is similar to the previous work, both our approach (the ways in which we leverage and build upon supervised and unsupervised learning techniques) as well as our uses cases (anomaly detection, honeypot traffic rerouting and botnet detection) are different. We also detail how such a network management system can be architected and built using open-source readily available components and show that the overhead added by feature collection and model evaluation is manageable.

II. System Overview

An overview of our proposed system for machine learning based flow classification and its integration with an SDN controller is presented in Figure 1. It leverages three methods of deriving knowledge that can be leveraged by the network controller in order to make better decisions: supervised learning for identifying the underlying application of each flow, unsupervised learning for clustering flows in order to identify unknown applications and flow grouping for determining which flows are seen together within the network.

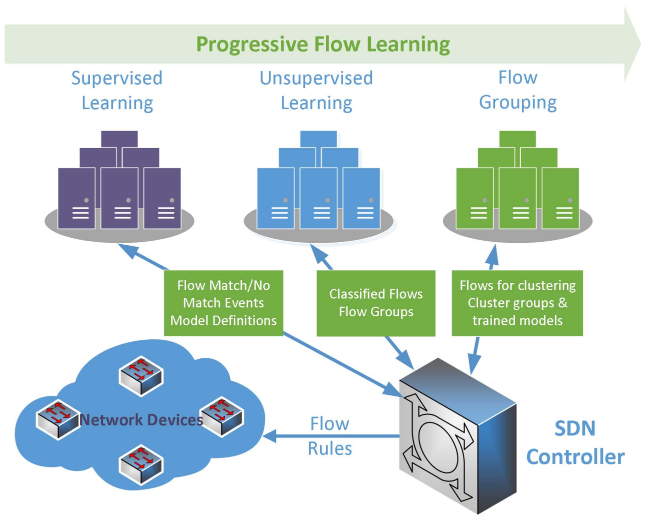
Progressive Flow Learning
Supervised Learning
Unsupervised Learning
Flow Grouping
Flow Match/No Match Events Model Definitions
Classified Flows Flow Groups
Flows for clustering Cluster groups & trained models
Network Devices
Flow Rules
SDN Controller

Fig. 1. Overview of the flow classification steps and their integration with the SDN controller logic.

The first type of classification is based on pre-trained models for supervised learning. If a flow matches one of these models, an event is triggered to the controller notifying it that the flow was classified as being of a certain type. In case the flow cannot be matched against a model, an event is again triggered to the controller informing it that the flow was not matched against any known model (in this case, the controller then proceeds to forward the flow information to the second classification step, namely unsupervised learning).

The controller adds the unclassified flows to a queue, and when sufficient flows are available it triggers the unsupervised learning algorithm. The output from the algorithm is a set of clusters along with a measure of their purity. In case the purity[1] for a cluster is above a certain threshold, and there are sufficient flows in that cluster, a supervised model can also be trained based on it and can be passed on by the controller to the supervised learning classifier.

One aspect to note about the solution is the fact that only the supervised learning part is done online, meaning as traffic comes in to a network device. In our experience with the Diffuse [6] system, this type of classification can be done in high throughput networks even when dealing with large numbers of flows. The other two steps can be run separately, at any time the controller deems appropriate, since they are used to infer new traffic patterns seen in the network.

A. Integrating Flow Type Classification

Our work can be applied on top of modern SDN controller architectures such as Procera [16], that allows operators to express various policies without resorting to general-purpose programming of a network controller. Such controllers are reactive in nature, because many realistic network policies react to dynamic changes in network conditions. As such, Procera and other controllers apply principles of functional reactive programming (FRP) [17], which provides a declarative, expressive, and compositional framework for describing reactive and temporal behaviors.

In order to integrate traffic patterns into such a controller programming model, we first need to define events that will be triggered and delivered to the SDN controller when a match is made against an existing traffic pattern, or when new patterns are discovered in the network. The controller can then compose these events with other event streams and carry out routing logic on the matched flows by installing flow rules on the devices or generating other events and actions to be carried out.

For the case of **supervised learning**, where we have already trained a model based on a predefined traffic pattern, we define the following events and their associated information:

- Model Match event: model identifier, match accuracy, flow 5-tuple, network node
- No Model Match event: closest matching model and accuracy, flow 5-tuple, flow features

The case of **unsupervised learning** builds upon the one for supervised learning, by having the controller add the flows that did not match on the supervised learning models to a queue, and when there are sufficient flows in the queue (or at a specified interval) run a clustering algorithm such as K-means or Random Forest Proximities in order to group similar flows together. Since the flows will all have associated a partial match from the first round of supervised learning, we can also deduce for each group of flows a majority partial match.

Based on the majority partial match of the group, besides deducing a label for the groups of flows, we can also choose to either train a new supervised learning model specifically for that group of flows, or add the flows as training data for the existing model on which the partial match was made. Only one event will be delivered to the controller, a **clusteringCompletedEvent**, which indicates that a run of the clustering algorithm has completed on the unknown flows. The event contains the clusters discovered by the algorithm (and for each cluster: cluster purity, majority partial match, flows in the cluster and their associated 5-tuple information, supervised model trained based on only those flows and learning features of each flow) and the flows that did not fit into the discovered clusters, along with 5-tuple information and learning features.

B. Detecting groups of related flows

Up until now, our integration solution only classified individual flows using supervised and unsupervised learning. However, what we generally observe in the network is *groups of related flows* that occur between nodes when a certain business or human operation is taking place. In order to be able to identify these groups of flows we need to introduce side information in the form of layer 3 source and destination IPs, layer 4 source and destination ports, and source and destination DNS domain names, as well as the time interval in which each flow was observed.

One such traffic flow grouping pattern arises when multiple nodes are connecting and executing the same communication flows against the same node in the same time interval, indicating that the destination node is hosting a common service on the network. This pattern can be detected by grouping flows classified as being of the same type by destination IP and port and then, in each group, sorting the flows based on their start time. Having the flows sorted by start time in each group, we can now select the flows that are within a certain threshold time interval. Once we detect this type of pattern, we can use it to predict that if a certain type of flow is executed against the destination service then, in that time interval, another N flows will be executed against the same service.

We can also detect the opposite case, where a single source node is connecting to multiple destination nodes and executes the same communication flow against them in the same time interval. Detecting this pattern is similar to the first case: we group flows of the same time by source IP, and then within each group we sort the flows by start time. We select next those flows that are within a certain time threshold of each other. Having detected this pattern, we can use it to predict that if a certain source node has executed a specific type of communication flow then it will like execute N more flows of the same type.

III. Security Scenarios

From a security perspective, we explore three different scenarios in which our system can be used, namely: *anomaly detection*, *botnet detection*, and *honeypot traffic rerouting*.

A. Anomaly Detection

Network anomaly detection boils down to constructing a baseline of the network traffic typically seen within a network,

[1]Purity is an external evalaution criterion of clustering quality. It is defined as the percent of the total number of objects (or data points) that were classified correctly, in the unit range [0..1].

and identifying traffic patterns that do not fit into that baseline as suspicious.

In order to accomplish anomaly detection with our proposed system, we (1) continually train and refine supervised models for the traffic flows (as seen in our network). When a new flow does not match any supervised model, we flag it as suspicious and add it to a queue of flows (and their associated ML features) that (2) will get analyzed by the unsupervised learning (clustering) algorithm.

By running the clustering algorithm we can determine if the suspicious flows are close enough to other normal flows that are present in the network, in which case we refine the existing models with the new flows. In case the new flows are something we never seen before, we forward the grouped flows and their characteristics to a network administrator for further inspection.

B. Botnet Detection

In case of a Botnet [18], groups of hosts within the network communicate periodically with a command and control server and receive commands from it that are then executed by the network hosts. In order to detect the botnet automatically, we can leverage the previous anomaly detection scenario. The communication flow between the infected host in the network and the command and control (C&C) server will be flagged as an anomaly. After the infected host receives the command from the C&C server, it performs additional anomalous flows carrying it out, flows that will be picked up by the system.

By correlating the anomalies originated from the same host and timing them we can automatically determine if we have a bot present in the network. The pattern we look for in this case is a first anomalous traffic flow (the botnet command) followed by a group of anomalous flows (the actual execution of the botnet command) different from the first flow.

C. Honeypot traffic rerouting

In the honeypot traffic rerouting [19], we are mainly interested in redirecting traffic from a suspicious host to an emulated host where the traffic can be analyzed in depth. The emulated host will respond to connections initiated by the suspicious host in the same way observed in the initial anomaly that triggered the host to be marked as suspicious.

As before, if a flow does not match any supervised model, we mark the host which initiated it as suspicious and store the flow 5-tuple. Besides the 5-tuple, we also store the actual payloads sent and received by the host, in order to be able to emulate the honeypot as close as possible. This can be done by the SDN controller: in our case, we insert a rule so that the contents of any flow matching the suspicious 5-tuple are saved in a packet capture. The next time the flow is encountered, the SDN controller will automatically reroute it to the honeypot where the emulated host will be able to both respond to connections initiated by the suspicious host as well as individual messages based on the packet capture.

IV. System Architecture

A general overview of the system architecture is presented in Fig. 2. The base of the system is the network controller which is composed of several Nettle SDN Controller [20] VM instances that share a common distributed state data store. Each VM controls a subset of network forwarding elements.

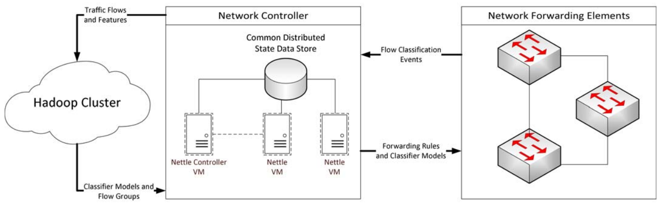
Traffic Flows and Features
Network Controller
Network Forwarding Elements
Common Distributed State Data Store
Flow Classification Events
Hadoop Cluster
Nettle Controller VM
Nettle VM
Nettle VM
Forwarding Rules and Classifier Models
Classifier Models and Flow Groups

Fig. 2. High level system architecture.

The network controller VMs install forwarding rules using OpenFlow onto the network forwarding elements and receive back network events. Besides these normal SDN rules, in order to carry out traffic classification, we also need to send classifier models to the network elements and receive back flow classification events. For this purpose we use Diffuse [6] and its specific protocol. Each network forwarding element has been enhanced with specific ML traffic classification logic for collecting the necessary features and evaluating them against the installed models. In order to carry out unclassified learning, as well as offline-training, for our machine-learning models we leverage a Hadoop cluster and Apache Mahout [21], which is a highly scalable machine learning library built on top of Hadoop.

V. Experimental Setup

As the basis for our experimental setup we used a Mininet VM [22] that simulated four Open VSwitch instances [23], each having a separate connection to each-other. The switches were configured to have one port map to a physical interface of the application server, which was connected to the Ixia Perfect Storm application traffic emulator[2] (see Fig. 3).

The Diffuse Classifier kernel modules were also loaded onto the same Mininet VM. The SDN Controller for the four switches, was placed in a separate VM, not to interfere with the rest of the setup nodes. The Nettle SDN Controller was configured only with the learning switch behavior and had a separate interface through which it received flow classification events from the Diffuse Classifier. The classification events were only recorded during the test, no traffic forwarding decision was taken based on them.

The Ixia Perfect Storm application traffic emulator was used in order to simulate mixes of application traffic that would normally be present in a network. Each type of application flow is emulated based on the protocol specification and can have multiple randomized payload sections.

[2]https://www.ixiacom.com/products/perfectstorm

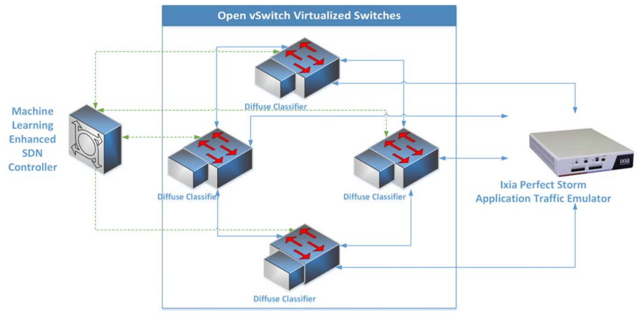
Open vSwitch Virtualized Switches
Machine Learning Enhanced SDN Controller
Diffuse Classifier
Diffuse Classifier
Diffuse Classifier
Diffuse Classifier
Ixia Perfect Storm Application Traffic Emulator

Fig. 3. Testbed architecture.

VI. Evaluation and Results

As of the time of this writing we have explored the parts of the system related to supervised learning and how it behaves in an enterprise network, both with and without malicious traffic (botnet & network attacks). As preliminary research we evaluated the classification accuracy, classification time and resource usage overhead of having the classifier running on the traffic forwarding elements.

As traffic mixes we have used the BreakingPoint Enterprise[3], SOHO/Small Business and Sandvine 2H 2016[4] North America Fixed application profiles, that emulate traffic that would be seen in an enterprise, a small business and a North American internet service provider. An example of the actual traffic types and their bandwidth and number of flows distribution in the application profile are shown for Sandvine in Figure 4. The weights are determined based on yearly traffic reports published by ISPs and aggregated by Sandvine [24].

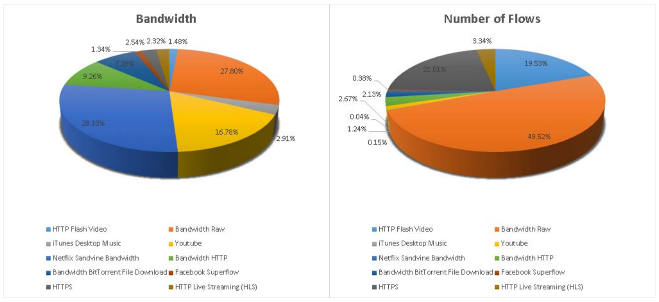

Fig. 4. Sandvine bandwidth and flow distribution.

In order to train the Diffuse C4.5 tree based classifier model we generated 256 different streams (flows with slightly different contents) for each flow type in the application profile. We then proceeded to extract the necessary features (inter-packet arrival time, packet sizes, packet count, flow tuple) for each stream from the generated packet capture. Finally, we trained a decision tree model for each flow type based on the 256 positive training samples as well as 5 other negative samples from each of the other flow types (170 total negative sample).

[3]https://www.ixiacom.com/products/breakingpoint
[4]https://www.sandvine.com/

TABLE I
Classification results for normal traffic conditions.

Profile	TP	FP	FN	Precision	Recall	F-score
Enterprise	17557	4833	2729	0.784	0.865	0.822
SOHO	19523	3753	2020	0.838	0.906	0.871
Sandvine	16341	4730	3825	0.775	0.810	0.792

TABLE II
Classification results for abnormal traffic conditions (attack traffic emulated besides normal traffic).

Profile	TP	FP	FN	Precision	Recall	F-score
Enterprise	11162	7462	2929	0.599	0.792	0.682
SOHO	14390	7755	3781	0.649	0.791	0.713
Sandvine	10216	8166	3979	0.555	0.719	0.627

A. *Classification Accuracy*

To see how the classifier behaves under normal conditions (the traffic being emulated is just the one present in the application profile) vs. abnormal conditions (along with the application profile traffic we also emulate network attacks and botnets), we compute the global F-score for all classifier models, by summing up the true positives, false positives and false negatives values generated by each classifier. In statistical analysis of binary classification, the F-score is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute a score, interpreted as a weighted average of the precision and recall. An F-score reaches its best value at 1 and worst at 0. By summing these values we also automatically take into account the actual flow distribution as described in the application profile (the flows most predominant in the profile should contribute more to the accuracy measure).

The classification results for normal traffic conditions are presented in Table I, while the ones for abnormal conditions are given in Table II. As can be seen, in the first case of normal traffic, the classifiers perform fairly well (the F-score is $\geq 80\%$ since we do not introduce new types of traffic. Once we introduce attack traffic, the F-scores for each application profile drop by $\approx 10-15\%$, but still remain high enough so that the system can still be useful. One reason for the drop is that the extra attack traffic we are sending is similar to the traffic normally found within the application profile. The malicious traffic we emulated during our tests is composed of critical strikes with a Common Vulnerability Scoring System (CVSS) score of 10 (607 different strikes) and botnet traffic (1646 different botnets) which is mainly HTTP based. The Common Vulnerability Scoring System[25] is an open industry standard for assessing the severity of security vulnerabilities.

B. *Classification Time*

For determining the classification time of a flow, we need to be able to answer the question: how many packets do we have to inspect before we can reach a conclusion about the flow type?

The classification time for flows in the Enterprise application profile is presented in Figure 5. Here we can see that simple binary fixed frame-size protocols (such as RTP or NFS) require less packets than more complex text based protocols (HTTP, FTP, SMTP, etc.). In order to determine the minimum number of packets necessary for classifying a certain type of flow we used a confidence threshold of 75% for the C4.5 decision tree algorithm. The results are averaged over 30 streams emulated from each flow type.

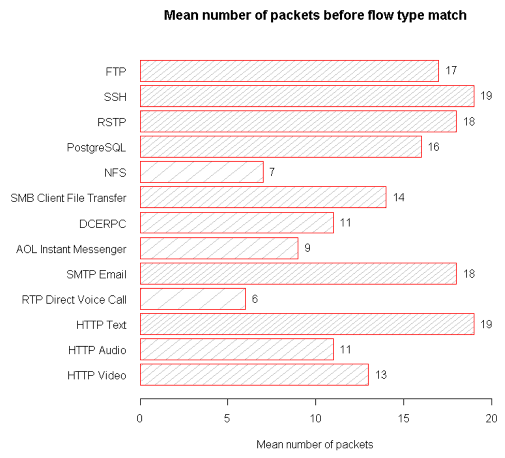

Fig. 5. Mean number of packets required before flow match.

C. Resource Usage Overhead

In order to determine resource usage, we next measured the mean CPU and memory usage for the mininet VM simulating the 4 OpenVSwitch instances in two situations: with and without the Diffuse traffic classifier enabled (and classifier models installed for each flow type). For both measurement situations we ran the BreakingPoint Enterprise Application profile with a frame rate of $\approx 20k$ packets per second and $\approx 80k$ concurrent traffic flows.

With traffic classification enabled, the CPU usage overhead is $\approx 17\%$, while memory usage rises by $\approx 13\%$ when compared with the base case where traffic classification is disabled. The extra CPU and memory are need in order to compute features for each traffic flow as well as match it against the installed models.

VII. Conclusions and Next Steps

In this paper we outlined how we can use machine learning coupled with SDN in order to enable complex network security operations such as anomaly detection, botnet detection and honeypot rerouting. We also delved into the architecture of such a system and how we can use existing solutions in order to distribute and scale both the SDN control part as well as the machine learning operations.

We have also explored the supervised learning portion of the overall system by using an experimental testbed based on Mininet. By using the tested and the Ixia Perfect Storm traffic emulator we measured the classification accuracy measure that we obtained for normal traffic vs. normal plus attack traffic cases and concluded that the classifier remains useful even when faced with network attacks whose traffic pattern is similar to that of normal traffic.

Finally, from a resource usage perspective, the traffic classification overhead is manageable, only adding 13% additional memory overhead and 17% CPU usage overhead when compared with the same system without the classification overhead.

Acknowledgment

The research presented in this paper is supported by projects: NETIO Tel-MonAer (cod SMIS2014+ 105976), SPERO (PN-III-P2-2.1-SOL-2016- 03-0046, 3Sol/2017) and ROBIN (PN-III-P1-1.2-PCCDI-2017-0734).

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

[1] R. H. Weber, "Internet of things–new security and privacy challenges," *Computer Law & Security Review*, vol. 26, no. 1, pp. 23–30, 2010.
[2] J. M. Batalla and P. Krawiec, "Conception of id layer performance at the network level for internet of things," *Personal and Ubiquitous Computing*, vol. 18, no. 2, pp. 465–480, 2014.
[3] K. Patel, "Security survey for cloud computing: Threats & existing ids/ips techniques," in *International Conference on Control, Communication and Computer Technology, 24th*, 2013.
[4] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, 2012.
[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
[6] S. Zander and G. Armitage, "Distributed firewall and flow-shaper using statistical evidence (diffuse)," 2010.
[7] P. Xu and S. Lin, "Internet traffic classification using c4. 5 decision tree," *Journal of software*, vol. 20, no. 10, pp. 2692–2704, 2009.
[8] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 50–60.
[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
[10] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 3–10.
[11] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *arXiv preprint arXiv:1606.06222*, 2016.
[12] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.
[13] M. Zorzi, A. Zanella, A. Testolin, M. D. F. De Grazia, and M. Zorzi, "Cobanets: A new paradigm for cognitive communications systems," in *Computing, Networking and Communications (ICNC), 2016 International Conference on*. IEEE, 2016, pp. 1–7.
[14] H. Derbel, N. Agoulmine, and M. Salaün, "Anema: Autonomic network management architecture to support self-configuration and self-optimization in ip networks," *Computer Networks*, vol. 53, no. 3, pp. 418–430, 2009.

[15] V. Rakocevic, J. Griffiths, and G. Cope, "Linear control for dynamic link sharing in multi-class ip networks," in *17th IEE UK Teletraffic Symposium, Dublin, Ireland*, 2001.

[16] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 43–48.

[17] Z. Wan and P. Hudak, "Functional reactive programming from first principles," in *Acm sigplan notices*, vol. 35, no. 5. ACM, 2000, pp. 242–252.

[18] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection." in *USENIX Security Symposium*, vol. 5, no. 2, 2008, pp. 139–154.

[19] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, 2004, pp. 1–14.

[20] A. Voellmy, A. Agarwal, and P. Hudak, "Nettle: Functional reactive programming for openflow networks," DTIC Document, Tech. Rep., 2010.

[21] A. Mahout, "Apache mahout scalable machine learning and data mining," 2012.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[23] B. Pfaff and B. Davie, "The open vswitch management protocol," 2013.

[24] B. Bilbao-Osorio, S. Dutta, and B. Lanvin, "The global information technology report 2013," in *World Economic Forum*. Citeseer, 2013, pp. 1–383.

[25] K. Scarfone and P. Mell, "An analysis of cvss version 2 vulnerability scoring," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 516–525.