Adaptive Query Rate for Anomaly Detection with SDN

NM SAHRI[†] **and Koji OKAMURA**[††],

Summary
In traditional approach, extracting important features for the application to analyze the anomaly detection problem, introduce significant overhead on the way of switch handling. Furthermore, high volumes of network traffic introduce notable issues that affect the performance and anomaly detection accuracy. Taking advantage of centralized control plane of Software Defined Networking (SDN), the task to handle the flow information is much more simplified programmatically. The accuracy of the measured flow statistic plays important role in anomaly detection. While the use of sampling is capable to lessen the scalability problem of traffic monitoring, the insufficiency of sampled flow statistic may have led to inaccurate detection rate of anomaly. In this paper, we propose an adaptive sampling strategy that is able to provide essential traffic statistics for more accurate anomaly detection in SDN. Our sampling mechanism utilizes the clustering analysis, which is used to classify the attack in the network to determine the severity of monitored traffic. By manipulating the type of service of incoming packet together, these two important parameter formulate our sampling mechanism algorithm. We show experimentally that by putting higher polling frequency on detected anomalous flow, we able to detect network attacks much more accurate.

Key words:
adaptive poll, anomaly detection, network security, OpenFlow, SDN.

1. Introduction

Flow-based method lures the interest from researchers for the network analysis of high-speed networks. With ever increasing load and network usage, clearly scalability is paramount issue to be tackle. In order to perform analysis, statistic features need to be recorded and it expose to unrestraint processing capacity and network bandwidth. For this reason, the accuracy of anomaly detection as well as the need to balance the trade-off between the overhead and the accuracy is crucial.
The emergence of SDN that promise the simplicity in managing networks seems to be the future of the current Internet architecture. In fact, several organizations have adopted SDN in place, most notably Google [1]. By separating the control plane that orchestrated by logical network controller platform and data plane as a forwarding drive, SDN, however, expose to network security threat that can be commence from outside as well as from inside attack of the network.

Network traffic statistics data has been used as inputs for anomaly detection as security analysis is critical for organization or network providers. In this paper, we emphasize on the issues related with the effect of sampling on anomaly detection problem. Network anomaly detection techniques [2] is based on analysis on network traffic and the characteristic of the dynamic statistic features in order to detect network abnormalities quickly and accurately. It is paramount to balance the trade-off between accuracy of anomaly detection and overhead introduced from the flow traffic measurements (e.g.-scalability). Sampling decision should have some intelligent ability to address some of the requirement such as to reach low false alarm and low computation convolution objective. In this paper, we proposed adaptive sampling decision for the controller to capture the most dominant service type of DDoS attack in the network where we aim to provide accurate anomaly detection while at the same time lowering the number of false reported alarm. Our sampling decision method ensure malicious flows that come from commonly used attack service port is given higher priority than others, thereby addressing the accuracy parameter.
The remainder of this paper is organizes as follows. Previous related work is discussed in Section 2 then the following Section 3 gives a brief detail of our adaptive anomaly detection architecture framework. Our flow collector method is presented in Section 3.1 followed by Section 3.2 where we explained important network features that are crucial for our anomaly detection. The methodology and algorithm of our proposed sampling decision technique is explained in Section 3.3. The performance evaluation results are shown in Section 4. Finally, we conclude our discussion in Section 5.

2. Related Work

Previously, many researchers started to give attention to the impact of sampling method on anomaly detection. [3], the author has signified that the packet sampling degrades the effectiveness of anomaly detection and dramatically increase the false positives. In [4], the author compares two type of sampling which is random flow sampling [5] and sample-and-hold technique [6]. The result shows that random flow sampling performs best for anomaly detection. In [7] the authors used flow-based metrics such as the number of source IP addresses for the detection and

Manuscript received June 5, 2016
Manuscript revised June 20, 2016

proved that the accuracy and performance of the anomaly detection depends mainly on the sampling rate applied and the author also proved it is less dependent on the sampling technique used.
Intrusion detection problem has not drawn so much intention from researcher despite substantial amount of work in Openflow. The eminent work regarding anomaly detection in SDN is reported in [8], [9] and [10]. In [8], the author utilizes the Openflow architecture for detecting the Distributed Denial of Service (DDoS) on the data plane. Periodic sampling is used for flow statistics collection retrieval and Self-Organized Map has been exploited for the intrusion detection classification. In [9], the author uses multiple type of anomaly detection algorithm in their research test where the author validates their algorithm in Small Office/Home Office (SOHO) environment. The author utilized Openflow for detecting network security problem close to the source of abnormality using the idea of decentralization control of the network devices. The author also uses periodic sampling for the flow statistics collection. Contrast to previous work, author [10] decoupled the controller communication channel with Openflow switches where the sFlow flow statistics collection method is used and the native Openflow communication channel is used only for the forwarding purposed separately. The experimental results show significant reduction in flow table size and the control plane load. However, the method used by the author increase the false positives percentage in the intrusion detection.

3. Adaptive Query Rate Methodology

To provide scalable and accurate sampling decision, we considered two important parameters which is the anomaly detection and the traffic measurement. In our work, two important method applied to the incoming packet into the controller. At first, the step is to be able to classify the incoming packet is either anomalous or normal traffic while the second step is responsible for our flow sampling decision. Our proposed architecture has been generalized in Fig. 1 where it contains 4 main components namely as Flow_Collector, Flow_Processing, Sampling_Decision and Forwarding_Logic modules. The components are developed to be part in the POX controller [11] that we use for our simulation. In general, Flow_Collector module collect all active flows in the network while the Flow_ Processing carried out flow-level analysis to find the anomaly behavior from the network flow inspected.
The module also responsible to provide attack event and also the type of service of the incoming packet that has been identified. This two information are sent to the Sampling_Decision module for the sampling decision making. The consideration of both the type of service port number and the condition of traffic, our sampling decision is defined in such a way that any malicious traffic service is given higher priority to be queried and sampled. This is to ensure to improve the accuracy of the anomaly detection in SDN. We describe each of the components function precisely in the following section.

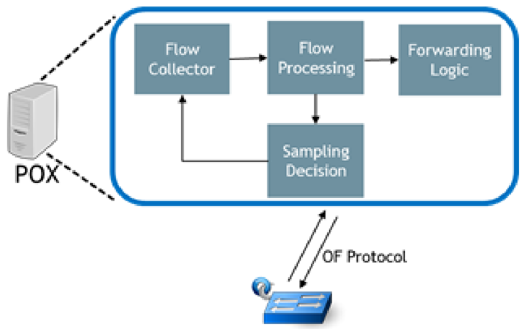
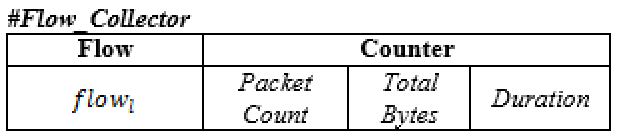


Fig. 1 Main Skeleton of proposed architecture

3.1 Flow Collector

It is important to describe how Openflow mechanism works when a new packet of a particular flow arrive in the switch. First, when any new packet reaches the switch, matching process operation are performed. The switch will check whether the flow entry that installed previously match with the incoming packet. If there is a match, the packet will follow the action set for the related flow entry. Otherwise, the switch will send the packet to the controller for further decision making. The controller receives the header information from a control message namely as Packet_In from Openflow switch which is result from unmatched flow in the switch flow table. Furthermore, the controller will perform necessary decision and install the rules for the flow into the switch flow table as a flow entry by using FlowMod control message. The switch then forwards the packet to their destination and subsequent packet of that flow are forwarded without interrupting the controller. In our proposed work, we record the flow information in an active flow table in the controller that consist of all current active flows in the network. Our structure of Flow_Collector can be described as in Fig. 2.



#Flow_Collector

Flow	Counter		
$flow_1$	*Packet Count*	*Total Bytes*	*Duration*

Fig. 2 Active Flow Collector Table in Controller

The table contains 2 main field namely as Flow and Counter. The Counter field is further divided into

Packet_Count, Total_Bytes and Duration field. After the controller receive the incoming Packet_In and perform the rules and forwarding decision, our method adds the new information to a list of active flows, flowl (in Flow field) which is associated with the incoming port of flow. Then, controller installs the associated rules as a flow entry in switch flow table via FlowMod message. When the flow information has been stored, the controller will send ofp_flow_stats_request control message to every switch that has information about the active flow in order to update the controller with their necessary statistic. The statistics are as follows: we record the number of packet per flow, total bytes per flow and duration of the flow. Controller kept previous time-window value record for the Counter and perform calculation when get new statistic to update current time-window value. Our active flows Counter are updated in controller when the Openflow switch reply to request message.

3.1.1 Feature Extractor

Selection decision of different set from network traffic feature sets to be used is common problem in anomaly detection. As an example, various types of features are widely known such as detection based on packet headers, application layer protocol or content byte streaming. In Openflow, the controller has the ability to check the packet header information. When the switch cannot match the incoming packet with their predefined flow table, it will forward the packet header information to the controller for further action. Our Feature Extractor module extract flow information from the Active_Flows module which are vital for our anomaly detection classification. All of the important features are derive from the packet header (source port, destination port, source IP, destination IP and protocol). In Openflow, we can get that information from control message namely as Packet In. The important features are grouped into 5-tuples flow information in a hash table. In this stage, all features selected is most likely that influence the judgment to classify network traffic as normal or as an attack.The 5-tuples features are as follows:

1. *Flow Byte*, B_l - the number of bytes of a particular flow capable to provide us a useful information for anomaly event in network, such as port scan, and it is normally small in size in order to increase the coherence of attacks.
2. *Flow Size*, F_l - IP spoofing is one of main example of DoS attack that make the task to detect the true source of spoofing is nearly impossible. The normal operation of spoofing usually generates flows with a small number of packets. This contradicts from normal network traffic where it usually generates a large number of packets for a particular flow.
3. *Number of different flows to same Destination IP*, $n(D_{IP})$- Flood attack are created to consume the resources of victim host and usually will generate a high number of flows. This feature will calculate the number of flows to same victim's destination IP address.
4. *Number of flows to different Destination Ports*, $n(D_{port})$- port scan attack is a process that send requests to a number of server port addresses on a particular host. The aim of this attack is to penetrate an active port on that host and any large number of different destination port indicate the abnormality and shows higher possibility of the network are under attack.
5. *Number of different Source and Destination pair*, $n(SD_{pair})$ 1– this feature able to spot the port and network scans as well as distributed type of attacks, which spike the number of source and destination pairs. We define this 5 features as $X_t = f\left(F_l, B_l, n(D_{IP}), n(D_{port}), n(SD_{pair})\right)$ where X_t is the observed value at time t. Furthermore, this 5 features selected vector fed to our *Anomaly Classifier* module. We purposely choose this 5 features since the number of packet and bytes of a flow allow us to detect anomalies in traffic volume while the others will show increment values in the number.

3.1.2 Anomaly Classifier

There are many notable algorithms that has been successfully proven to classify network traffic for anomaly detection [12, 13]. In our simulation, we adopt K-mean algorithm as our anomaly classifier for simplicity and brevity purpose. This algorithm has the ability to learn and detect anomalies from the audit data without the intrusion signature which is usually provide by the security expert. The advantage of this machine learning algorithm is it can automatically identify groups of similar objects in the training dataset. This clustering algorithm groups multiple objects into predefined K disjoint clusters.
We summarize the steps of performing this algorithm in the followings:

1. Define the number of *K clusters*. In our anomaly detection problem, we set the ***K=2*** where we assume that legitimate and anomaly network traffic features are from different cluster in space.
2. Initialize the randomly chosen *K* clusters and set to be as *centroid* (center of cluster).
3. The calculation process begins to find the distance from each objects to all centroids using distance function method where the algorithm

1 We consider port numbers and IP addresses in utilizing this feature

continues to read each objects from the data set and assigns it to their nearest cluster.

4. Recalculate/iteration process is done after every new objects insertion to the algorithm to get the new cluster centroids.
5. Step 3-4 are repeated until the centroids do not change.

In this algorithm, distance function is required to calculate the similarity between two different objects. The following equation is Euclidean function which is commonly utilized to compute the distance where $a = (a_i, \dots a_m)$ and $b = (b_i, \dots b_m)$ are the two input vector with m features.

$$d(a,b) = \sqrt{\sum_{i=1}^{m}(a_i - b_i)^2} \quad (1)$$

Using this function however, the features must be normalized first since the features are usually measured with different metrics. For the evaluation of our proposed adaptive anomaly detection method, we use weighted Euclidean function as in the following equation:

$$d(a,b) = \sqrt{\sum_{i=1}^{m}\frac{(a_i - b_i)^2}{s_i}} \quad (2)$$

The $s_i$is weight factor and empirical normalization and of the i^{th} feature. The classification of the network traffic is done by the controller where it utilizes this algorithm to detect the anomaly. Whenever the 5-tuples features are classified as attack, alerts are notices to an administrator.

3.2 Sampling Decision

Accuracy and efficiency are two important factors that our formulation for the sampling decision is based on. The effect of polling rate to anomaly detection and traffic measurement derived from accuracy parameter. Higher polling rate is favorable to accurately detect the network traffic abnormality within short period of time. On the contrary, efficiency factor denotes the effect of the polling method to the controller memory and CPU resources. Since high polling rate in the network lead to large number of sampled flows, it is crucial for the controller to have the ability to vary the polling rate so that it will not drain the resources. Therefore, our sampling decision must have the ability to dynamically adjust the *polling rate*, $S[t_i]_l$ based on previous stated two parameters.

3.2.1Anomaly Detection and Traffic Measurement

According to a report from Akamai [14], the concentration of attack traffic is increased during the second quarter of 2013 where the increased concentration was driven by indicatively increases in attack volume targeting Ports 80 (WWW/ HTTP) and 443 (SSL/HTTPS). For our objective of traffic measurement accuracy, we classify common attack port as the commonly used source of attack port service over the overall flows population. We favor sampling to flows that had been attack with commonly used source port. Consider a set of m flows of various source port service, $src_{port} = \{src_{(port)l} : l = 1, 2, \dots m\}$ where $l \subset m$. If the source port service of a particular flow, $src_{(port)l}$ is port 80, we assume that flow is using commonly used attack source port service. Priority is given to the network traffic based on severity level and the service port of the particular flows where we define any attacked flow with source port service is port 80 is given highest priority. Given a flow with attack probability/event $\hat{A}_l$ and the source port service of the flow is $src_{(port)l}$, the prioritization can be expressed as following equation:

$$Pr(\hat{A}_l) = x_l * \omega_l \quad (3)$$

We define x_l as $\{(\hat{A}_l) * src_{(port)l}\}$ and ω_l is a weight given to them x_l. From the equation above, a large $x_l$value denote the severe network attack on a flow with commonly used attack port service. Since both information are known parameters, the ω_l value is constructed in such a way that higher value is given to abnormal flows. Thus, we ensure that the flow with the priority is given more precedence compare to other flows. With the network dynamically change from time to time, it is very challenging to determine the exact value of the weight ω_l , for that reason and also for the simplicity, we manually define the value of the weight. The appropriate value of the weight of a particular flow can be defined by using any other heuristic algorithm. After the above steps are completed, our polling rate, $S[t_i]l$for a particular flow are decided as following equation:

$$S[t_i]l \begin{cases} \frac{T}{\alpha} \; for \; Pr(\hat{A}_l) \\ \frac{T}{\beta} \; for \; (\hat{A}_l) \\ T * \partial \; for \; normal \end{cases} \quad (4)$$

For flows with priority $Pr(\hat{A}_l)$, we set to poll the statistic information with higher frequency, $\frac{T}{\alpha}$ and for the flow with attack that has lower severity, we poll the flow information lower than the higher priority flow. We leverage the accuracy and scalability of the sampling decision by lower down the poll frequency for legitimate traffic, $T * \partial$. The decision for ofp_flow_stats_request scheduling timer are set within predefined minimum and maximum timeout value where $T_{min} \leq \propto, \beta, \partial \leq T_{max}$. The pseudo-code is given in Table 1. Note that our sampling decision favor flows with certain bias criteria where higher priority is given to malicious flows with commonly used attack service port number.

Table 1 Proposed Adaptive Polling Algorithm

```
Algorithm 1: Polling Rate, $S[t_i]l$
Input: $x_t$, Feature: Active_flows, A ; $S[t_i]l$ ; C, control
message {if $C = C_1(Packet_{In})$,
 if $C = C_2(ofp\_flow\_stats\_request)$,
 if $C = C_3\left(ofp_{flow_{stats_{reply}}}\right)$,
$T_{min}, T_{max}$
if $C = C_1$
    store $flow_l$ into $A$
     for all $flow_l$
        send $C_2$ to $flow_l$ switch, $S[t_i]l = T$
    end for
end if
else if $C = C_3$
     for all $flow_l$ in $A$, extract input $x_t$
     execute intrusion_detection_module (Algorithm 2)
        then
           if $P_r(\hat{A}_l)$ then $S[t_i]l$ at $\frac{T}{\alpha}$ end if
           else if $(\hat{A}_l)$ then $S[t_i]l$ at $\frac{T}{\beta}$ end if
     else if normal traffic then $S[t_i]l$ at $T * \partial$ end if
    end for
end if
```

After the controller receive the Packet_In message, it will send FlowMod message to install the flow entry into the related switch. Then, we utilize Openflow standard message type [15] OFPMP_FLOW_STATS request which is sent from controller to switches. Our poll scheduling algorithm will start to send message to Openflow switch requesting the flow statistics information. Furthermore, the classification of anomaly is done where the anomaly flow will be marked as $\hat{A}_l$, the sampling decision is made. The process of the sampling decision is simplified for viewing in Fig. 3.

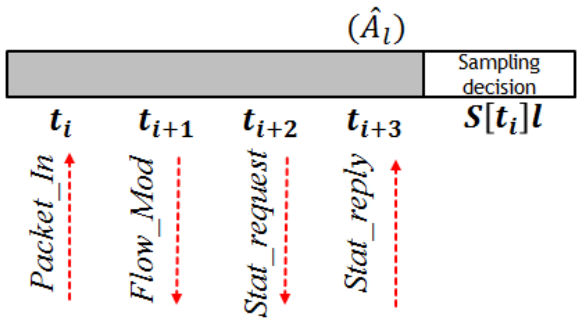


Fig. 3. Sampling Decision in Timeline

4. Performance Evaluation

In this section, we present our experimental setup and performance of our proposed anomaly detection method. We focus on the accurateness level of anomaly detection with our adaptive poll method and perform comparison with static poll mechanism. We also measure the CPU performance for the controller in order to leverage the possible overhead introduced using our proposed technique. We used Mininet [16] to emulate the network attack consisting of Openflow switches, links and hosts on a single machine. We provide details of the experimental setup in Section A followed by explanation about dataset and traffic generation that we used in our test Section B. In Section C and D, the results are presented.

4.1 Experimental Setup

In our anomaly detection method, all of the algorithm is implemented on POX controller which is written in Python language. For the simulation purpose, we choose Mininet network emulator version 2.2.0 with software switch availability that support the Openflow standard software switch which is OpenvSwitch [17]. We ran our experiments to emulate the network attack scenario, as well as to train the K-mean classification algorithm, on a system with an Intel core i3 CPU and 8 GB RAM memory capabilities. Fig. 4 shows the topology setup and the network attack scenario that has been used for our simulation. Victim network consist of three Openflow standard switches that connected to POX controller via Openflow protocol channel. We configure the link between Victim and Attacker network via a gateway with 1 Gbps bandwidth and 20 milliseconds of delay and all other links are assumed to have 100Mbps bandwidth. The network attack simulated is assumed origin from outside of Victim network and all Victim host is connected directly to Openflow switch 3 (OFS3).

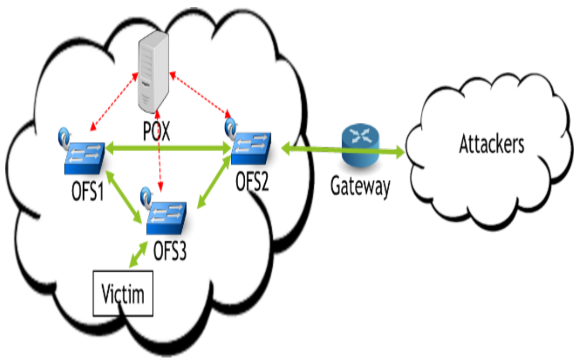


Fig. 4. Simulation topology.

4.2 Dataset and Traffic Generation

In order to simulate high traffic network, we use CAIDA benign Internet trace [18] aiming to evaluate our anomaly detection method with real network environment. Since the dataset size is huge, we extract only 10% from the data which make up approximately 110Mbps of packet and almost 6000 flows, enough to simulate high traffic behavior of high traffic network. This dataset was used to evaluate the accurateness level of anomaly detection with the adaptive method proposed. We use Tcpreplay tool [19] to replay the extracted CAIDA dataset in the Mininet. This tool has the ability to do editing and replaying previously captured network traffic and initially it is design to replay the malicious network traffic patterns to Intrusion Detection/Prevention Systems.
For the network attack traces, we utilized Scapy [20], a computer network manipulation tools written in Python. This tool allows us to generate sequence of traffic randomly, thus it can be used to simulate attack traffic behavior. For portscan attack scenario and to imitate the commonly behavior of the attack, we generate and injected packet with specific source and destination IP address. Furthermore, the source and destination ports were randomly selected in each packet generated. Next, to emulate the DDoS attack, SYN packets with a set of predefined destination port and IP address, together with a constantly changed and random set of source port and IP address.

4.3 Training Time and Traffic Classification

In our anomaly detection, a model that represent the normal behavior of a particular network is constructed. We train the benign CAIDA dataset with the weighted K-Mean algorithm to learn the normal behavior of the data. For the testing phase, we manually inject the attack packet and let the weighted K-Mean algorithm differentiate and classify the attack packet as anomaly. In Table 2, we present the training and classification time take by the algorithm to perform task such as training time and classification of the sample. From the 5 data feature set that we used, the training time takes around 7 hours and the classification time takes around 315 miliseconds.

Table 2. Dataset training and classification time.

	Weighted K-Mean Training	**Weighted K-Mean Classification**
	Hrs	Ms
5 tuples	6,37 hs	314

4.4 Accuracy and Anomaly Classification

Three important factor to evaluate our proposed mechanism is considered: (i) average network traffic rate, (ii) the number of attack packet per second and (iii) polling rate as shown in Table 3. We used a real 110Mbps Internet dataset derived from CAIDA. We injected attack packets that emulate the DDoS and port scan attack at different packet rate. For our experiment we replayed the benign 110Mbps dataset while injecting DDoS and port scan.

Table 3. Parameter values used in experiment.

Average Traffic Rate (Mbps)	**Attack Rate (pps)**	**Poll Rate**	
110 Mbps	200,350,500 pps	Every 5 seconds (static)	$S[t_i]l$

Our objective in this experiment is to have a better accuracy in detecting network anomalies by doing comparison using two different kind of network polling rate mechanism. For the first experiment, we manually set the polling rate to collect the network statistic from the Openflow switches at every 5 seconds and the next experiment we tested our proposed $S[t_i]l$ mechanism. In anomaly detection problem, Receiver Operating Characteristic (ROC) curve is usually used to measure the performance of the method. The ROC curve is a plot of intrusion detection accuracy against the false positive probability. In Fig. 5 and Fig. 6, we present the ROC curves that we have experimented with two different type of attacks with different polling rate mechanism (Table 3). In this first experiment, we inject 200 network attack packet per second.

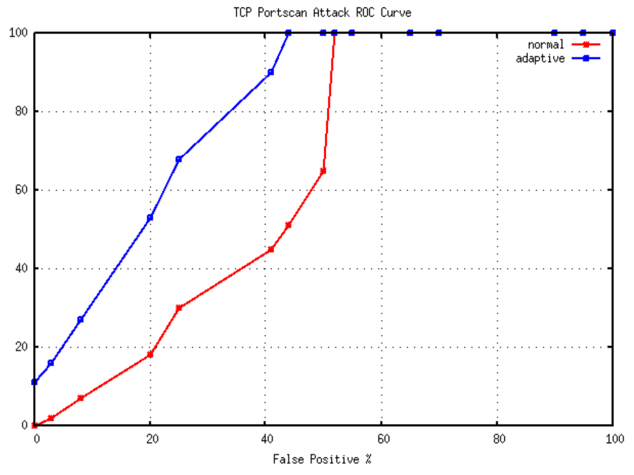


Fig. 5 ROC Curve for TCP Portscan attack.

Fig. 5 depicts the ROC curves for TCP Portscan attack with the static and $S[t_i]l$ algorithm. We set the polling rate value of $\alpha = 3, \beta = 2$ and $\partial = 1.1$. From the graph, the K-Mean anomaly classification algorithm achieve nearly 100% anomaly detection accuracy for both type of polling rate mechanism. For the static polling rate, the False Positive is approximately almost 52% whereas our proposed adaptive polling rate implementation performed better where the False Positive of is almost 43%. This clearly shows that while the detection rate is almost identical, our proposed method able to reduce the False Positive factor where the legitimate traffic classified as attack which can lead to different action taken from the network administrator. Furthermore, it also can lead to unnecessary network service disruption for the real customers.

In Fig.6, the ROC curve illustrate our experiment with DDoS type of network attack. In this experiment, we also injected 200 attack packet per second. When we experiment the static polling rate, the False Positive is approximately almost 45% and when we tested our algorithm, our proposed adaptive polling rate implementation performed better where the False Positive value significantly drop to almost 34%. The main achievement of our method is that when using adaptive poll rate, the anomaly detection rate is much faster and more accurate than normal poll rate thus enable administrator to alert/mitigate anomalous or suspicious packet efficiently.

With the adaptive poll rate proposed, the algorithm might force the POX controller to perform more computation processing thus could increase the CPU processing time since the controller need to handle anomaly detection and forwarding decision at the same time. Furthermore, it also could increase the communication between the controller and all Openflow switches under control. We analyzed the impact of the algorithm proposed on controller CPU processing in the following Section 4.5.

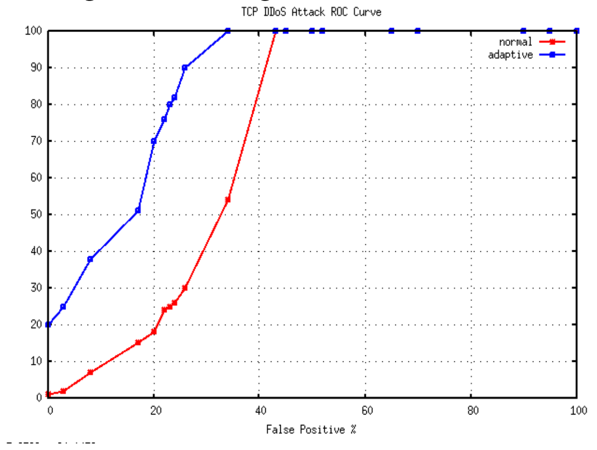


Fig. 6. ROC Curve for DDoS attack.

4.5 CPU Performance

From our first experiment, where we use 110Mbps of traffic rate together with 200 attack packet injected per second, we further test the performance of the controller to find the possible overhead that might introduced. We measure and compare the system resources of the controller with two types of polling rate. From Table 4, we depict the positive factor of the adaptive approach versus the static polling rate approach. We perform two type of test where the first experiment is tested without the attack injected. In this first experiment, we make comparison between the static and adaptive polling rate with the objective is to find the average CPU percentage introduced while performing the polling mechanism. As we can see, the required CPU cycles for our adaptive polling rate with the weighted K-mean classification algorithm is reduced to only 45% when compared to the respective static polling rate approach (57%).

For the next experiment, we inject 200 attack packet that depict the Portscan and DDoS attack while replay the 110Mbps traffic rate. During the attack phase, the static polling rate CPU utilization is increased from 57% to 74% (average of 17% increment of CPU power needed to perform the algorithm. While with our adaptive method, the CPU increase from 45% to 61% (14% different). As shown in Table 4, we can achieve a slightly decrease in the CPU cycle usage of the POX controller with our adaptive polling rate methodology. In our method, even though we poll more frequently for flow that classified as attack by the weighted K-Mean algorithm, at the same time we leverage the polling rate for flow that not classified as attack to be more relaxed. By doing this way, we can achieve lower increment in term of CPU usage percentage for the controller.

Table 4 CPU performance comparison between the static polling rate versus our proposed adaptive rated methodology.

	110Mbps traffic (%)	**110Mbps with 200 attack pps (%)**
Weighted K-Mean normal poll rate	57	74
Weighted K-Mean adaptive poll rate	45	59

5. Conclusion

We presented our work on developing more accurate intrusion detection mechanism for the network attack in SDN paradigm, ultimately allowing better defense against the network cyber-attack for an organization. We showed that the adaptive query rate anomaly detection is able to detect the abnormal traffic behavior much more accurate compared with static interval polling rate time. We prove that by using the adaptive method, the False Positive is reduced significantly. Furthermore, by relaxing the polling rate for traffic that not classified by our method as abnormal, we only introduce small increment in CPU percentage compared to static polling rate that does not differentiate any type of flows.

As proven from our simulation, the classification K-Mean algorithm did not achieve 100% detection rate for the injected attack packets. To be exact, the algorithm only able to detect 97.82% from total manipulated attack packets. We strongly believe that this algorithm are not suitable to be used as any DDoS attack defense mechanism for classification. The important achievement in this work is to prove that by giving higher polling frequency to any high probability attack flows from the network, we are able to detect more accurate attack flows as opposed to the previous related work [8, 9] that use fix time periodic sampling for all type of flows.

Acknowledgments

The author would like to thank the Ministry of Higher Education of Malaysia and University of Technology MARA Malaysia for financially supporting this research under Contract KPT(BS)790405085321.

References

[1] S. Jain, A. Kumar, S.Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J.Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software DefinedWAN,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 3–14, Aug. 2013

[2] P. Barford and D. Plonka, “Characteristics of Network Traffic Flow Anomalies,” Proc. 1st ACM SIGCOMM Internet Measurement Wksp, San Francis- co, CA, Nov. 2001, pp. 69–74.

[3] MAI J., SRIDHARAN A., CHUAH C.N., ET AL.: ‘Impact of packet sampling on portscan detection’, IEEE J. Sel. Areas Commun., 2006, 24, (12), pp. 2285–2298

[4] MAI J., SRIDHARAN A., CHUAH C.N., ET AL.: ‘Is sampled data sufficient for anomaly detection?’. Internet Measurement Conf., Rio de Janeiro, Brazil, October 2006, pp. 165–176

[5] DUFFIELD N.G., LUND C.: ‘Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure’. ACM SIGCOMM Internet Measurement Conf., Miami, FL, USA, October 2003, pp. 179–191

[6] ESTAN C., VARGHESE G.: ‘New directions in traffic measurement and accounting’. Proc. SIGCOMM’02, Pittsburgh, PN, USA, August 2002, pp. 323–336

[7] ANDROULIDAKIS G., CHATZIGIANNAKIS V., PAPAVASSILIOU S., ET AL.: ‘Understanding and evaluating the impact of sampling on anomaly detection techniques’. IEEE Military Communications Conf.,Washington, DC, USA, October 2006

[8] Rodrigo Braga, Edjard Mota, Alexandre Passito, Lightweight DDoS flooding attack detection using NOX/OpenFlow, in: LCN ‘10 Proceedings of the 2010 IEEE 35th Conference on Local, Computer, 2010, pp. 408–415.

[9] Syed Akbar Mehdi, Junaid Khalid, Syed Ali Khayam, Revisiting traffic anomaly detection using software defined networking, in: RAID’11 Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, 2011, pp. 161–180.

[10] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” Computer Networks, vol. 62, no. 0, pp. 122 – 136, 2014.

[11] POX.’An Openflow Controller’, Online Referencing, http://www.noxrepo.org/pox/about-pox/ (2008, accessed May 2015).

[12] Y. Gu, A. McCallum, and D. Towsley, “Detecting anomalies in network traffic using maximum entropy estimation,” in Proc. Internet Measurement Conference, 2005

[13] RAMADAS, M., OSTERMANN, S., AND TJADEN, B. C., “Detecting anomalous network traffic with self-organizing maps.” In Proceedings of the Conference on Recent Advances in Intrusion Detection. 2003, 36–54

[14] Ed. Belson David, “The State of the Internet,” Volume 6, Number 2, Akamai Internet Quarterly Report, Online Referencing, http://www.akamai.com/dl/documents/akamai_soti_q213.pdf (2013, accessed March 2015).

[15] Open Networking Foundation, "OpenFlow switch specification, version 1.3.", Online Referencing, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (2012, accessed April 2015).

[16] Mininet, “An Instant Virtual Network on your Laptop”, Online Referencing, http://mininet.org (2012, accessed April 2015).

[17] Ben Plaff et al., Extending networking into the virtualization layer, in: 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York, City, 2009.

[18] CAIDA, “The CAIDA UCSD Anonymized Internet traces 2013.”, Online Referencing, http://www.caida.org/data/passive/passive_2013_dataset.xml (2013, accessed August 2015).

[19] Tcpreplay, Online Referencing, http://tcpreplay.synfin.net(accessed June 2015).

[20] SCAPY, Online Referencing, http://hg.secdev.org/scapy (accessed June 2015).

Nor Masri Sahri received his first Bachelor Degree (B. of Information Technology) from Northern University of Malaysia on 2001 and obtained his Master Degree (MSc. of Information Technology) from University of Technology MARA on 2006. He has 6 years of industrial experience in one of the largest network service provider in Malaysia working as Senior Network Engineer. He is currently a Ph.D. student and belong to the department of Advanced Information Technology, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan.

Koji Okamura is a Professor at Department of Advanced Information Technology and also at Computer Center Kyushu University, Japan. He received B.S. and M.S. Degree in Computer Science and Communication Engineering and Ph.D. in Graduate School of Information Science and Electrical Engineering from Kyushu University, Japan in 1988, 1990 and 1998, respectively. He has been a researcher of MITSUBISHI Electronics Corporation Japan for several years and has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan and Computer Center, Kobe University, Japan.