JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 807-821 (2017)
DOI: 10.6688/JISE.2017.33.3.13

An SDN-based Sampling System for Cloud P2P Bots Detection

JIANG-YONG SHI[1], JIE HE[2,+] AND YUE-XIANG YANG[1]
[1]*School of Computer*
National University of Defense Technology
Changsha, 410072 P.R. China
[2]*Department of Information Engineering*
Officers College of PAP
Chengdu, 610213 P.R. China
E-mail: {shijiangyong; yyx}@nudt.edu.cn; jack.237@163.com

Cloud network monitoring is a crucial problem in protecting cloud security. As the traffic is huge and the network structure is dynamically changing, it is hard to monitor collaborative attacks such as P2P botnets. This paper presents a two-stage sampling system based on SDN, which is able to extract security related packets from the vast cloud traffic thus reducing the performance cost. We implement a prototype of the sampling system to detect P2P bots in cloud. The prototype is evaluated with real-world P2P botnet traffics. The experimental results demonstrate that our method can identify potential P2P bots quickly and accurately with few false positives and high detection accuracy at an acceptable performance cost.

Keywords: sampling, SDN, cloud, P2P botnet, security

1. INTRODUCTION

With the development of cloud computing, more and more enterprises and government organizations are moving their IT infrastructure into cloud. There is an urgent demand for network monitoring to protect the security of digital assets and enterprise network from being attacked. However, due to the inability of efficiently inner-cloud network traffic monitoring, it's very likely that one insider attacker could attack or infect the others on the same cloud or even the same host machine where their virtual machines co-exist. Besides, the homogeneity of virtual machines facilitates the spread of P2P bots on cloud servers, which might cause deny of service (DoS) to the server, and the bots could hide them through the cover of the cloud.

The dynamic nature of cloud computing makes it hard to statically monitor the network. Virtual machine migration and dynamic resource scheduling make the network structure unstable state. Software Defined Network (SDN), which is proposed to manage the dynamic cloud network, is used to implement security-related applications, such as DDoS prevention and IDS. However, one of the main problems of SDN-related security applications is the performance bottleneck of the controller. As a centric model, SDN controller is crucial in collecting security related information of the network, which would get overload when the traffic is large, especially in a cloud environment. Therefore, sampling is required to reduce the performance cost to the controller.

To meet these demands, we propose a two-stage sampling system to monitor net-

Received July 17, 2016; accepted October 10, 2016.
Communicated by Ce-Kuen Shieh.
[+] Corresponding author.

work status in cloud. In the first stage, we capture the package header information in a set of time windows, which can be easily accomplished by SDN flow query commands. The header information consists of five-tuples, *i.e.* source IP address, destination IP address, source port, destination port, and IP protocol type. Based on this simple information, we can use a set of algorithms to roughly judge the network status and get a small set of suspicious flow and host. For example, we can use Sample&Hold [1] and Lightweight [3] to detect elephant flows and DDoS traffic respectively. In the second stage, we can use the SDN controller to push static flows to block these suspicious traffics, or redirect them to deep packet inspection (DPI) engine for further analysis, including P2P bots detecting.

Compared with mirror dispatching method, which needs expensive equipment and typically delays in detecting threats, our method is more economical and efficient in terms of performance and cost. What's more, we use a modular design to enable the future expansion for security applications. For example, we can use different modules to detect different threats, such as P2P, IDS, DDoS. In addition, the Cloud Service Provider (CSP) can benefit from using the centric controller model of SDN to detect collaborative attacks such as DDoS and P2P botnet.

To test the performance of our system, we implement a P2P botnet detection module to effectively detect P2P bots in cloud. As our module is working online, it is very timely compared with other offline methods. Besides, with the help of SDN controller, we can response quickly based on the detection results. For example, we can block the P2P botnet traffic by updating the switch flows as long as we detect it.

Our main contributions include:

1. We propose a cloud traffic sampling system based on SDN, which is able to monitor real-time network traffic of cloud and detect collaborative network attacks.
2. We introduce a two-stage sampling rate calculate algorithm and implement it with OpenFlow protocol, which aims to use limited resources to sample as many security-related packets as possible.
3. We implement a prototype system, and evaluate it with real-world P2P botnets traffic datasets. The experimental results show that our system is able to identify potential P2P bots in cloud quickly with high accuracy and few false positives and greatly increase the proportion of botnet-related packets in all sampled packets.

2. RELATED WORK

SDN is flexible for its detachment of control plane and data plane. Users can write their applications easily with the data collected by controller. Existing works include Firewall [2], DDOS detection [3-5], IDS and IPS [6-8], moving target defense [9], network monitoring [10-13] and so on. However, as the network is becoming more and more crowded, sampling is required to effectively obtain useful SDN information to feed these applications. OpenNetMon [14] uses an adaptive rate to poll the switch so as to measure the throughput, packet loss and delay. The adaptive policy is based on the rate of new flow generation, which can reduce unnecessary flow samplings when there are stable traffic flows. However, it cannot overcome the controller limitation when there is outburst traffic. OpenSample [15] uses a math model to prove that there exist two dis-

tinct flow samples over the identical flow. Based on the information carried by the two samples, including sequence numbers and time stamps, it can estimate the packet rate and detecting elephant flows. OpenSample is limited in packet rate estimation while security related network monitoring needs more information than that. Other methods such as Sample&Pick, FleXam and Wildcard Samper [16-18] are mainly concerned with heavy flow detection and traffic scheduling, instead of security monitoring. What's more, they need to modify either OpenFlow protocol or the switch hardware, which are complex to implement.

Some prior works have been done in exploring the application of SDN-based network security monitoring, including malware detection, DDoS, and IDS [19-21]. However, these works mainly focus on anomaly detection and signature-based detection, which have significant features of traffic bursting over specific host or specific port, or features matching with existing malicious behaviors signature, such as urls and traffic bytes. Different from elephant flow detection and anomaly detection, P2P control traffic is much stealthier with its decentralized architecture. However, there are still some features which can be used to identify P2P control traffic and P2P botnet, such as multiple transport protocols, package size, in-degree and out-degree.

BotHunter [22] and BotMiner [23] identify zombie hosts based on detectable malicious activity, such as scanning, spam, exploitation and DDoS. However, P2P botnets' malicious activity is more subtle and less likely to be detected. BotGrep [24] and BotTrack [25] use a TDG-based method to detect P2P botnets by analyzing the communication graphs extracted from network flows collected from multiple large networks. Although they do not rely on the zombie host to identify malicious activities, they need to collect from other systems (such as honeypots) infected host information to guide the detection, which is difficult in reality. PeerSorter [26] and PeerDigger [27] rely on cluster flow classification and analysis to detect P2P botnet traffic, which is easy to implement and efficient. However, due to the large traffic of cloud data center, they are inefficient in detecting P2P botnets online in real time.

Our system uses PeerDigger as the DPI engine for P2P bots detecting. We combine it with SDN-based network monitoring approaches, to make it workable online. Besides, we implement an effective two-stage sampling method based on SDN, which can overcome its performance limitation and make it work in real time in cloud environment. Our work can be extended to build a general online cloud network monitoring system, which focuses on security application developments over SDN.

3. ARCHITECTURE AND ALGORITHM

3.1 Sampling Architecture

As the Fig. 1 shows, the architecture of our sampling system mainly consists of three modules: temp flow capture, sampling rate decision and packet sampling.

The temp flow capture module is based on Floodlight controller. By default, it forwards network traffic which is active and accessible, and generates temp wildcard flows for the traffic. As these flows survive only a short period of time, the controller must timely query the switch to get the status of the network.

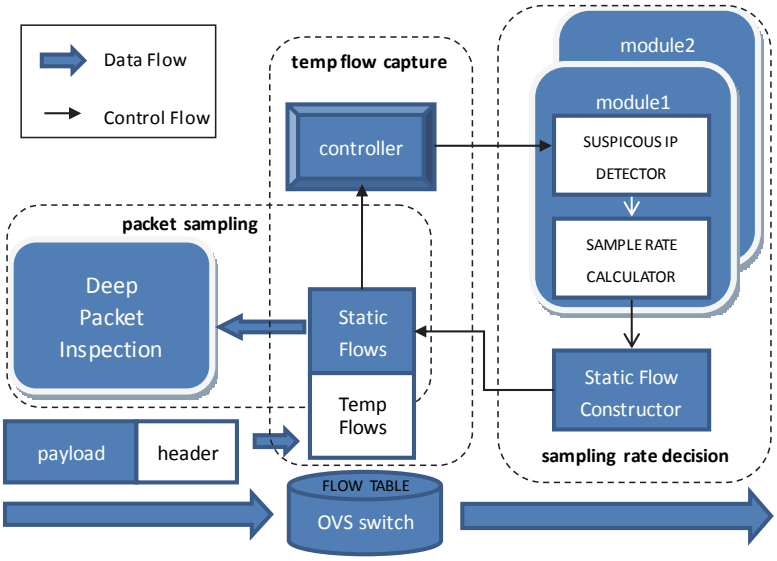


Fig. 1. Architecture of our sampling system.

The sampling rate decision module is made up of two components, namely the suspicious IP detector and sampling rate calculator. The detector detects suspicious IPs based on the flows information, which is five-tuples of <src.ip, dst.ip, src.port, dst.port and proto> in our special purpose to detect P2P hosts. Then the calculator decides how to assign the preset sampling rate to different IPs based on the suspicious degree of the IPs. The more suspicious the IP is, the larger sampling rate will be assigned to the flows related with the IP. The sampling rate decision module is a standalone application, which makes it extensible and can be customized to better detect other kinds of threats, including DDoS, port scanning, and application protocols detection.

Lastly, the packet sampling module is used to capture the full packet whose IP is suspicious by installing static flows on the switch and adaptively send the packets to the Deep Packet Inspection (DPI) module. The static flows are installed by the controller according to the suspicious detection results. A timeout is set to control the survival time of the flows based on their sampling rate calculated in the last stage.

The architecture is simple, but a lot of challenges need to be met in order to make it work. We will first describe the sampling decision module and then discuss the details in solving these challenges. The sampling decision module is customized based on the security needs. For different threats, we might implement different sampling strategies. Here, we propose a two-stage algorithm for P2P detecting, including suspicious IP detecting and adaptive sampling rate decision.

3.2 Sampling Algorithm

To sample as much as suspicious traffic, we utilize our previous work named T-Sampling [29]. T-Sampling mainly consists of two stages, namely suspicious IP detecting and sampling rate allocation.

The inner IP addresses are classified as suspicious or not based on two features, *i.e.* BGP prefixes number and re-connection rate (RCR). BGP prefixes represent the distribution of the communication to the host with that IP address. The larger it is, the more likely the host is a P2P bot. This is due to the fact that P2P bots try to obtain resources from other P2P nodes in order to keep connectivity, while those nodes belong to different subnets. RCR represents the number of connections to the same IP address. P2P bots has large RCR as they need to frequently re-connect other bots to exchange information such as topology and command. We set two thresholds θ_{BGP} and θ_{RCR}, and classify an IP address as suspicious when both BGP and RCR are larger than the thresholds.

T-Sampling Algorithm
Input: Raw Flow Counting Table (RFCT), target sampling rate SR_{tar}
Output: Sampling rate for each inner IP SR^1_{ins}, SR^2_{ins}…, SR^m_{ins}, SR_{non_sus}
begin
delete entries whose count(SYN) > 1;
merge RFCT entries by <srcip, dstip, srcport, dstport, proto>;
for each srcip in RFCT:
calculate BGP and RCR;
if BGP > θ_{BGP} and RCR > θ_{RCR}:
add srcip to suspicious IP set Set_{sus};
calculate packet proportion $p_{sus}, p_1, p_2, \ldots, p_m$
if suspicious IP set is empty:
$SR_{non\_sus} = SR_{tar}$;
return SR_{non_sus};
end
$SR_{sus} = SR_{tar}/p_{sus}$;
if $SR_{sus} > 1$:
$SR_{sus} = SR^1_{ins} = \ldots = SR^m_{ins} = 1$;
$SR_{non\_sus} = (1 - \frac{SR_{sus} \times p_{sus}}{SR_{tar}}) \times \frac{SR_{tar}}{1 - p_{sus}}$;
return $\{SR^1_{ins}, SR^2_{ins}\ldots, SR^m_{ins}, SR_{non\_sus}\}$;
end
else
$SR_{non\_sus} = 0$;
for each IP^i_{sus}
$V_i = \omega_1 \times BGP_i + \omega_2 \times RCR_i$;
$V_{sum} = \sum_{i=1}^{m} V_i$;
for each IP^i_{sus}
$K_i = V_i - \frac{V_{sum} \times p_i}{SR_{sus}}$;
Reorder Sampling Table in inverted order by K_i;
for each $i = 1, 2, \ldots, m$
$SR^i_{ins} = SR_{sus} \times \frac{V_i / V_{sum}}{p_i}$;
if $SR^i_{ins} > 1$:
$SR^i_{ins} = 1$;

```
        for each j = i + 1, …, m
            V_j+ = K_i/(m–i);
    return {SR^1_ins, SR^2_ins…, SR^m_ins, SR_non_sus};
end
```

After we get the suspicious IP addresses, we can allocate the sampling rate to these IP addresses based on the suspicious degree. Firstly, we allocate enough sampling rate to the suspicious traffic, and the remaining sampling rate are equally allocated to other normal IP addresses. Secondly, we further allocate the sampling rate of suspicious traffic to each IP address based on its suspicious degree. The suspicious degree is measured based on the proportion of packets and the value of its BGP and RCR. Further details about T-Sampling can be found at [29], and we will not discuss the details due to the limit of space.

It is worth to note that our sampling system is not limited to detect P2P bots, but can be adjusted according to the requirement of specific task. We can use different sampling algorithm with different tasks. The core principle is that we should first have a brief classification of traffic based on the counters of SDN switches, and then determine the sampling rate of each suspicious flow based on the classification. After that, we can install flows on the switches in order to obtain samplings. The details will further discussed in the following section.

4. IMPLEMENTATION DETAILS

4.1 Raw Flow Counting

We use Openvswitch (OVS) as the SDN switch and Floodlight as the SDN controller [30]. In the raw flow counting phase, we need to use the SYN flag to define a TCP flow. However, Floodlight doesn't support TCP flags matching by the time we write this paper, so we must manually add the matching field to it. OVS 2.1 and above already supports TCP flags matching, but we need to modify the controller to be able to detect and set the new matching field. In floodlight, we need to define the OXM (OpenFlow Extendable Module) for TCP flags. Corresponding to OXM speculation, the Type-Length-Value (TLV) structure for TCP flags is shown in Fig. 2.

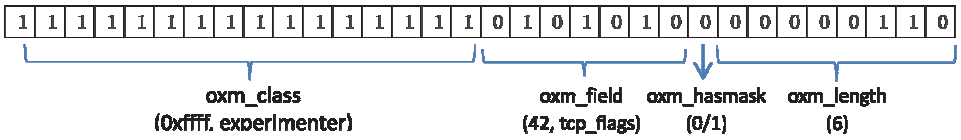


Fig. 2. TLV structure of TCP flags.

The length is 6 bytes, of which 2 bytes are used to represent the type of TCP flags, including FIN, SYN, RST, PSH, ACK, URG, ECE, CWR, NS; the other 4 bytes is the experimenter ID. Even though we only use SYN, other flags can be used to determine

the state of the TCP session, which are useful in other traffic engineering applications. As floodlight uses loxigen as its backend, we need to create a matching module under openflow_output using the TLV header and add corresponding structures in other related source files. As we use the forwarding module to generate temp flows, we also need to modify the forwarding module to match TCP flags. The patching files have been merged into loxigen and floodlight at github.

The statistic time *window* for raw flow counting is set up as 5 minutes to collect sufficient information for suspicious IP detecting. However, this time greatly exceeds the default survive time (5s) of the temp flows, which means we need to query the switches every 5s to get the statistics before the temp flows are removed. We name the survival time of each flow as a *slice*, and add a temp list to store the flows in last *slice*. Each time a flow of the *slice* is added to the counting table of the *window*, it will check if it is counted in the last *slice*, thus avoiding the duplicated flows when we merge the 60 slices into one *window*.

4.2 Packet Sampling

After sampling rate is calculated, implementing the sampling rate to the switches needs careful handling. Although traditional sampling technologies such as Netflow and sFlow provide sampling function, they use uniform sampling rate for all traffics, which is inefficient. Dynamical sampling rate adjustment is provided by sFlow, but it only adjusts the sampling rate based on the overall traffic throughput, rather than based on the importance of each flow.

To implement per-flow sampling rate to SDN switches, we covert the sampling rate to a time concept of OpenFlow protocol. For those monitoring jobs which only need packet header information (such as elephant flow detection), we just install suspicious IP flows and periodically query the flows to get statistics. The query interval is determined by the sampling rate and statistic time window as,

$$interval = window/rate. \tag{3}$$

For those monitoring jobs which require deep packet information (such as P2P bots detection), we can install static flows with suspicious IP and set the flow actions to forwarding packets to controller using packet-in message. However, this would increase the performance cost of the controller and cause the controller unable to provide service. To reduce the cost introduced by frequently packet-in messages, we set the static flows to forward suspicious packets to a separate DPI system. The survival time (hard timeout) of the flows is calculated based on the sampling rate as,

$$timeout = rate * window. \tag{4}$$

As an example, if we calculated the sampling rate to be 0.5 in the second stage, and the statistic time window is 5 minutes, we can get the hard timeout for the forwarding flows to be 2.5 minutes. That means the statistic flows will expire when it's idle more than 5 seconds (idle timeout) or after 2.5 minutes (hard timeout) regardless of its activity. In a short window, it may seem like a continuous sampling of 2.5 minutes. But over the longtime, it is a dynamical changing sampling rate at no fix interval as the idle time is at

random. This is not only efficient in performance, but also resistant to attacks which detect periodic sampling using network delay probing and bypass the sampling by attacking during the interval of samplings.

4.3 Performance Cost Reduction

The performance cost of our sampling system mainly consists of three parts: the storage cost of switches to store the temp flows, the controller overhead of handling flow queries, the communication overhead between SDN controller and switches.

Temp flows (or reactive flows) are flows installed for undefined traffics and only exist for a limited period of time (5s in floodlight). We can increase the survival time by adjusting the idle-timeout parameter. However, this would increase the number of temp flows existed in the switch, and the packet forwarding process would be slower as the flow matching time increases. What's more, temp flows will frequently get updated as new flows are generated and old ones expire. When the flows expire, the switch will send a *FlowRemoved* message to the controller, and next time the traffic comes to the switch, the switch will send a *PacketIn* message to the controller. All these messages would increase the burden of the controller. For these reasons, we should install static flows (or proactive flows) for known service traffic and potential attacking traffic (whitelist and blacklist), and leaving the rest of traffic to be dealt with temp flows.

Our sampling system reduces the temp flow cost by identifying suspicious flows and installing static flows for them, thus avoiding frequently flow generating and expiring. After installing static flows that forward packets to DPI instead of controller, we can reduce the number of temp flows and messages send to controller. However, if we just install permanent static flows for DPI, the sampling would gradually becoming bias and loss the representativeness. This is because infected bots won't continually send malicious packets, but rather keep communication with other bots by sending intermittent packets. Most of the time, the bots are just act normal hosts. We improve this by implementing proper hard-timeout for the installed static flows. The timeout is calculated based on the first stage's analysis of suspicious, the more suspicious an IP is, the hard-timeout is bigger, thus maintain the representativeness of the samples while reduce the cost.

In the first stage of identifying suspicious IP addresses, we need to set up a time window long enough to collect flow counts which can represent the statistics of the traffic during that period. In our practice, we found that 5 minutes is a proper time windows for that. However, this time window is much larger than the default timeout of 5 seconds, which means we need to frequently query the switches to get the statistics. When there are a lot of temp flows, the flow statistic data might cost network bandwidth and increase the resource cost of the controller. We will evaluate this performance cost in Section 4.3.

5. EVALUATION

5.1 Experiment Preparation

The experiment layout is shown in Fig. 3. The Host1 is a Dell PowerEdge R730xd

server, with one Intel Xeon CPU E5-2603 v3 and 6 cores, and installed with Xen 4.6.0 and OVS 2.5.0. The network cards eth1 is BRCM 10GbE 2P 57810S adapter. The physical switch is 10GbE convergence switch. Each virtual machine is assigned with 1 core CPU and 2G memory. VM1 and VM2 are infected by botnet, VM3 is normal. All of the VMs are installed with Ubuntu 1404 64bit system and use tcpreplay for traffic replaying. They separately replay the traffic listed in Table 1 to simulate the real-world cloud traffic. The Floodlight controller and P2P detector are installed on standalone physical machines to better test the performance impact. They are both installed with Ubuntu 1204 32bit, and configured with two cores CPU and 4G memory.

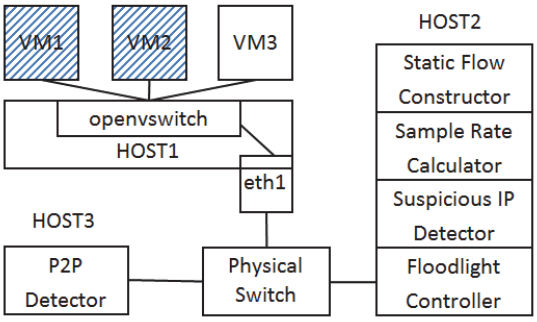


Fig. 3. Experiment layout.

The background dataset came from a span port mirroring a backbone router at the campus network. The hosts connected to the router are freshly installed and well protected. We collected all traffic crossing the router for 12 hours. This dataset contains a large number of general traffic from a variety of applications, including web-browsing, email, online-games, P2P file-sharing systems, P2P-TV platforms, *etc.* Overall, we observed 655 internal IP addresses. We also obtained datasets of two popular P2P botnets, Storm and Waledac, from third parties [31]. The dataset of Storm includes 13 individual bots, while the dataset of Waledac includes 3 individual bots. Table 1 summarizes brief information of the datasets.

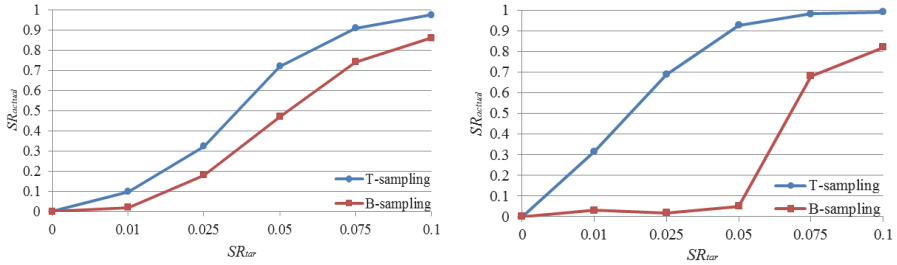
Table 1. Summaries of traffic datasets.

VM	Dataset	Duration	Hosts	Packets
VM1	Storm	12 hour	13	27.18M
VM2	Waledac	12 hour	3	6.51M
VM3	Background	12 hour	655	1812.29M

5.2 Sampling Algorithm Evaluation

We compare our T-Sampling algorithm with B-Sampling [32], another adaptive sampling algorithm to detect botnet. Figs. 4 (a) and (b) present the sampling rate for Storm and Waledac packets when setting different target sampling rate. The results show that T-Sampling has higher sampling rate than B-Sampling for both Storm and Waledac

packets. For example, when the target sampling rate is 0.05, T-Sampling captures 92.6% of Waledac packets while B-Sampling only captures 5% of Waledac packets.



(a) Overall sampling rate for packets in Storm. (b) Overall sampling rate for packets in Waledac.
Fig. 4. Sampling performance comparison between T-Sampling and B-Sampling.

The time costing of T-Sampling and B-Sampling is evaluated and shown in Table 2. From the table we can see that B-Sampling needs one hour to get the first instant sampling rate, while T-Sampling only requires 5 minutes. Moreover, T-Sampling can update the sampling rate more quickly with only one third of the time of B-Sampling. The results prove that T-Sampling can adjust its sampling rate more quickly to adapt to the change of network traffic, thus making it more suitable for working online.

Table 2. Time cost of T-sampling and B-sampling.

	T-Sampling	B-Sampling
Boot time	5 minutes	60 minutes
Time window	5 minutes	15 minutes

After the sampling phase, we would like to verify if the sampling packets contains enough bots traffic for P2P bots detection. Here we use PeerDigger as our detection algorithm [27]. The results are shown in Table 3. As long as the target sampling rate is higher than 0.05, PeerDigger can obtain high TPR and low FPR close to that of all traffic detection. Moreover, the FPR is 0% because the sampled traffic contains little normal traffic.

Table 3. P2P bot detection results under different target sampling rate.

SR_{tar}	0.01	0.025	0.05	0.075	0.1	1
TPR	26.45%	63.83%	93.08%	96.27%	98.76%	99.65%
FPR	0%	0%	0%	0%	0%	1.2%

5.3 System Performance Evaluation

To test the flow storage cost of our sampling system, we send the dataset in 3000 pps (packets per second) and query the OVS to get the flow numbers in every 5 seconds. Then we get the average flow number in each time windows, which is 5 minutes in our

experiment. The result is shown in Fig. 5, from which we can see that after implementing our sampling system, we can reduce the number of the flows to half of that when no sampling is taken. This is due to the fact that our sampling system installs static flows for suspicious flows, which reduces the repeatedly generating and expiring of temp flows.

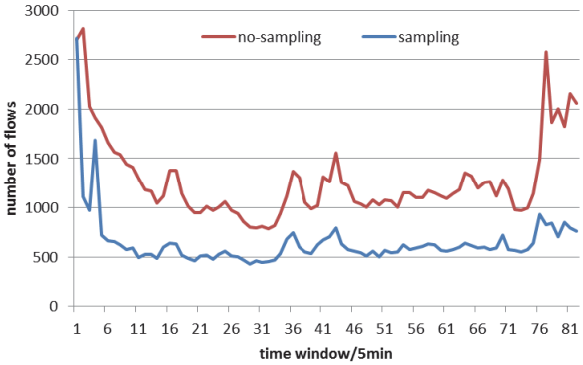


Fig. 5. Number of flows in OVS.

We use iperf [33] to test the impact of our sampling method on the overall throughput of SDN network. We test it every one minute during the traffic replaying. The bandwidths under different conditions are shown in Fig. 6. We can see that when the OVS is connected to SDN controller, the bandwidth gradually decreases as the time in the first five minutes. This is due to the number of flows in the switch increases as the time grow, which increases the matching time for traffic. After the five minutes of the first sampling window, the flows generation and expiring come to a relative balance. So the bandwidth is relatively stable after the first five minutes.

When the SDN controller is not connected, the traffic just directly goes through the switch. When SDN is enabled, the switch must first check if the traffic match any flow entries in the flow table, thus decreases the throughput. The bandwidth degradation caused by SDN is 12% in average, mainly caused by flow matching of OVS switches.

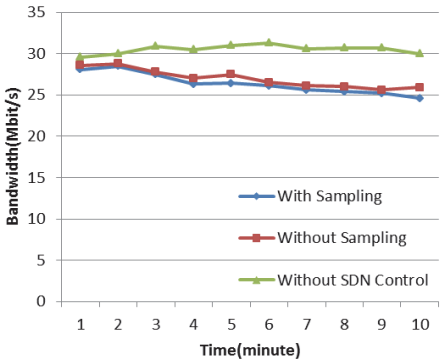


Fig. 6. Bandwidth comparison using iperf.

However, the sampling function actually has little impact on the overall throughput. Compared with SDN network when sampling is disabled, the overall throughput just decreases 2% in average. This is due to the fact that sampling process queries the switches to get the flow statistics in every 5 seconds. As the flow statistics data is small, its impact on bandwidth is also small. Overall, the performance cost caused by SDN and sampling is about 14%, which is acceptable and can be further improved using hardware switches.

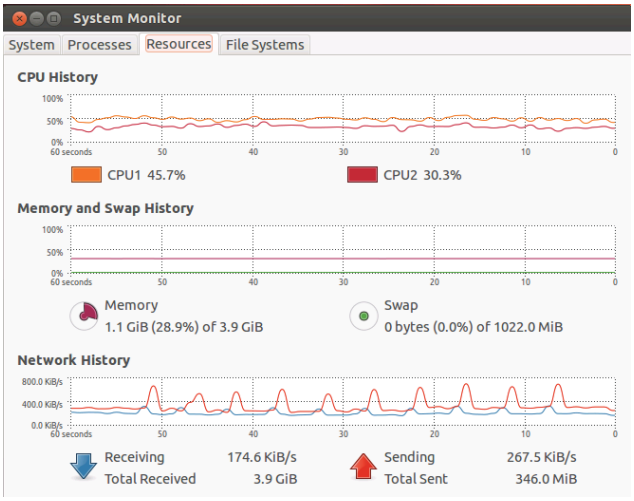


Fig. 7. Performance of controller.

Fig. 7 presents our sampling system's influence to the controller. We start our sampling at 50s, and stop it at 5s. Each peak at network indicates one query of flow counts. Overall, the additional network cost is 400KB/s at peak and below 100KB/s in average. The CPU and memory cost introduced by sampling is small and can be ignored.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a traffic sampling system based on SDN to detect security threats in cloud computing environment. We implemented a prototype to detect P2P bots, which can effectively reduce the traffic that P2P botnet detectors need to process while keeping their detection accuracy, thus allowing them to operate on high-speed and high-volume networks. Our sampling system mainly includes a suspicious IP identification stage and a sampling rate decision stage. By exploiting the inherent nature of P2P botnet, the algorithm identifies a small number of suspicious IP addresses that are likely P2P bots as soon as possible. Then it dynamically tunes the instant sampling rate of every internal IP address to capture as many botnet-related packets as possible while keeping the actual sampling rate close to the target sampling rate. The experimental results show that our system achieves good performance. It is able to identify potential P2P bots in 5 minutes with TPR of 99% and FPR of 6.9% and capture much more botnet-related

packets than the B-sampling techniques. By carefully integrate the sampling algorithm to open-source SDN software, including statistic flow installing, dynamically timeout setting and forwarding suspicious traffic to DPI, the overall performance cost of our system is acceptable and can be further reduced using hardware SDN switches. This means our system can work on-line and response timely when threats are detected. As a modular design, our system can be easily extended to detect other cloud security threats such as DDoS, which will be our future work.

REFERENCES

1. C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, Vol. 21, 2003, pp. 270-313.
2. H. Hu, W. Han, G. J. Ahn, and Z. Zhao, "FlowGuard: Building robust firewalls for software-defined networks," in *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 97-102.
3. R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proceedings of IEEE 35th Conference on Local Computer Networks*, 2010, pp. 408-415.
4. H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 239-250.
5. B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, Vol. 81, 2015, pp. 308-319.
6. T. Xing, D. Huang, L. Xu, C. J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-based intrusion prevention system in cloud environment," in *Proceedings of Research and Educational Experiment Workshop*, 2013, pp. 89-92.
7. A. Le, P. Dinh, H. Le, and N. C. Tran, "Flexible network-based intrusion detection and prevention system on software-defined networks," in *Proceedings of International Conference on Advanced Computing and Applications*, 2016, pp. 106-111.
8. T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Computer Networks*, 2016, pp. 1-11.
9. J. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 127-132.
10. A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *Proceedings of IEEE Network Operations and Management Symposium*, 2014, pp. 1-9.
11. S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks," in *Proceedings of IEEE Network Operations and Management Symposium*, 2014, pp. 1-9.
12. S. Seungwon and G. Guofei, "CloudWatcher: Network security monitoring using

OpenFlow in dynamic cloud networks," in *Proceedings of the 20th IEEE International Conference on Network Protocols*, 2012, pp. 1-6.
13. J. Ko and A. V. Vasilakos, "Software defined monitoring of application protocols," *IEEE Transactions on Computers*, Vol. 65, 2016, pp. 615-626.
14. N. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *Proceedings of IEEE Network Operations and Management Symposium*, 2014, pp. 1-8.
15. J. Suh, T. T. Kwon, and C. Dixon, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proceedings of IEEE 34th International Conference on Distributed Computing Systems*, 2014, pp. 228-237.
16. Y. Afek, A. Bremler, and S. L. Feibish, "Sampling and large flow detection in SDN," *ACM SIGCOMM Computer Communication Review*, Vol 45, 2015, pp. 345-346.
17. S. Shirali-Shahreza and Y. Ganjali, "FleXam: flexible sampling extension for monitoring and security applications in openflow," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 167-168.
18. P. Wette and H. Karl, "Which flows are hiding behind my wildcard rule? adding packet sampling to OpenFlow," *ACM SIGCOMM Computer Communication Review*, Vol. 43, 2013, pp. 541-542.
19. Z. Abaid, M. Rezvani, and S. Jha, "MalwareMonitor: An SDN-based framework for securing large networks," in *Proceedings of CoNEXT on Student Workshop*, 2014, pp. 40-42.
20. S. Lim, J. Ha, and H. Kim, "An SDN-oriented DDoS blocking scheme for botnet-based attacks," in *Proceedings of the 6th International Conference on Ubiquitous and Future Networks*, 2014, pp. 63-68.
21. S. Seeber and G. D. Rodosek, "Towards an adaptive and effective IDS using openflow," in *Proceedings of IFIP International Conference on Autonomous Infrastructure*, *Management and Security*, 2015, pp. 134-139.
22. G. Gu, P. A. Porras, and V. Yegneswaran, "Bothunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of Usenix Security Symposium*, 2007, pp. 1-16.
23. G. Gu, R. Perdisci, and J. Zhang, "BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *Proceedings of USENIX Security Symposium*, 2008, pp. 139-154.
24. S. Nagaraja, P. Mittal, and C. Y. Hong, "BotGrep: Finding P2P bots with structured graph analysis," in *Proceedings of USENIX Security Symposium*, 2010, pp. 95-110.
25. J. François, S. Wang, and T. Engel, "BotTrack: tracking botnets using NetFlow and PageRank," in *Proceedings of International Conference on Research in Networking*, 2011, pp. 1-14.
26. J. He, Y. Yang, and X. Wang, "Peersorter: classifying generic p2p traffic in real-ime," in *Proceedings of IEEE 17th International Conference on Computational Science and Engineering*, 2014, pp. 605-613.
27. J. He, Y. Yang, and X. Wang, "PeerDigger: Digging stealthy P2P hosts through traffic analysis in real-time," in *Proceedings of IEEE 17th International Conference on Computational Science and Engineering*, 2014, pp. 1528-1535.

28. A. Ramachandran, S. Seetharaman, and N. Feamster, "Fast monitoring of traffic subpopulations," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, 2008, pp. 257-270.
29. J. He, "Study on P2P botnet real-time detection based on network traffic," Dissertation, School of Computer, National University of Defense Technology, 2016.
30. "Floodlight SDN controller," https://github.com/floodlight/floodlight.
31. B. Rahbarinia, R. Perdisci, and A. Lanzi, "Peerrush: Mining for unwanted p2p traffic," *Journal of Information Security and Applications*, Vol. 19, 2014, pp. 194-208.
32. J. Zhang, X. Luo, and R. Perdisci, "Boosting the scalability of botnet detection using adaptive traffic sampling," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011, pp. 123-134.
33. "IPerf – The network bandwidth measurement tool," https://iperf.fr/.

Jiang-Yong Shi (施江勇) is currently pursuing the Ph.D. degree in the School of Computer, National University of Defense Technology. His research interests include virtualization and SDN-based security.

Jie He (何傑) received his Ph.D. degree in Computer Science from National University of Defense Technology in 2015. He is currently working in Department of Information Engineering, Officers College of PAP. His research interests include traffic classification, network security, and information security.

Yue-Xiang Yang (杨岳湘) is currently a Professor at National University of Defense Technology. His research interests include network security and web service.