View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Crossref

Hindawi
Security and Communication Networks
Volume 2018, Article ID 4723862, 13 pages
https://doi.org/10.1155/2018/4723862

Research Article

Detecting P2P Botnet in Software Defined Networks

Shang-Chiuan Su, Yi-Ren Chen, Shi-Chun Tsai, and Yi-Bing Lin

Department of Computer Science, National Chiao Tung University, Hsinchu 30050, Taiwan

Correspondence should be addressed to Shi-Chun Tsai; sctsai@cs.nctu.edu.tw

Received 21 January 2017; Revised 24 July 2017; Accepted 20 August 2017; Published 29 January 2018

Academic Editor: Jesús Díaz-Verdejo

Copyright © 2018 Shang-Chiuan Su et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Software Defined Network separates the control plane from network equipment and has great advantage in network management as compared with traditional approaches. With this paradigm, the security issues persist to exist and could become even worse because of the flexibility on handling the packets. In this paper we propose an effective framework by integrating SDN and machine learning to detect and categorize P2P network traffics. This work provides experimental evidence showing that our approach can automatically analyze network traffic and flexibly change flow entries in OpenFlow switches through the SDN controller. This can effectively help the network administrators manage related security problems.

1. Introduction

There are many peer-to-peer (P2P) network traffics in the Internet. Through P2P, cyber threats caused by botnets have significantly increased in recent years. Attackers can use botnet to construct various malicious activities. In order to enforce appropriate network management and security policies, we need to detect botnet while they are communicating instead of when the attacks have already happened. As a result, it is an important and difficult task for network administrators to identify and categorize P2P traffic types.

A botnet consists of a collection of compromised computers controlled by a botmaster, which issues instructions to the infected computers through the command-and-control (C&C) server. The C&C channel of the botnet can be any communication protocol such as Internet Relay Chat (IRC), Hyper Text Transfer Protocol (HTTP), or P2P network. Botnets represent a collaborative and highly distributed platform that conduct a wide range of malicious and illegal activities, such as launching Distributed Denial of Service (DDoS) attacks, sending SPAM e-mails and click fraud, and collecting confidential information. In order to mitigate security threat posed by botnets, many detection methods have been proposed in the literature over the last decade [1–6]. These detection methods are based on numerous technical principles and assumptions that the botnets produce their own behaviors and the patterns of network traffic. One of the most prominent botnet detection methods is based on identifying network traffic produced by botnets using machine learning techniques [5–9].

Hu et al. [10] studied characteristic of fast-flux botnet to find features and used multilevel Support Vector Machine (SVM) to detect fast-flux botnet. Bilge et al. [11] analyzed numerous features extracted from NetFlow and built different detecting models based on various machine learning algorithms (i.e., J48 decision tree, SVM, and Random Forest) to detect C&C server. Saad et al. [9] and Stevanovic and Pedersen [3] experimented various machine learning algorithms and compared the detection performance. The main assumption of the machine learning-based methods is that botnets create distinguishable patterns within the network traffic, which can be efficiently detected and analyzed by machine learning algorithms [7]. These methods propose a flexible detection that does not require traffic payload to exhibit any anomalous characteristics or much prior knowledge of botnet traffic patterns.

Software Defined Network (SDN) [12–14] separates the control plane and the data plane. Usually, there is one SDN controller in the control plane, and the data plane consists of network devices. The network devices use specific protocols such as OpenFlow [15] to communicate with the controller via the control plane. They just handle packets according to

the flow tables managed by the controller rather than process packets by themselves. In traditional network environment, network administrators have to manage each network device one by one. In contrast, SDN devices can be managed automatically by programming modules for the SDN controller. Developers can implement network functions by programming their modules and embed them into the SDN controller to control the packet flows in the data plane. However, traditional security issues still exist and would be even worse in SDN [13, 16] if not handled properly.

Zaalouk et al. [17] and Giotis et al. [18] detected malicious activities by analyzing sFlow records, where sFlow is an industry standard technology for monitoring networks [19]. They showed how to mitigate damage caused by the malicious activities with SDN functionality. Some authors focused on detecting DDoS attack in SDN [20]. However, they did not automatically manage through SDN functionality. Moreover, recent versions of OpenFlow protocol can only handle packet headers up to the OSI transport layer. If we want to manage network traffic generated from some specific applications or P2P botnets, an additional agent needs to be developed to analyze network traffic.

To address the above issues, based on a work by Su [21], we build a system for P2P botnet traffic detection and application categorization. Once we detect traffic generated by P2P botnet, we can eliminate them with our SDN-enabled solution. By developing modules of SDN controller, we can automatically update flow tables in the network devices in accordance with the analyzed results and then drop packets with botnet traffic patterns. We can also modify the destination fields in the packet header to redirect suspicious traffic to a specific environment (e.g., honeypot) for further analysis. Our solution can detect P2P botnet traffic efficiently and guard network automatically.

The rest of the paper is organized as follows. Backgrounds of botnet and SDN are introduced in Section 2. In Section 3 we propose a P2P botnet detection solution with SDN. In Section 4 we present experiments and evaluation of our solution. Related works are compared with our solution in Section 5. Finally, we conclude with Section 6.

2. Background

2.1. Botnet. The botmaster of a botnet may control infected systems through one or several C&C servers. Over the past decade, there are numerous Internet security incidents caused by botnet. Attackers can use botnet to launch various malicious activities. Some botmasters also use these compromised machines to do distributed computing, such as data mining. Modern botnets usually mimic network traffic generated by normal applications to evade detection from network security agents. For this reason, detecting a botnet has become an important research issue. Botnet life-cycle has been defined by several authors, such as Leonard et al. [22] and Silva et al. [2], which consists of three stages: infection stage, C&C communication stage, and attack stage.

In the Infection stage, botmasters infect other computers through fishing or social engineering and so on. Each such compromised device is called a "bot." Victims usually are unaware of downloading bot code or malware from a binary server and become a part of botnet army when they are opening an email or browsing some web sites. In the communication stage, botmasters usually use a Command and Control (C&C) server to update malware code and propagate commands to bots. Bots also connect to the C&C server periodically to report their statuses. In the attack stage, bots controlled by the botmaster launch diverse malicious activities according to received commands or search for other victim computers.

IRC and HTTP/HTTPS are two popular communication protocols in Internet. Due to their popularity and convenience of deployment and management, botmasters have widely used IRC-based and HTTP-based C&C for deploying botnets. This type of botnets have centralized C&C network architecture, and all bots connect to one or few C&C servers. The main weakness of centralized systems is that they are vulnerable to single point failure. The centralized botnet can be identified easily and disabled because of huge amount of connection between bots and the C&C server. Once the C&C servers have been discovered and disabled, the entire IRC-based or HTTP-based botnet could be taken down.

In order to prevent single point of failure, many botmasters deploy their botnet architecture with peer-to-peer (P2P) communication protocols, such as Kademlia, Bittorent, and Overnet. These botnets, called decentralized botnets, can use any of the bots or P2P nodes to issue commands to other peers or gain useful information. Decentralized botnet offers higher resiliency than centralized botnet, since every bot or P2P node could play the role as a client or the server. Even though some P2P botnets are taken down, the remaining bots could still communicate with the botmaster and other nodes to launch malicious activities. To detect cyber threat from botnet, many solutions have been proposed. These methods can be generally classified as host-based or network-based. A host-based method deploys botnet detection at end point computers, identifying unusual usage of computers, such as CPU utilization, sensitive registers, and memory block. Thus host-based detection approach is not affected by the encrypted communication channel used by botnet [4]. However, the main drawback of host-based detection is that it requires monitoring resource usage of every end host. On the other hand, network-based detection approach inspects network connection behavior and identifies possible network traffic patterns in any period of botnet life-cycle. A network-based method assumes that botnet generates distinguishable network traffic patterns. There are also some similar connections and group activities within the botnet. For example, they would connect to the C&C server, launch DDoS attacks, or spread spam mails at the same time.

Network-based detection approaches can be further classified as signature-based and flow-based methods. Signature-based method analyzes network traffic based on packet level and signatures of malicious payload through deep packet inspection (DPI). Therefore, it has higher accuracy for known attacks. However signature-based method could only analyze attacks or botnets already known. Network administrators are responsible for updating signature database frequently to ensure the safety from the latest detected malware or botnet.

Moreover, botmasters may evade detection of signature-based method through encrypted or compressed payload [2]. Flow-based method, on the other hand, detects botnet by analyzing connection behavior of network traffic flow. A flow is usually defined as the packets with the same source and destination within a specific time period. This detection method identifies suspicious botnet connection traffic patterns by analyzing features extracted from network flow such as flow size, duration, and mean packet size. The flow-based detection does not require inspecting every individual packet payload but analyzing information from the packet header. Therefore, flow-based detection is more efficient because it is not affected by encrypted payload. Moreover, detecting botnet through inspecting network traffic patterns could detect not only a specific botnet but also a botnet family with similar connection behavior. Different botnets in the same botnet family may have different signatures but similar traffic patterns or the same malicious activities.

With experimental evidence, we show how to detect P2P botnet with various functionalities of SDN and machine learning algorithms, which highlights our contribution of this paper.

2.2. Software Defined Network. Since the control and the data planes are separated in SDN, so the network devices do not need to learn network forwarding rules by themselves. They forward or drop the packets according to the rules given by their controller. One popular implementation of SDN southbound protocols is OpenFlow [23], which regulates the communication between the controller and switches. Figure 1 illustrates a simplified OpenFlow switch specification. A flow table consists of flow entries (Figure 1(d)), whose main components include

(i) *match fields*: to match against packets;

(ii) *priority*: to indicate matching precedence of the flow entry;

(iii) *counters*: to be updated when packets are matched;

(iv) *instructions*: to modify the action set or pipeline processing;

(v) *timeouts*: to set maximum amount of time or idle time before flow is expired by the switch;

(vi) *cookie*: to be used by the controller to filter flow statistics, flow modification, and flow deletion.

Most OpenFlow controllers can load programmable modules (e.g., Ryu applets and OpenDaylight Karaf features) to achieve software-defined networking. Some of the modules are designed to plan the packet paths between the end hosts (Figure 1(c)), and each path consists of forwarding rules. The modules use the Application Programming Interface (API) provided by the controller to modify the flow tables of OpenFlow switches by translating their forwarding rules. Thus, these OpenFlow controllers provide a framework for their loadable modules to manage the flow entries of OpenFlow switches.

It is not convenient for network administrators to program and manage network devices in traditional network environment, whereas network service and functionality can be both achieved easily by using OpenFlow in SDN. In OpenFlow 1.3 [23], there are almost 40 match field keys that we can use to compose our forwarding rules. However, under the current OpenFlow protocol, the switches can only process the packet header from layer 1 to layer 4 of OSI model. In other words, they cannot handle the content of the higher layers in the packets, such as application layer. If we want to manage network traffic in accordance with the application layer, the programmable module must do extra works such as commanding the OpenFlow switches to notify their controller of the incoming raw packet by OpenFlow "packet-in" message. Nevertheless, on a busy network, it consumes much time for OpenFlow switches and OpenFlow controller to handle OpenFlow "packet-in" messages. In practice, developers should not process the content of packets higher than layer 4 of OSI model to prevent such extra works.

3. SDN-Enabled P2P Botnet Detection

The network architecture of our solution is shown in Figure 2, where the dashed lines from the OpenFlow controller (named Rule Arbitrator (Figure 2(a))) to the OpenFlow switches (called Data-link Bridges (Figure 2(b))) indicate the management connection. We implement the programmable module whose functionality arbitrates the flow rules of the Data-link Bridges, so the OpenFlow controller acts as a Rule Arbitrator after it loads the module. The Data-link Bridges are OpenFlow switch, whose forwarding behavior is mostly like the traditional layer 2 switch if Rule Arbitrator does not add any flow rules. Firstly, the Rule Arbitrator commands the Data-link Bridges to duplicate all incoming packets to their neighbor Detection Agent(s) (Figure 2(c)). The captured packets with the same 5-tuple (i.e., source IP address, source port number, destination IP address, destination port number, and protocol) information and packets with reverse direction will be recognized as the same flow if they occur closely within a short time period. The flow recognition is the process that the Detection Agent gathers packets into distinguishable flows. After we have gathered flow-level information, we can extract several features from these flows to study the behaviors that occur in the network. The Detection Agent analyzes and categorizes each flow and then labels the flow as P2P botnet or benign P2P application through machine learning models, which are built from the traffic records generated by different P2P botnet or known P2P applications.

After a flow has been classified, the Detection Agent reports the result to the Rule Arbitrator with the 5-tuple information and the type of P2P botnet or application. The Rule Arbitrator, afterwards, modifies the related flow tables in the Data-link Bridges in accordance with the result reported by the Detection Agent. Finally, the Data-link Bridges automatically drop the malicious packets that are recognized by the classifiers.

3.1. Traffic Flow and Feature Extractor Module. In order to get some useful information to classify network traffic between different hosts, this module (Figure 3(a)) aggregates packets into network traffic flows with the same 5-tuple information

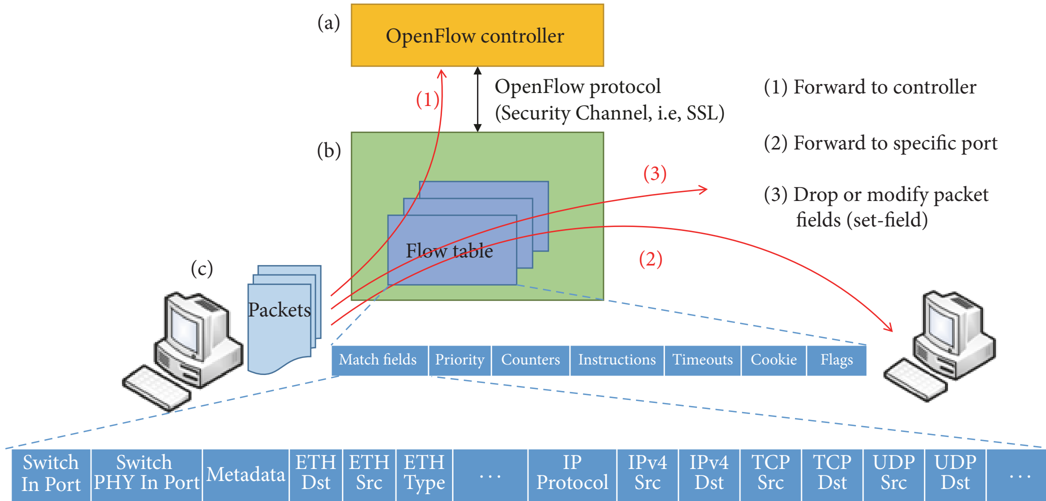


Figure 1: OpenFlow switch specification.

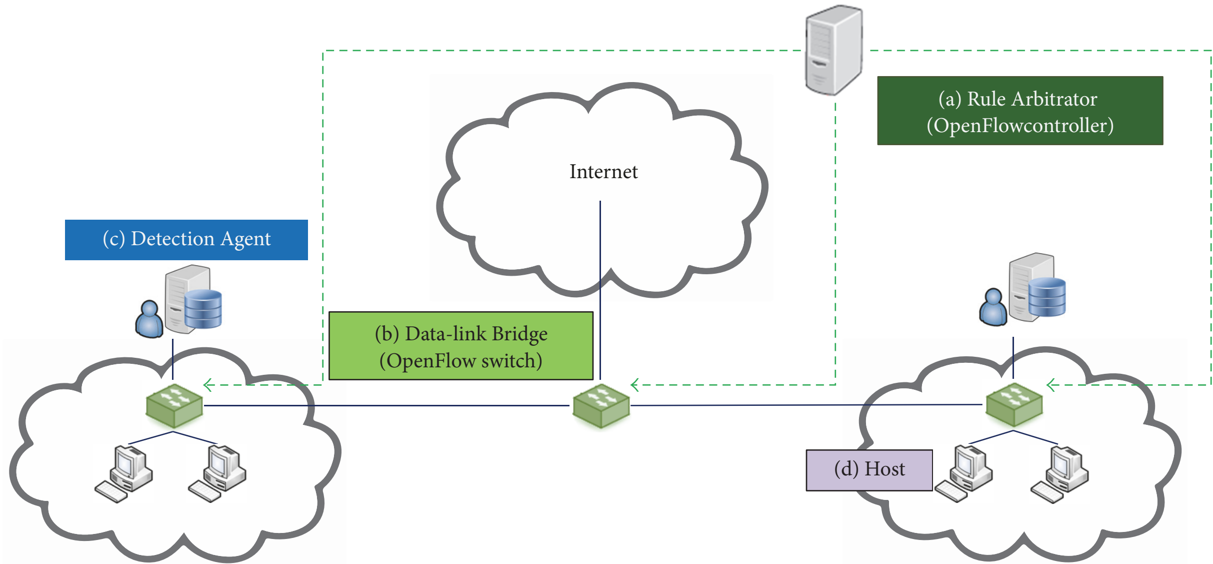


Figure 2: System overview.

after they are mirrored to the Detection Agent. We use *Netmate*, an open source tool [24], to capture packets and transform them into traffic flow, from which we extract feature vectors for machine learning analysis. This tool has been frequently used to capture network packets [6, 25]. We need to define the time frame to capture packets (i.e., a flow duration). In Section 4, we test different flow durations and analyze the performances, which are shown in Table 1. Based on the result, we find that the flow duration with 600 seconds has the best performance.

3.2. P2P Network Traffic Detection Module. This module (Figure 3(b)) has the core idea of classification in our design. After obtaining feature vectors, the module classifies them with machine learning algorithm. This process contains the following four parts.

Table 1: Evaluation on different flow durations.

Duration	Accuracy
15 seconds	81.58%
300 seconds	95.45%
600 seconds	99.88%
3600 seconds	98.34%
36000 seconds	97.70%

3.2.1. Building Detection Model and Training Set. To classify different P2P network traffics with machine learning algorithm, we need to prepare datasets and choose appropriate algorithms for training. We adopt the approach used in *PeerRush* [5] and collect network traffic samples generated by different P2P botnets and normal P2P applications from [5].

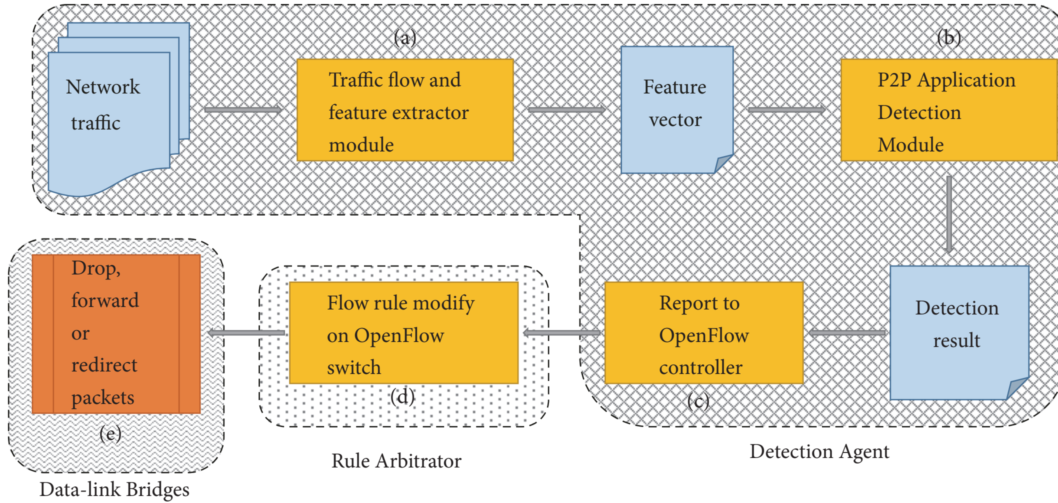


Figure 3: Flow diagram of detecting procedures: procedures (a), (b), and (c) are performed in Detection Agent; procedure (d) is executed by the Rule Arbitrator; the action of (e) is done in the Data-link Bridges.

Table 2: P2P traffic dataset used.

Class	Protocol	Hosts	Days Duration	Transport
Storm	—	13	7	UDP
Zeus	—	1	34	UDP
eMule	eDonkey	2	9	TCP/UDP
uTorrent	BitTorrent	2	9	TCP/UDP
Skype	Skype	9	83	TCP/UDP

To train our classification models to find out network traffic of P2P botnet, we use network traffic trace files of Storm and Zeus Botnet as malicious training samples. Network traffic trace files of eMule, uTorrent, and Skype are used for network traffic samples of benign P2P applications. The datasets we used for training models are summarized in Table 2

There are two submodules in this module, that is, Primary Classification Module and Secondary Classification Module. For each trace file we build a binary classifier for Detection Module (Figure 3(b)), with which we can test feature vectors of network flows and report the analysis result. We build 5 binary classifiers and put them into our Primary Classification Module. Additionally, we use the whole dataset with all kinds of network traffic traces to build a multiclass classifier as the Secondary Classification Module.

The reason why we use multiple binary classifiers to build our Primary Classification Module is to have more flexibility on training and testing, and our detection method can be modularized. With multiple binary classifiers, we can use different training sets, different algorithms, and different parameters for training models. Then we can find an optimal combination to obtain the best classification performance. While with a multiclass classifier, we can only train a model with the same machine learning algorithm, the same training set, and less flexibility. Moreover, when we test an instance with a multiclass classifier, we will certainly classify it into one class of the training set, and that might cause a misclassification. On the other hand, if we use multiple binary classifiers to classify a test instance, we could classify it into *"Unknown"* if none of the classifiers matches. We build the Secondary Classification Module with a multiclass classifier, which is used to classify a test instance when more than one binary classifier match in the Primary Classification Module. The Secondary Classification Module can exactly classify the ambiguous test instance into one class.

3.2.2. Feature Selection. It is crucial in machine learning to choose an appropriate feature set, which affects significantly the classification performance. We adopt some features for P2P application traffic categorization and botnet detection used in [5, 8, 9, 26]. We list our feature set as follows:

(i) Packet count
(ii) Packet size (min, max, mean and standard deviation)
(iii) Total volume (flow size)
(iv) Interarrival times (min, max, mean, and standard deviation)
(v) TCP Push flag count
(vi) Duration of the flow
(vii) Total bytes used for headers
(viii) TCP Urgent flag count.

We compare the detection performance of our feature set with the works by Rahbarinia et al. [5] and Narang et al. [26]. We evaluate the result of a classifier with the ROC curve (i.e., Receiver Operating Characteristic curve) and the P/R curve (i.e., Precision-Recall curve). True positives (TP) are instances correctly labeled as positives. False positives (FP) are negative instances incorrectly labeled as positive. True negatives (TN) correspond to negative instances correctly labeled as negative. False negatives (FN) are positive instances

Table 3: Model comparison for our work and others.

	AUC of ROC	AUC of P/R curve
Narang et al. [26]	0.982	0.960
Rahbarinia et al. [5]	0.951	0.899
Our result	0.982	0.957

Table 4: Performance of different K in K Nearest Neighbors.

K	Accuracy	AUC of ROC	AUC of P/R curve	Avg. time cost in sec
1	91.13%	0.873	0.805	0.026
2	91.19%	0.916	0.842	0.019
3	91.56%	0.931	0.855	0.019
4	91.43%	0.941	0.859	0.018
5	91.43%	0.939	0.856	0.018
6	91.13%	0.948	0.866	0.019
7	91.66%	0.952	0.876	0.019
8	91.23%	0.950	0.867	0.019
9	91.50%	0.951	0.862	0.019
10	91.16%	0.952	0.868	0.019

incorrectly labeled as negative. Then define *True Positive Rate* = TP/(TP + FN); *False Positive Rate* = FP/(FP + TN); *Recall* = TP/(TP + FN); and *Precision* = TP/(TP + FP). The ROC curve is a plot of True Positive Rate on the y axis against False Positive Rate on the x-axis. The P/R curve is a plot of Precision on the y-axis against Recall on the x-axis. The experiment results show that our feature set has close or slightly better performance both in ROC curve and in P/R curve as shown in Table 3, since our areas under the curves (AUC) are close to 1.0 but with fewer features.

3.2.3. Classifier. We experimented for training the classifier with different machine learning algorithms in Primary Classification Module. We use *Sklearn* [27], a package written in *Python*, to conduct the experiments. We have tested the performance of K Nearest Neighbor Algorithm (KNN) and Support Vector Machine (SVM).

The training phase of KNN stores the feature vectors and class labels of the training samples. Let K be a user-defined constant. In the classification phase, an unlabeled vector (a test instance) is classified by assigning the label which is most frequent among the K nearest training samples around this vector [28]. We experiment the performance for $K = 1$ to 10 and the result is summarized in Table 4. We see that the best result for KNN in our tests is with $K = 7$.

The Support Vector Machine Algorithm constructs a hyperplane with maximum margin, which can make the best separation of training data between different classes. At the testing stage, the SVM classifies a test feature vector based on the side of the hyperplane that the vector locates. We analyze the performances of KNN and SVM with 10-Fold Cross Validation for verification and the results are shown in Table 5.

Table 5: SVM versus KNN with 30111 instances and 10-Fold Cross Validation.

	SVM	KNN
Avg. training time	225.845 seconds	0.371 seconds
Avg. accuracy	97.55%	94.14%
Avg. AUC of ROC	0.997	0.967

Table 6: Training P2P traffic dataset summary.

Traffic class	Flows
Storm	60000
Zeus	44786
eMule	60300
uTorrent	62800
Skype	57750
Total	285636

Table 7: Evaluation of KNN binary classifier.

Traffic class	ROC AUC	P/R AUC
Storm	0.987	0.982
Zeus	0.987	0.962
eMule	0.971	0.959
uTorrent	0.985	0.977
Skype	0.980	0.958

Considering efficiency, we find that the detection performances of these two algorithms are similar, but the training time of SVM is much longer. As a result, we use KNN as our classification algorithm for the binary classifier in Primary Classification Module. However, we can always replace it with another method when necessary.

While training the models, we use roughly the same amount of traffic flow samples for each class in order to balance the training sample of all application classes. Table 6 summarizes the training set which we used to train each classifier in Primary Classification Module. We also use the 10-Fold Cross Validation to evaluate the performance of each binary classifier as shown in Table 7.

For the Secondary Classification Module, we adopt the Random Forest Algorithm, which has very good classification performance. The most important issue in Secondary Classification Module is that we need to accurately separate the ambiguous instances, so the Random Forest Algorithm is an appropriate choice. The Random Forest Algorithm is a classification algorithm with several decision trees. A decision tree is a flowchart-like tree, where each internal node denotes a test on an attribute, each outgoing link indicates an outcome of the test and each leave node has a class label. The topmost node of a tree is the root node. The basic concept of Random Forest Algorithm is that it needs to randomly sample some subsets from the whole training set to build decision trees. After building the decision trees, we can have the classification result obtained from the majority of the decision results.

Table 8: Random forest training summary.

	Random forest
Flows (feature vectors) used for training	281786 flows
Avg. training time	1850.025 seconds
Avg. accuracy	99.77%
Avg. AUC of ROC	0.999
Avg. AUC of P/R curve	0.999

We examine the performance of Random Forest Algorithm and evaluate it with the 10-Fold Cross Validation. Table 8 shows the result of our experiment.

3.2.4. Traffic Analysis. For analyzing the network traffic, we mirror each packet passing through the Data-link Bridge to the port that the corresponding Detection Agent attaches. Note that we design a mechanism that allows a Detection Agent to communicate with the SDN controller and send a "registering" packet first when the Detection Agent starts to run. The registering packet is secret, so the hosts could not pretend to be the Detection Agent. When the Rule Arbitrator receives the registering packet sent from the Detection Agent, it gathers the following information: the identifier of the Data-link Bridge to which the Detection Agent is connected, the port on which the Detection Agent is attached, and the MAC and IP address of the Detection Agent.

We show the sequence diagram in Figure 4. First, the Rule Arbitrator (Figure 4(a)) initializes the flow tables (Figure 4(1)) in the Data-link Bridges (Figure 4(b)) to trap the registering packet that may be sent later from the Detection Agent (Figure 4(c)). The entries of the initial flow tables are secret to prevent malicious hosts (Figure 4(b)) from pretending to be the Detection Agent. When the Detection Agent (Figure 4(c)) starts to work, it sends the registering packet to its neighbor Data-link Bridge (Figure 4(2)), which then notifies the Rule Arbitrator of the registering packet wrapped in the OpenFlow *packet-in* message. In other words, we implement the *packet-in* event handler in the Rule Arbitrator's module to gather the information of the *packet-in* message, which contains the ingress port number, the identifier of the OpenFlow bridge, and so on. Hence, the Rule Arbitrator, then, can add a flow entry to command the Data-link Bridge to duplicate incoming packets from the other ingress ports to the port which the Detection Agent is attached on. At the same time, the Detection Agent monitors its network interface and captures received packets (Figure 4(6)). We use *Netmate* in the Detection Agent to capture received packets and recognize them as distinguishable traffic flows in accordance with the 5-tuple information of packet header, and the Detection Agent extracts features from the flows and transforms them into feature vectors for analysis. After that, if it is an alarming result, the Detection Agent will send the result to the Rule Arbitrator via Restful API (Figure 4(7)), and then the Rule Arbitrator modifies the flow tables of the corresponding Data-link Bridge to drop matched malicious packets (Figure 4(8)).

Figure 5 shows the flow diagram of testing phase of the P2P Application Detection Module running in a Detection Agent. After we extract the feature vector from each network flow, we obtain the classification result by testing it with all binary classifiers in Primary Classification Module. Each binary classifier will have a testing result, indicating yes or no (true or false). If there is only one or no classifier that has the testing result indicating yes, then we label this testing instance accordingly or classify it as *Unknown*, respectively. If there are at least two classifiers that label this testing instance positively, then do a further test with the Secondary Classification Module to classify this ambiguous testing instance. Finally, we get an analysis report that contains the 5-tuple information (e.g., srcip = "1.1.1.1", srcport = 15931, dstip = "2.2.2.2", dstport = 80, and proto = 6) and the classification result (e.g., Zeus Botnet).

3.3. Report to the Rule Arbitrator. If the classification result of the Detection Agent matches any class of the P2P botnets or benign applications which we are interested in, the Detection Agent will notify the Rule Arbitrator to adjust the flow entries in Data-link Bridges in accordance with the classification result. We use RESTful API [29] as the communication method between the Detection Agent and the Rule Arbitrator. While the Detection Agent finishes the analysis, it collects the 5-tuple information and classification result, puts them into the body of RESTful HTTP request with the JSON format, such as {"5 tuple":{"src ip":"1.1.1.1", "src port":15931, "dst ip":"2.2.2.2", "dst port":80, "protocol":6}, "application":"Zeus Botnet"}, and sends it to the Rule Arbitrator.

3.4. Modify Flow Entry on the Data-link Bridges. When the Rule Arbitrator receives the RESTful HTTP request sent from the Detection Agent, it will modify flow tables according to the RESTful HTTP request in the Data-link Bridges. All related Data-link Bridges will thus handle packets according to their flow tables. For example, when the Rule Arbitrator receives a RESTful HTTP request which indicates that the 5-tuple information is source IP address = 1.1.1.1, TCP port = 15931, destination IP address = 2.2.2.2, TCP port = 80, the communication protocol is TCP (6), and the class is Zeus botnet, it will add a flow entry in the Data-link Bridges where the match field is set as *OFPMatch (srcip = "1.1.1.1", srcport = 15931, dstip = "2.2.2.2", dstport = 80, proto = 6)*, and the action field will be *DROP*. Thus, if there is any packet that matches the match field, it will be considered as a Zeus botnet-related and will be dropped when it get into the Data-link Bridges. In other words, packets matching the P2P botnet traffic pattern will be dropped after our analysis. Additionally, we can control the lifespan of flow entries by setting their *idle_timeout* or *hard_timeout* attribute. We can also design a database to record victims that involve in botnet communication and set the time out of flow entries as 5 minutes when the first time the victims are detected, and 30 minutes for the second time, and so on. Thus, we can drop suspicious packets or forward them for further analysis, such as *honeypots* by modifying the destination of the packet field (i.e., set-field). Our system can thus achieve automatic management of network traffic.

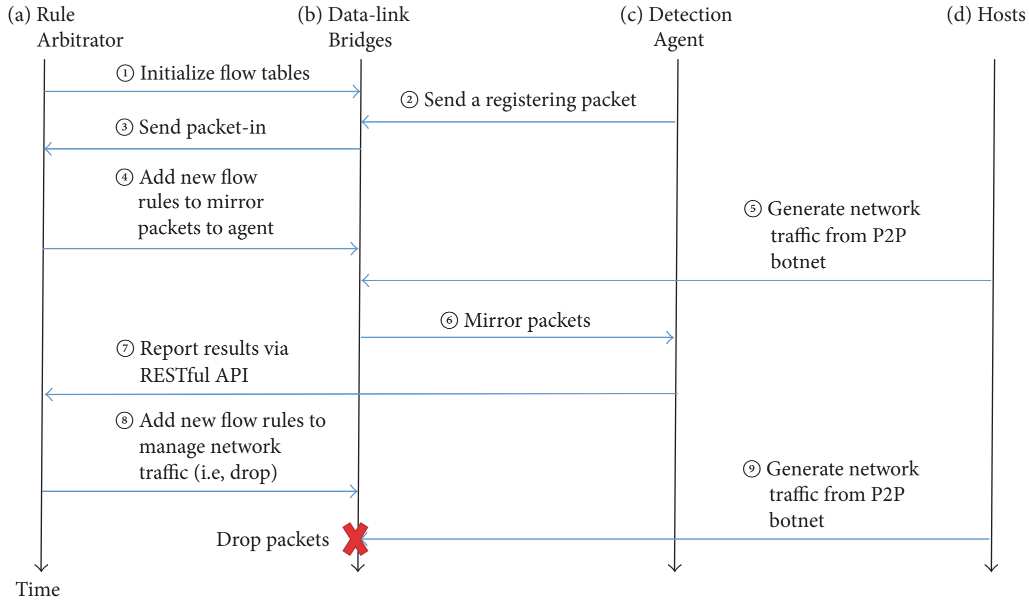


Figure 4: Sequence diagram of bot detection.

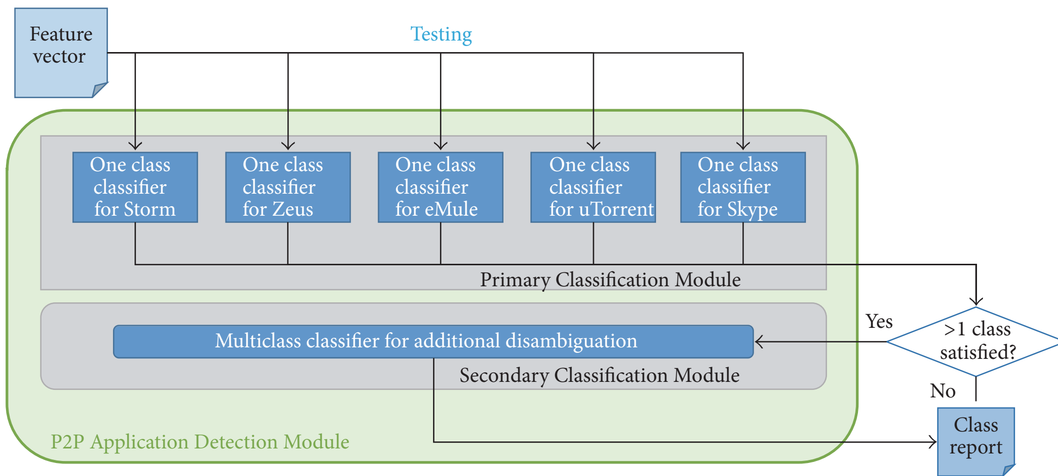


Figure 5: Testing phase of P2P Application Detection Module.

4. Experiments and Evaluation

4.1. System and Network Environment Implementation. With the virtualization technique, we use *OpenStack* [30] to build our virtual network experimental environment. OpenStack is an open source cloud computing software platform developed by Rackspace Hosting and NASA. We launch five virtual machines as network nodes and devices with *libvirt* [31] and *KVM* [32] for our experiment test bed. We make one of the virtual machines as the controller node in OpenStack, one as network node, and the other three virtual machines as compute nodes. In the controller node, we install related packages. We also install *Neutron*, which is responsible for all network related job in the network node. We build virtual OpenFlow switches with *Open vSwitch* [33] in the network node and all compute nodes. We integrate the Open vSwitches with OpenStack; thus every VM launched by OpenStack can be plugged with an Open vSwitch. We install the Ryu controller in the network node, and we bind the Ryu controller on the IP address of network node to allow Open vSwitches to connect to Ryu controller with the IP address. We design our own Ryu controller app by extending the sample code of Ryu app (i.e., simple_switch_13.py [34]) to manage forwarding rules and dynamically add or modify flow entries by the event handler.

4.2. Data Collection. From the authors of *PeerRush* [5], we collect network traffic trace files generated from P2P botnets and normal P2P applications. We use the Zeus and Storm botnet trace files in the dataset as the botnet network traffic and the network trace files of eMule, uTorrent, and Skype as benign P2P application network traffic.

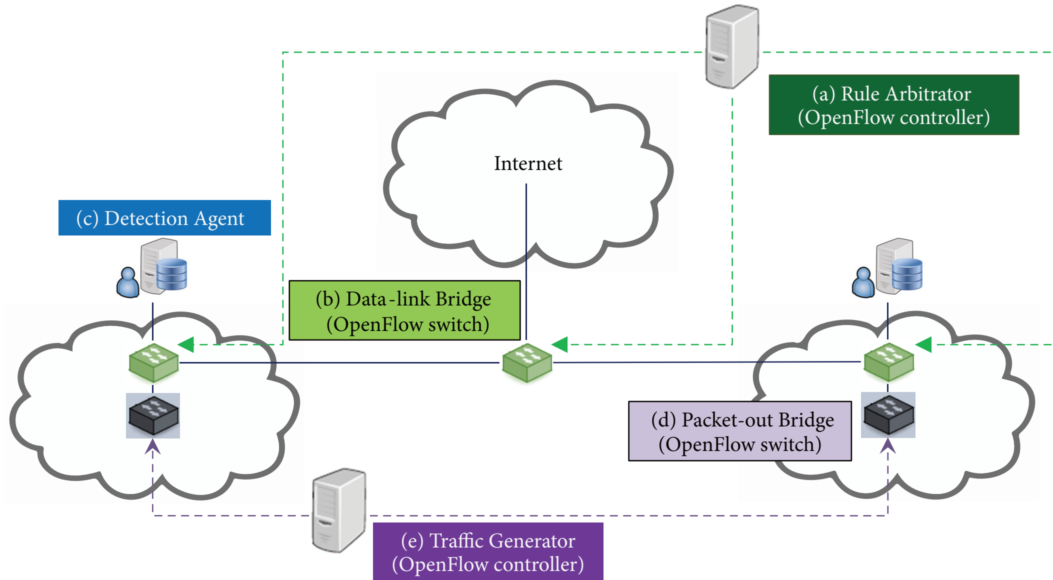


FIGURE 6: Implementation with Traffic Generator: (a) the Rule Arbitrator as in Figure 3(a); (b) the Data-link Bridge as in Figure 3(b); (c) the Detection Agent as in Figure 3(c); (d) the packet-out bridge which simulates the hosts as in Figure 3(d); (e) the Traffic Generator reads the traffic trace file and instructs the packet-out bridges to replay.

4.3. Traffic Generator. To verify that our system does detect P2P botnet in real network traffic, we need to replay the network traffic trace files and simulate the network behaviors of real P2P botnets and applications. In order to do that, we build a Traffic Generator to replay the packets in the network traffic trace files and to simulate the network behaviors of P2P hosts.

We generate network traffic with the Traffic Generator (Figure 6(e)) and the packet-out bridges (Figure 6(d)) with the popular packet replay tool: *TcpReplay* [35]. A common practice deploys virtual machines or uses physical machines to replay network traffic trace file and verify the system security or the performance of firewall [9, 18]. However, the main drawback is that the trace files are usually collected at the gateway of the network or from a single machine such as *HoneyPot* [36]. If we would like to replay these network traffic trace files on multiple hosts or VMs and simulate the network behavior between multiple hosts, then we have to modify the information in the trace files, such as source/destination IP addresses and source/destination ports, and then rewrite them into several versions for different hosts to replay. This would be a tedious job.

We take advantage of the functionality that OpenFlow controllers can send packets through OpenFlow switches (i.e., Packet Out) to build our Traffic Generator. First, we launch the Traffic Generator (Figure 6(e)), and it reads the network traffic trace file (i.e., the pcap file Figure 7(b)) and then uniformly assigns all IP addresses occurring in the network trace file to the packet-out bridges (Figure 7(d)) according to the number of the packet-out bridges or any specific arrangement. Thus we map all P2P host IP addresses to the packet-out bridges and treat each of them as a gateway of a subnet. The host IP addresses are treated as the P2P host addresses. After that, we make the Traffic Generator read the network trace file again to send Packet Out from the packet-out bridge whose source IP address corresponds to the source IP of packet and propagate the packet to the packet-out bridge whose IP is used as the destination IP address. We can simulate the scenario in the traffic trace file and mimic the P2P network communication behavior. We show the details of Traffic Generator as follows. We use RESTful API to control and trigger the process of the Traffic Generator.

(1) Load Trace File into the Traffic Generator. At first, the Traffic Generator reads packets in the network trace file (i.e., the pcap file). Then it extracts the source IP address and destination IP address and records the information of the packet-out bridges, such as MAC address, the port number connecting to their neighbor Data-link Bridge, and the identifier number given by the Traffic Generator.

(2) Start Preprocessing. In this step, the source and destination IP addresses of each packet in the file which are read by the Traffic Generator, and then they are uniformly mapped to the Packet-out Bridges. In other words, each IP address of the P2P hosts is mapped to one of the Packet-out Bridges. The purpose of this step is simulating that the P2P hosts are connected to their mapped Packet-out Bridge.

(3) MAC Learning. To make packets propagate properly with their IP addresses, we need to make the Data-link Bridges perform the MAC learning initiated by their controller, which is the Rule Arbitrator. In this step, Packet-out Bridges send an ARP request packet to each other and reply when receiving the requests. After all of them have been done, the Data-link Bridges have accomplished the MAC learning.

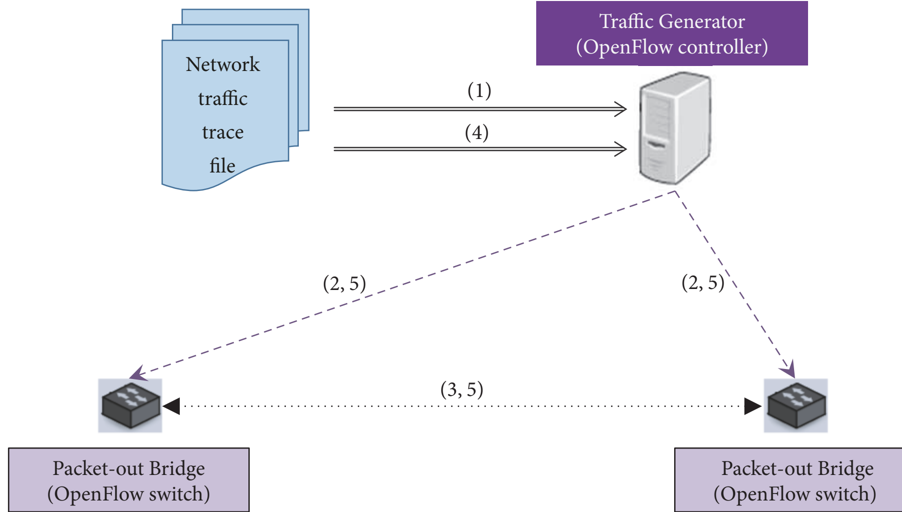


Figure 7: Traffic Generator: generate traffic with Ryu controller and Open vSwitch.

(4) Handle Packet One by One. After MAC learning, the Traffic Generator reads the packets again from the network trace file and modifies the source and destination MAC addresses of the packets to their mapped address of the Packet-out Bridge. As a result, each packet can be sent from the Packet-out Bridge mapped to its original source IP address and received by the Packet-out Bridge mapped to its original destination IP address. Therefore, the communication among P2P hosts is simulated.

(5) Send Packet Out. At the final step, the Traffic Generator sends packets out from the specific Packet-out Bridge according to the mapping between source IP addresses and the identifier numbers of the bridge. Note that, with this setting, we can define the replay speed, such as sending the packets with the clock rate of CPU or delaying a few milliseconds between two consecutive packets, or we can also send out the packets following the timestamp to simulate the original transfer rate.

Table 9: Network traffic evaluation and detection accuracy.

Class	Packets	Flows	Detection accuracy
Storm	1747562	39281	92.824%
Zeus	21714	9098	98.299%
eMule	1104034	8905	95.046%
uTorrent	1067949	33087	92.752%
Skype	1177166	8121	95.024%

Table 10: Real world evaluation result.

Class	True positive rate	False positive rate
Storm	94.16%	1.41%
Zeus	98.46%	0.13%
eMule	96.04%	0.43%
uTorrent	91.75%	0.07%
Skype	93.47%	0.48%

4.4. Evaluation. We experiment with the network trace files of different P2P botnets [5] and benign P2P applications. We would like to point out that the accuracy follows from the learning algorithms and the features vectors. Whenever there are new learning algorithms, there is a chance to improve the accuracy further. Through evaluation results, we show that our approach achieves what we expected.

We choose records of 24 hours for each P2P network trace file. These records are different from the dataset we used to train classifiers earlier. Table 9 summarizes the data we used for evaluation. Note the packet number for Zeus is much smaller, because the acquired dataset has fewer traffic from Zeus.

We use the Traffic Generator to replay packets, which are mirrored to Detection Agent through the Data-link Bridges. The Detection Agents analyze the network traffics and send the analysis report to the Rule Arbitrator with RESTful API. Then the Rule Arbitrator gives specific flow rules to related Data-link Bridges according to the analysis result. We test every traffic sample independently. Table 9 shows the detecting performance.

To perform an experiment closer to real world scenario, we mix different network traffic trace files of P2P botnets and P2P applications. Again we can effectively analyze and detect network traffics generated from different P2P botnets and benign P2P applications. The results are shown in Table 10.

The experimental results provide evidences showing that our approach can effectively distinguish P2P malicious packets from normal ones with pretty good accuracy. From Tables 9 and 10, the performances are close. Since the sample of Zeus is relatively rare, there is a good chance of building a good model from the training data. Thus, it has the highest accuracy. Among which Storm and uTorrent have lower accuracy in both scenarios. As the number of test packet is much larger, it is likely that the training model can be further improved with more training data.

5. Related Works

There are many works addressing botnet detection. Some appeared before the SDN technology. We first survey those without SDN. Then we review those with SDN.

5.1. Network-Based and Flow-Based Botnet Detection. There are many researches about detecting botnet based on various algorithms, assumptions, and system architectures, especially the network-based and flow-based botnet detection. Since modern botnets usually use P2P as their architecture, related researches of detecting P2P botnet are getting more attention these years.

Alshammari and Zincir-Heywood [37] showed that C4.5 based approach performs much better than other machine learning algorithms on the identification of both SSH and Skype traffic. This work focused on the application of machine learning on identifying some network traffics. However, botnet detection is not discussed in their work.

Graham et al. [38] experimented on how flow export could be used to capture network traffic parameters for identifying C&C server within a virtual machine of a cloud platform. They used NetFlow exported from virtual switches to detect C&C botnets within virtualized infrastructures. In their analysis phase, a neural network algorithm was used to detect traffic patterns among captured traffic. It is not a typical machine learning algorithm. They did mention the SDN technology for the protection phase. But we do not see any related discussion in their implementation.

Saad et al. [9] focused on detecting P2P botnet based on machine learning algorithm and flow-based method. They defined 17 different features within botnet's communication and control stage. They experimented the detecting performance of 5 different machine learning algorithms (i.e., Support Vector Machine, Nearest Neighbors Classifier, Artificial Neural Network, Naive Bayes Classifier, and Gaussian Based Classifier). They also created a dataset for experiment by merging the botnet dataset and benign dataset together. They got botnet network traffic of Storm botnet and Waledac botnet from the French section of the honeynet project, and they asked Traffic Lab at Ericsson Research in Hungary for benign network traffic generated from different P2P application and web browsing and gaming network traffic such as Quake and World of Warcraft. They used *TcpReplay* to replay the mixed packets and used *WireShark* to capture and record this traffic with the *ISOT dataset* [39] to evaluate its detection framework. Again, SDN and related techniques are not mentioned in their work. As in our experiments, they noticed that SVM algorithm took much more time for training. We have similar prediction rate as theirs, but we further integrate with SDN mechanism for automatic network management.

Rahbarinia et al. [5] proposed a different botnet detection framework for traditional network. They not only detected P2P botnet but also categorized different kind of unwanted P2P traffic. They constructed their detection framework by considering the characteristic that a different P2P application creates distinguishable network traffic pattern. In their design, they build a classification profile for every P2P application to analyze and categorize network traffic. They also collect different kind of botnet network traffic sample (i.e., Storm, Zeus, and Waledac) and different kind of benign network traffic of P2P application for experiment. They applied machine learning algorithms for training and prediction. Their experiment result shows that their profile has well performance on categorize network traffic. Based on this work, we integrate it with SDN technology for network management.

Narang et al. [8] used a 2-tuple conversation approach for P2P botnet detection and relied only on the information obtained from the TCP/UDP/IP headers. They also used machine learning algorithms to classify the traffic and obtained pretty good prediction rate. Their work has nothing to do with SDN and the traffic data for analysis is different from ours.

5.2. Anomaly Detection in SDN. Braga et al. [20] proposed a detection frame work for lightweight DDos flooding attack. They implemented a NOX controller [40] app and collected several features of flow rules in OpenFlow switches under the controller periodically and used Self-Organizing Map (SOM) algorithm to analyze if there is any DDoS attack happening. They built a test bed and used a DDoS attack tool named *Stacheldraht* to launch DDoS attack for evaluating the detection performance. Their results showed that their framework could collect features for analysis with low overhead and obtained pretty good detection performance. The differences between our work and theirs are that we detect botnet behavior and pattern by collecting and analyzing packets and network traffic passing through OpenFlow switches directly, and we analyze and detect suspicious botnet network traffic behavior in their communication and control stage, instead of the attack stage. Furthermore, when they find the DDoS attack happening, they need to notify the network administrator for additional handling instead of automatically adding flow rules to cut off the network traffic with SDN.

Giotis et al. [18] used sFlow [41] to collect network flow information and analyze the anomaly behavior (i.e., DDoS attack, Worm propagation, port scan, etc.) by calculating the entropy of source IP address, source port, destination IP address, and destination port between network flows. They used Nox controller and Open vSwitch to build their experimental environment and used *TcpReplay* to replay packets and traffic samples collected in their test environment. Their result shows their system can collect flow information with sFlow efficiently and mitigate the damage by modifying flow rules in flow table when finding any anomaly behavior. The main differences between theirs and our work are that they do not focus on detecting P2P botnet or P2P application, and their test environment is much simpler than ours (i.e., they used only one OpenFlow switch). Moreover, since our system separates Detection Framework (i.e., Detection Agent) from SDN controller, the effect on network performance is negligible.

Wijesinghe et al. [42] applied IPFIX for capturing the traffic flow from the OpenFlow switches to detect bots. Their main focus is to deal with the bot attacks in data plane.

They defined generic template with IPFIX that can be applied to SDN switches from different vendors. They also used machine learning to classify the traffic, but they did not address much on the performance and the features used. We use different method to collect traffic information.

Recently, Yan et al. [43] gave a very nice survey on SDN and DDoS attack in cloud, while the integration of P2P botnet detection and machine learning is not their focus.

6. Conclusion

In the past few years, the network security incidents have increased significantly. Although there are many researches about P2P botnet detection, most of them need further assistance from network administrator. In summary, this paper makes the following contributions.

(i) We propose a system which can detect and categorize P2P traffic in SDN with machine learning, automatically and flexibly adjust flow entries to manage network traffic, and thus reduce the load of network administrator.

(ii) We experiment our system in a test bed to evaluate the performance of classification accuracy and traffic management. Experiment results show that our system can detect all the considered types of P2P network traffic with high accuracy rate and automatically manage traffic with flow entries through SDN controller.

We believe similar techniques can be applied to real network environment and can orchestrate with other network security system in traditional network and SDN.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the Ministry of Science and Technology of Taiwan under Contracts MOST 105-2622-8-009-008, 105-2221-E-009-103-MY3, and 106-3113-E-009 - 001.

References

[1] Z. Abaidy, M. Rezvani, and S. Jha, "MalwareMonitor: An SDN-based framework for securing large networks," in *Proceedings of the 2014 ACM CoNEXT Student Workshop*, pp. 40–42, aus, December 2014.

[2] S. S. C. Silva, R. M. P. Silva, R. C. G. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, no. 2, pp. 378–403, 2013.

[3] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in *Proceedings of the International Conference on Computing, Networking and Communications (ICNC '14)*, pp. 797–801, IEEE, February 2014.

[4] S. Shin, Z. Xu, and G. Gu, "EFFORT: efficient and effective bot malware detection," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 2846–2850, Orlando, Fla, USA, March 2012.

[5] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "PeerRush: Mining for unwanted P2P traffic," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 7967, pp. 62–82, 2013.

[6] S. Guntuku, P. Narang, and C. Hota, "Real-time Peer-to-Peer Botnet Detection Framework based on Bayesian Regularized Neural Network," CoRR 2013.

[7] M. Stevanovic and J. M. Pedersen, "Machine learning for identifying botnet network traffic," Tech. Rep., Aalborg University, Aalborg, Denmark, 2013.

[8] P. Narang, S. Ray, C. Hota, and V. Venkatakrishnan, "PeerShark: Detecting peer-to-peer botnets by tracking conversations," in *Proceedings of the 2014 IEEE Computer Society's Security and Privacy Workshops, SPW 2014*, pp. 108–115, usa, May 2014.

[9] S. Saad, I. Traore, A. Ghorbani et al., "Detecting P2P botnets through network behavior analysis and machine learning," in *Proceedings of the 9th Annual International Conference on Privacy, Security and Trust (PST '11)*, pp. 174–180, IEEE, Montreal, Canada, July 2011.

[10] X. Hu, M. Knysz, and K. G. Shin, "Measurement and analysis of global IP-usage patterns of fast-flux botnets," in *Proceedings of the IEEE INFOCOM 2011*, pp. 2633–2641, chn, April 2011.

[11] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC 2012*, pp. 129–138, usa, December 2012.

[12] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[13] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[14] D. Kreutz, F. M. V. 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.

[15] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[16] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 55–60, Hong Kong, China, August 2013.

[17] 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 Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, IEEE, Krakow, Poland, May 2014.

[18] 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, pp. 122–136, 2014.

[19] "sFlow: Sampled flow," http://www.sflow.org/.

[20] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN '10)*, pp. 408–415, Denver, Colo, USA, October 2010.

[21] S.-C. Su, *Detecting P2P Botnet in Software Defined Network*, National Chiao Tung University, Hsinchu, Taiwan, 2015.

[22] J. Leonard, S. Xu, and R. Sandhu, "A framework for understanding botnets," in *Proceedings of the International Conference on Availability, Reliability and Security, ARES 2009*, pp. 917–922, jpn, March 2009.

[23] "OpenFlow Switch Specification Version 1.3.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[24] "NETMATE: Network Measurement and Accounting System," http://archive.li/6y1im.

[25] R. Alshammari and A. Nur Zincir-Heywood, "A flow based approach for SSH traffic detection," in *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2007*, pp. 296–301, can, October 2007.

[26] P. Narang, J. M. Reddy, and C. Hota, "Feature selection for detection of peer-to-peer botnet traffic," in *Proceedings of the 6th ACM India Computing Convention: Next Generation Computing Paradigms and Technologies, Compute 2013*, ind, August 2013.

[27] "scikit-learn: Machine Learning in Python," http://scikit-learn.org/.

[28] "K Nearest Neighbor Algorithm," http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[29] "Representational State Transfer," http://en.wikipedia.org/wiki/Representational_state_transfer.

[30] "OpenStack: Open source software for creating private and public clouds," http://www.openstack.org/.

[31] "Libvirt: The virtualization API," http://libvirt.org/.

[32] "KVM: A Full Virtualization Solution for Linux on x86 Hardware Containing Virtualization Extensions (Intel VT or AMD-V)," http://www.linux-kvm.org/page/Main_Page.

[33] "Open vSwitch: Production Quality, Multilayer Open Virtual Switch," http://openvswitch.org/.

[34] "A Component-based Software Defined Networking Framework," http://osrg.github.io/ryu/.

[35] "TcpReplay: Replay The Traffic Back Onto The Network," http://tcpreplay.synfin.net/.

[36] "The Honeynet Project," http://www.honeynet.org/project.

[37] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying SSH and Skype," in *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, can, July 2009.

[38] M. Graham, A. Winckles, and E. Sanchez-Velazquez, "Botnet detection within cloud service provider networks using flow protocols," in *Proceedings of the 13th International Conference on Industrial Informatics, INDIN 2015*, pp. 1614–1619, gbr, July 2015.

[39] "SOT: combination of several existing publicly available malicious and non-malicious datasets," ISOT Research Lab, http://www.uvic.ca/engineering/ece/isot/datasets/.

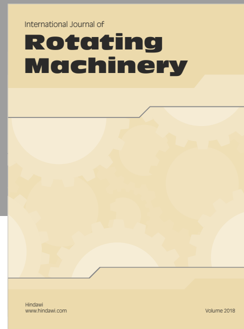
[40] N. Gude, T. Koponen, and J. Pettit, "NOX: towards an operating system for networks," *Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

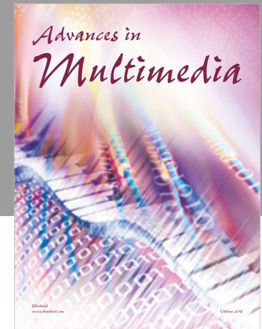
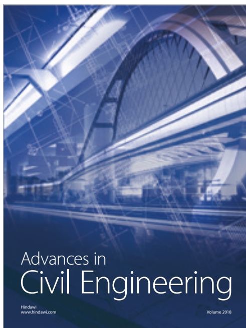
[41] P. Phaal, "sFlow Specification Version 5," 2004.

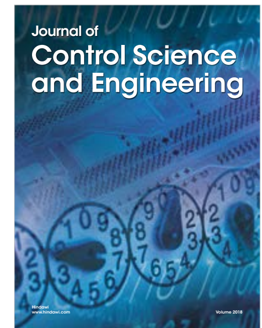
[42] U. Wijesinghe, U. Tupakula, and V. Varadharajan, "Botnet detection using software defined networking," in *Proceedings of the 2015 22nd International Conference on Telecommunications, ICT 2015*, pp. 219–224, aus, April 2015.

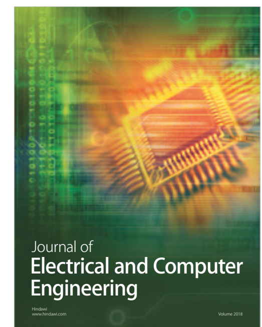
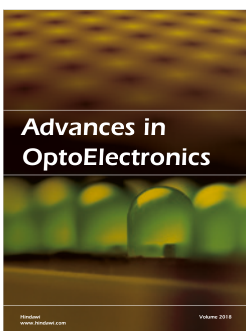
[43] Q. Yan, F. R. Yu, Q. X. Gong, and J. Q. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: a survey, some research issues, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

Journal of
Engineering

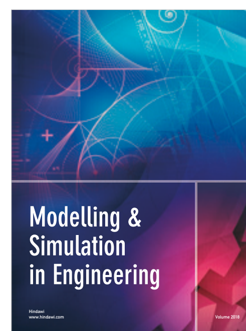
The Scientific
World Journal

International Journal of
Rotating
Machinery

Journal of
Sensors

Advances in
Multimedia

Advances in
Civil Engineering

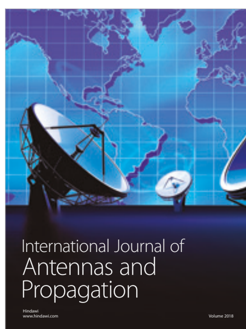
Hindawi
Submit your manuscripts at
www.hindawi.com

Journal of
Control Science
and Engineering

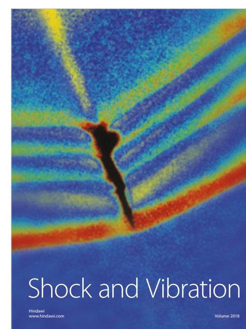
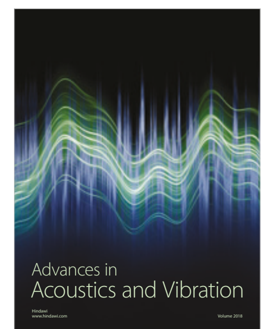
Journal of
Robotics

Journal of
Electrical and Computer
Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and
Observation

Modelling &
Simulation
in Engineering

International Journal of
Aerospace
Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and
Propagation

Active and Passive
Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration