Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow

Rodrigo Braga, Edjard Mota, Alexandre Passito
Departamento de Ciência da Computação
Universidade Federal do Amazonas
Av. Rodrigo Octavio, 3000 - 69077-000 - Brazil
Email: {rodrigo.braga, edjard, passito}@dcc.ufam.edu.br

***Abstract*—Distributed denial-of-service (DDoS) attacks became one of the main Internet security problems over the last decade, threatening public web servers in particular. Although the DDoS mechanism is widely understood, its detection is a very hard task because of the similarities between normal traffic and useless packets, sent by compromised hosts to their victims. This work presents a lightweight method for DDoS attack detection based on traffic flow features, in which the extraction of such information is made with a very low overhead compared to traditional approaches. This is possible due to the use of the NOX platform which provides a programmatic interface to facilitate the handling of switch information. Other major contributions include the high rate of detection and very low rate of false alarms obtained by flow analysis using Self Organizing Maps.**

***Index Terms*—Network Security, Programmable Networks, Artificial Neural Networks.**

I. Introduction

Flooding-based distributed denial-of-service (DDoS) attack detection is one of the main challenges to Internet security today. The DDoS attack mechanism relies on the exploitation of the huge resource asymmetry between the Internet (comprised of thousands of hosts), and a victim server's limitation in dealing with an exceptionally large number of fake requests. As a consequence, legitimate user requests are not even processed because system resources run until exhaustion, and the victim is taken off the Internet.

This kind of attack has been facilitated, in part, by user-friendly tools available such as Stacheldraht [1]. These tools allow even users lacking expertise in computer networks to launch massive attacks to a target. Moreover, the use of Internet Protocol (IP) spoofing makes it difficult to track the attacker. One of the most recent victims of DDoS was the Twitter website which had its services unavailable for many hours after an attack [2].

What challenges should one tackle to successfully detect a DDoS attack? The first difficulty arises from packet header fields being modified to look like normal ones. As a result, distinguishing between legitimate packets of normal traffic and useless ones sent by compromised hosts to their victims is a very hard task. The second difficulty is that of the huge number of packets to be analyzed. These are challenges that together make the accuracy of detection difficult and its response time even worse.

In this paper, we propose a lightweight method to DDoS attack detection based on traffic flow features to tackle the challenges above. This method is implemented over a NOX-based network, where OpenFlow (OF) switches [3] keep Flow Tables with statistics about all active flows. All feature information needed is accessed in an efficient way by means of a NOX controller [4] and then processed by an intelligent mechanism of attack detection. To the best of our knowledge, this method is in direct contrast to existing approaches, which all require heavy processing in order to extract feature information needed for traffic analysis. It is for this reason that we refer to our method as lightweight.

This paper is organized as follows. In Section II, we address approaches similar to our method. In Section III, we briefly describe NOX/OpenFlow and SOM. Section IV describes how our SOM flow-based detection method works. Section V presents our experiments and Section VI our results. Finally, Section VII concludes the paper and discusses future work.

II. Related Work

Some flow-based methods for detecting attacks on traffic have already been proposed. In [5], five features were presented to compose, along with a Bloom filter, a ”white list” in which a packet is routed to its destiny only if its source host belongs to that list. One disadvantage of this approach is its deployment on hardware (switches). Future updates are very likely to be difficult even in the best case.

Detection functions composed with flow information to generate threshold values were proposed by [6] in order to characterize whether the traffic is becoming abnormal or staying normal. In this work, a flow generator builds up flows and stores them in a database. Flows are built from packets gathered from a pipeline probing point. These preprocessing steps together increase the overhead of the entire process.

The intelligent mechanism employed by our method is based on Self Organizing Maps (SOM) [7], an unsupervised artificial neural network trained with features of the traffic flow. We use SOM to classify network traffic flows as either normal or abnormal, i.e. a potential attack, taking statistics about flows as parameters for the SOM computation.

Previous works, e.g. [8], [9], [10], [11], [12], utilized the SOM technique to detect abnormal traffic. But a slight variation of SOM was proposed by others to increase the accuracy of detection, including: a) Emergent SOM was used by [13] to increase the grid of artificial neurons to tens of thousands; and b) a grey relational coefficient technique was used by [14] to optimize the process of adjusting the weight of neurons in the neighborhood of each node.

All of the works mentioned classify traffic either as normal or abnormal based on connection information. In general, they evaluate their classification results according to the KDD-99 dataset. To extract from an actual traffic the features used in this dataset, it would be necessary to preprocess the packets flowing to the victim. In [15], it is shown how to transform traffic log files, generated by a tcpdump application, into connection records. In the worst case scenario, this technique leads to an extremely high overhead caused by a DDoS flooding attack with a high flow of packets.

Our approach differs from the ones mentioned in the following aspects. First, the features of interest for detection are extracted with low overhead. Second, we can easily update our detection mechanism to cover new kinds of attacks, or change our classification technique. Third, as more than one switch can be monitored by the NOX platform, it is possible to quickly add or remove switches from the detection loop. Furthermore, as we shall see, the experimental results obtained with our SOM approach proved to be very efficient.

III. Background

A. NOX/OpenFlow Overview

Recently proposed software-defined networks control underlying switching hardware by means of an external, detached control plane [16]. This control plane consists of three parts: the OpenFlow protocol, a network operating system, and network applications that run on top of the network operating system.

The OpenFlow (OF) protocol [3] provides a common interface to control how packets are forwarded by accessing the data plane's internal flow tables, configuration, and statistics. A flow can be defined as a group of packets which belong to the same time interval and share the same specific features.

Per-flow statistics are a compound of three values: 1) *Received packets* which count the number of packets matching the flow they belong to; 2) *Received bytes* which store the amount of bytes received by the current flow; and 3) *Duration* of the time interval a flow spends in the Flow Table of the switch being analyzed. If a flow entry of a given switch does not receive any new packets, then it is automatically deleted from the switch's Flow Table by the network operating system.

Network operating systems allow centralized watch and control of the network from the packet flow viewpoint. On top of this structure, an interface is provided to allow the development of applications that will either collect information or manage the resources of the network. NOX [4] is the most comprehensive implementation of a network operating system to date and uses the OpenFlow protocol to access the data plane.

The network operating system, hereafter called NOX, interacts with OF switches to request information, or send instructions to add or remove flows from a switch's Flow Table. Figure 1 depicts the operation of the NOX and OpenFlow mechanism.

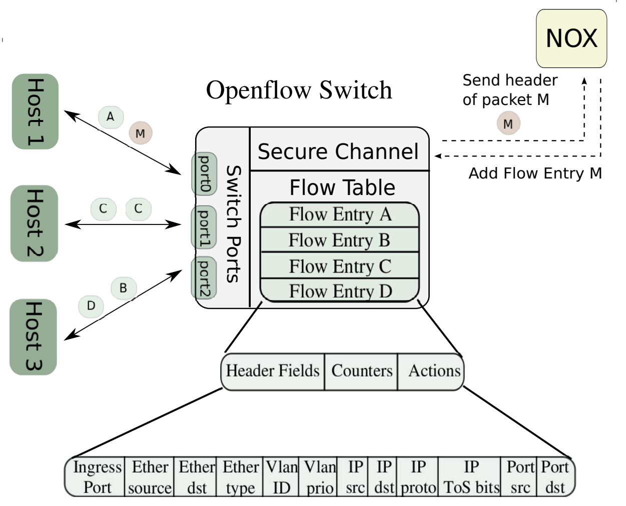

Figure 1. *Operation of NOX/OpenFlow mechanism.*

In this configuration, when a new packet arrives through some switch port, it is compared against all flow entries using the fields seen in Figure 2. If this packet matches an existing entry in the switch's Flow Table, the switch updates its counters and executes the associated actions. Otherwise, the packet is sent to the NOX controller through a secure channel. Such packets are called flow-initiations [4].

In Port	Ether			Vlan		IP				Port	
	src	dst	type	id	prio	src	dst	proto	ToS bits	src	dst

Figure 2. *Header fields of Flow Entry*

Through its applications, the NOX decides what to do with flow-initiations by associating actions to them, and by adding a new entry into the Flow Table with this information. Then, the same actions will be applied to all packets with the same features that arrive at the switch. Details about the kinds of actions that can be applied by the OpenFlow switch are provided in [17].

B. Self Organizing Maps

Self Organizing Maps (SOM) [7] is an artificial neural network which transforms a given n-dimensional pattern of data into a 1- or 2-dimensional map or grid. This transformation process is done following a topological ordering, where patterns of data (synaptic or vector weights) with similar statistical features are gathered in regions close to each other in the grid. This learning process can be classified as competitive-based because neurons compete against each other to be placed at the output layer of the neuron network, but only one wins, as illustrated by Figure 3. It is also unsupervised because the neuron network learns only with entry patterns, reorganizing itself after the first trained data and adjusting its weights as new data arrive.

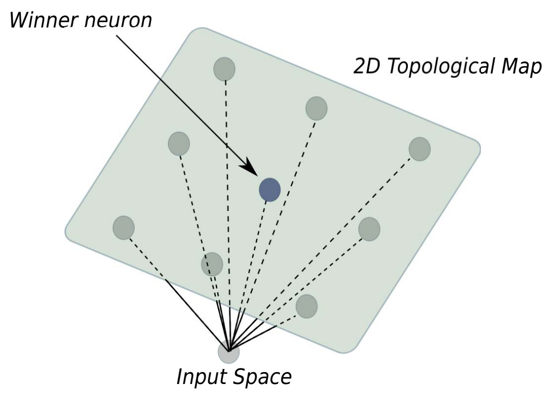

Figure 3. *Example of a Kohonen's SOM map*

A detailed description of the complete SOM algorithm is presented in [18]. In what follows, we provide a summary of the main steps of how the SOM's learning process works.

1) Initialization: at the beginning of the process all neuron vectors have their synaptic weights randomly generated. Such vectors must have the same dimension of the entry pattern space.
2) Sampling: a single sample x is chosen from the entry pattern space, and fed to the neuron grid.
3) Competition: based on the minimum Euclidean distance criterion the winning neuron $i(x)$ is found as follows:

$$i(x) = \arg\min_j \|x - w_j\|, \ j = 1, 2, ..., l \tag{1}$$

where l is the number of neurons in the grid.
4) Synaptic adaptation: after finding the winning neuron, all synaptic weights of each neuron vector are adjusted:

$$W_j(t+1) = W_j(t) + \eta(t)\Theta_j(t)(x(t) - W_j(t)) \tag{2}$$

where t represents the current instant, $\eta(t)$ is the learning rate which gradually decreases with time t, and $\Theta_j(t)$ is the neighborhood function which determines the grade of learning of a neuron j according to its relative distance to the winning neuron.
5) Repeat steps 2 to 4 until no significant change happens in the topological map.

SOM is a suitable way to classify network traffic (either as normal or attack), because of its competence in recognizing patterns even with a high rate of noisy data signals. Furthermore, it can find hidden relations of data and segments in the entry space. As our entry space is composed of flows, we are thereby able to detect attacks that would be hard to notice by examining packet headers only.

IV. Lightweight Flow-based Detection of DDoS Attack

Our lightweight method for DDoS attack detection consists of monitoring NOX registered switches of a network during predetermined time intervals. During such intervals, we extract existing features of interest from flow entries of all switches. Each sample is then passed to a classifier module that will indicate, using the spatial location of the winning neuron in the topological map, whether this information corresponds to normal traffic or an attack.

The method is divided into three modules placed within the detection loop of the NOX controller, as depicted in Figure 4. The three detection loop modules are described below.

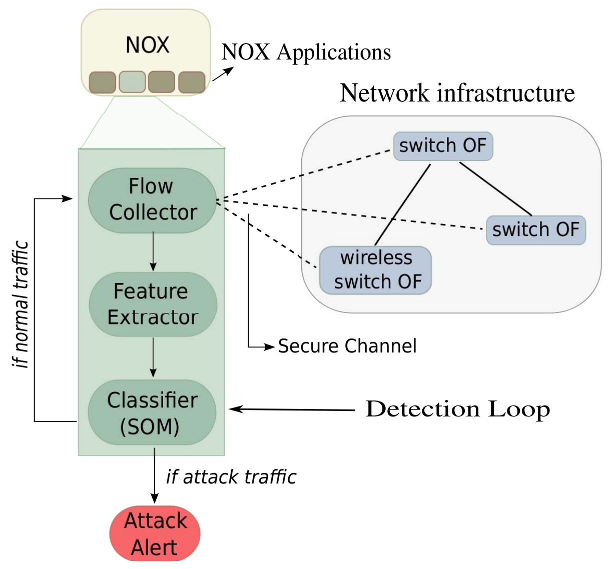

Figure 4. *Detection Loop Operation*

1) The *Flow Collector* module is responsible for periodically requesting flow entries from all Flow Tables of OF switches. Such requisitions, as well as their corresponding answers, are transmitted through a Secure Channel, i.e. a channel that is isolated from hosts connected to the switches.
2) The *Feature Extractor* module receives collected flows, extracts features that are important to DDoS flooding attack detection, and gathers them in 6-*tuples* to be passed to the classifier. Section IV-B gives a more detailed explanation of this 6-*tuple*. Every 6-*tuple* is associated to the switch it belongs by a switch ID, which can be easily obtained by the NOX.
3) The *Classifier* module analyzes whether a given 6-*tuple* corresponds to a DDoS flooding attack or to legitimate traffic. This classification can be made by any statistical or learning method. In this work we used SOM as the classification method.

A. Collecting flows using NOX/OpenFlow

First, an OF switch needs to be authenticated by the NOX controller which generates an ID for the switch. Only after authentication is complete are all hosts connected to the switch allowed to exchange packets. Every packet which arrives in an OF switch has its header matched against flow entries of the switch's Flow Table. In case some entry successfully matches the packet's header its statistics are updated (for instance, the number of bytes and of packets are increased). Otherwise, if no flow entry (in the switch's Flow Table) matches the packet's header, then the entry is forwarded to the controller. In its turn, the controller may add, according to the defined policy, a new flow entry to the Flow Table of every switch as required for policy enforcement. Thus, the traffic generated by all hosts connected to a given OF switch will populate the Flow Table of said switch.

The collection of flow entries from an OF switch is performed at predetermined time intervals by the controller. From this collection important features are extracted to classify traffic as normal or as an attack. As the collector gathers samples from all OF switches authenticated by NOX, the switch ID is used to help the classifier module pinpoint in which OF switches DDoS flooding attacks were detected.

The definition of the time interval to collect flow entries is of great importance. If collection is made at infrequent time intervals, then there will be a delay to detect an attack and consequently a reduction of the time available for a possible mitigation. On the other hand, if the time interval for collection is too short, there will be an increase of packets requesting flows which will lead to an increase in the overhead of our detection mechanism.

By default, every OpenFlow switch registered at the NOX is automatically added into the detection loop. But the network administrator can also restrict sample collection to those switches that are most relevant.

As the NOX manages information of network switches in a centralized way, we are able to monitor all selected switches and analyze their traffic from the point of view of a DDoS attack. This analysis is done by the controller in our method. Whenever a given 6-*tuple* is classified by SOM as an attack, an alert is immediately sent to the network administrator.

B. Selecting traffic features

We employ four features proposed in [5]–APf, PPf, GSf, and GDP–to compose the 6-*tuple* utilized in our method. The remaining two features, i.e. ABf and ADf, were chosen from among those most likely to influence the decision of whether a traffic flow is normal or an attack. A more detailed description of each field of the 6-*tuple* follows.

1) Average of Packets per flow (APf): One of the main features of DDoS attacks is the source IP spoofing, which makes the task of tracing the attack's original source very difficult. A side effect is the generation of flows with a small number of packets, i.e. about 3 packets per flow. Given that normal traffic usually involves a higher number of packets, we compute the median value. Before computing this value, we order the flows in ascending order based on the number of packets per flow. Equation 3 is then used to compute the median value, where X is the number of packets per flow, and n is the number of flows.

$$md(X) = \begin{cases} X((n+1)/2), & \text{if } n \text{ is odd;} \\ \dfrac{X(n/2) + X((n+1)/2)}{2}, & \text{otherwise.} \end{cases} \tag{3}$$

The use of this feature for the SOM computation was initially proposed by [5], although a simple average computation was employed. Our tests indicated, however, that computing the simple average may lead to misinterpretations in cases involving flow entries with an exceptionally high number of packets.

2) Average of Bytes per flow (ABf): Another peculiarity of DDoS attacks is their payload size, which is often very small in order to increase the efficacy of this kind of attack. For example, in TCP flooding attacks, packets of 120 bytes are sent to victims. The ABf feature is also computed as a median value using Equation 3, where X now represents the number of bytes.

3) Average of Duration per flow (ADf): Similarly, we propose the use of a median value for the duration of time a flow spends in the Flow Table. This feature decreases the number of false-positives when there is a small number of packets exchanged between applications. Equation 3 is also employed for this computation but now X is the duration of a flow in the switch and n the number of flows.

4) Percentage of Pair-flows (PPf): This feature allows the verification of how many pair-flows occur in the flow stream during a certain interval. For example, given any two flows, i.e. flow 1 and flow 3, the following conditions are checked to verify if these flows constitute a pair-flow.

- The source IP of flow 1 must be equal to the destination IP of flow 3;
- The destination IP of flow 1 must be equal to the source IP of flow 3; and
- Both flows must have the same communication protocol.

A DDoS attack increases the number of single-flows into the network because they send packets with a fake IP. To compute the percentage of this occurrence we use Equation 4.

$$PPf = \frac{2 * Num\_Pair\text{-}flows}{Num\_flows} \tag{4}$$

5) Growth of Single-flows (GSf): In the beginning of a flood attack the number of flows can skyrocket. To compute this growth we take the total number of flows minus two times the number of pair-flows, and divide by the time interval during which the flow feature were analyzed. This is computed using Equation 5.

$$GSf = \frac{Num\_flows - (2 * Num\_Pair\text{-}flows)}{interval} \tag{5}$$

6) Growth of Different Ports (GDP): In the same way that IP spoofing is generated by DDoS attacks, ports can also be generated at random by attacks. Thus, we compute this rate of growth using Equation 6.

$$GDP = \frac{Num\_ports}{interval} \tag{6}$$

C. *Classifying network traffic*

As mentioned above, the classifier module was implemented using SOM. Before SOM is able to classify any 6-*tuple* collected by the extractor module, it needs to be trained with a sufficiently large set of 6-*tuple* samples collected during attack traffic as well as normal traffic. In this way, SOM is able to create a topological map where different regions represent each kind of traffic. Then, whenever the trained SOM is stimulated with a 6-*tuple* extracted from collected flow entries of some OF switch, it will be able to classify it either as normal traffic or attack traffic. The amount of 6-*tuples* used for the training phase is presented in Section VI.

Our implementation of SOM consisted of a 40×40 matrix of neurons. Table I shows the parameters used for the learning process and their respective values.

Parameter	Value
Initial learning rate	0.5
Initial neighborhood radius	20
Epoch limit	3000

TABLE I
SOM TRAINING PARAMETERS

To compute the topological neighborhood a Gaussian function was used because it induces the SOM algorithm to converge more quickly than a rectangular topological neighborhood does [19].

V. Experiments

Our test scenario was built to be as realistic and reliable as possible by taking into account some suggestions proposed by [20]. Our approach included mixing different types of legitimate traffic and varying attack traffic parameters.

The legitimate traffic generated during tests is a composition of several different protocols: 85% is TCP, 10% is UDP, and 5% of ICMP (ping). These values are based on the collected network traffic during 7 years of a specific link between Japan and the USA. Over 90% of this traffic, as reported in [21], was related to TCP/UDP, and about 5% was ICMP.

Moreover, 80% of the TCP traffic are FTP-like connections in which a continuous data flow is kept between client and server. The other 20% are Telnet-like connections, whose main feature consists in sending a mist of small packets with a long interval (pause) between them.

The Stacheldraht tool was used to generate the traffic for the DDoS flooding attack. In Table II we present the types of attacks launched for the training and testing phases, along with their respective estimated number of generated flows. The values within parentheses are related to the size of packets sent during an attack, where the maximum size allowed by this tool is 1024 bytes. The description of each parameter which affected the DDoS attacks depicted in such table is presented below. Only one of those parameters was changed during each time interval while the others kept their default values.

Attack Types	Training	Tests
TCP/SYN flood	6080	137612
UDP(60) flood	2967	52733
UDP(200) flood	-	31858
UDP(400) flood	3598	40019
UDP(800) flood	-	103165
ICMP(60) flood	-	57973
ICMP(1024) flood	5113	55344

TABLE II
DDoS ATTACKS USED FOR TRAINING AND TESTING

1) *Attack rate* regulates the botnets' number of packets sent during a particular time interval. Its default value is moderate, but it can also vary from low to high.
2) *Dynamics of attack stream* adjusts the flow of packets sent during an attack. It can be set as continuous or with a pause in between packets' flows of attack. The default value was set as continuous.
3) *Dynamics of attack sync* configures botnets to send packets' flows of attack either in a synchronized or interleaved mode. Synchronized is the default value.
4) *Legitimate traffic rate* configures the number of legitimate flows of packets participating in the experiment, while the portion of mixed protocols and TCP packets stay unchanged. Moderate is the default value of this feature.

The testing scenario was implemented based on scripts available in [22], which use emulation to build a NOX-based network. Figure 5 depicts the topology used for the experiments. Network 1 corresponds to the victims' network and it is composed of three OpenFlow switches controlled by the NOX. The botnets that generate DDoS flooding traffic are in Network 2, and the backscatter traffic flows with IP spoofing to hosts placed in Network 2 and Network 3.

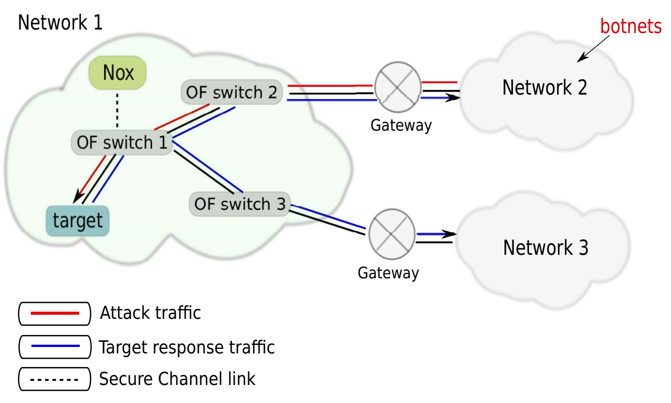

Figure 5. *Testing Scenario's Topology*

The original script was modified to allow the configuration of certain attributes, e.g. bandwidth channel and delay. The wirefilter tool [23] was used to configure the link between

Network 1 and Network 2 with 1 GBps bandwidth and 30 ms of delay. The link between Network 1 and Network 3 was set to a 1 GBps bandwidth and 10 ms of delay. All other links were assumed to have 100 MBps bandwidth.

We ran our experiments to emulate the scenario, as well as to train the SOM, on a server with an Intel Quad Core Xeon E5410 processor, 2.33 GHz, and 16GB RAM memory.

VI. Results and Analysis

A. Samples collected in experiments

Two experiments were completed. In the first one, we generated 106,000 flows during the training of the SOM. From this number, 43,000 flows were collected during an interval of attack, while the other 63,000 flows were collected during normal traffic. Table III shows the quantity of tuples that were extracted from the total number of flows.

In the second experiment, we launched attacks with packets of variable size as shown in Table II. A total of approximately 478,000 flows were generated; from these, 253,000 were produced during DDoS attacks, and the other 225,000 were generated while no traffic of attack took place. Table III lists the number of samples extracted from each OF switch during the experiments.

The values corresponding to generated flows were obtained from OF switch 1, since all packets passed through it. In both experiments, corresponding to the SOM training phase and the attack scenario, the time interval for the detection loop was set to 3 seconds.

By definition, a tuple of features is computed from a set of flow entries collected from an OF switch at a certain moment. For example, for an OF switch with 150 flow entries the extractor will compose the 6-*tuple* based on this set once it is gathered by the collector module.

	Attack Traffic Phase		Legitimate Traffic Phase	
	Training	Testing	Training	Testing
OF switch 1	1750	11845	1703	9106
OF switch 2	1750	11834	1703	9104
OF switch 3	-	11892	1702	9107
Total	3500	35571	5108	27317

TABLE III
SAMPLES USED FOR TRAINING AND TESTING

B. Time to training and classifying traffic

The training phase of SOM used 3,500 samples collected during an attack, and 5,108 samples collected during a non-attack period. We did not consider the samples generated by OF switch 3 during a period of attack because it is not in the route of attack, but it is in the route of traffic answering those packets of attack.

Table IV compares the 6-tuple and the 4-tuple in regard to the time needed to perform SOM tasks, such as training the neuron grid, classifying a given sample, and the execution of the discriminant function. The values for the classification were obtained using a system with 1.8 GHz, dual core CPU and 2 GB RAM memory, while the values for the training execution were obtained using the server that emulated the testing scenario, as described at the end of Section V.

	SOM Training execution	SOM Classification
4-tuple	5,53 hs	271 ms
6-Tuple	7,16 hs	352 ms

TABLE IV
TIME EXECUTION OF SOM DETECTION TASKS

C. Detection performance

The efficiency of our detection mechanism was evaluated through the Detection Rate measurement (DR) and the False Alarm rate (FA), computed using Equations 7 and 8, respectively.

$$DR = \frac{TP}{TP + FN} \tag{7}$$

where TP (True Positives) are attack traffic logs classified as attacks, and FN (False Negatives) are attack traffic logs classified as legitimate.

$$FA = \frac{FP}{TN + FP} \tag{8}$$

where FP (False Positives), are legitimate traffic logs classified as attack, and TN (True Negatives) are legitimate traffic logs classified as legitimate.

Table V presents the results obtained from the analysis of the features of the 6-*tuple* and 4-*tuple*. The latter does not include PPf and GSf features. These two features do not contribute to differentiate attack traffic from legitimate traffic from OF switch 1, since their values for both kinds of traffic are very similar. This happens because the victim is immediately connected to OF switch 1, and packets of attack, as well as their corresponding answers, are registered as flow-initiation of OF switch 1. Thus, all switches sharing both kinds of packets must behave in the same way.

	6-tuple		4-tuple	
	DR(%)	FA(%)	DR(%)	FA(%)
OF switch 1	98.61	0.59	98.57	0.48
OF switch 2	99.11	0.46	98.73	0.62
OF switch 3	3.52	0.12	3.27	0.13

TABLE V
DETECTION METHOD RESULTS

Table V shows that for OF switch 1 the use of 4-*tuple* (0.48) is better than the use of 6-*tuple* (0.59) mainly when false alarms are taken into account. This occurs because in some cases the values of PPf and GSf mislead SOM interpretations of the traffic. However, when collection of samples start from OF switch 2, we have the opposite situation, i.e. the 6-*tuple* has a better result (0.46 of false alarm and 99.11 of detection rate), than the 4-*tuple* (0.62 of false alarm and detection rate of 98.73). The reason is that for this switch PPf and GSf values for attack traffic and legitimate traffic are remarkably different, and make the classification of samples an easy task.

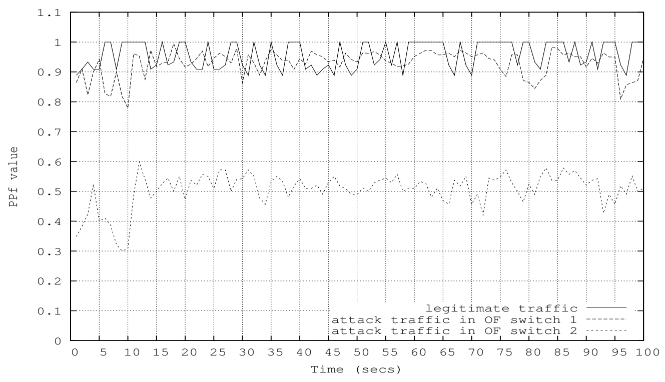

Figure 6. *Comparison of PPf traffic patterns between OF switches 1 and 2.*

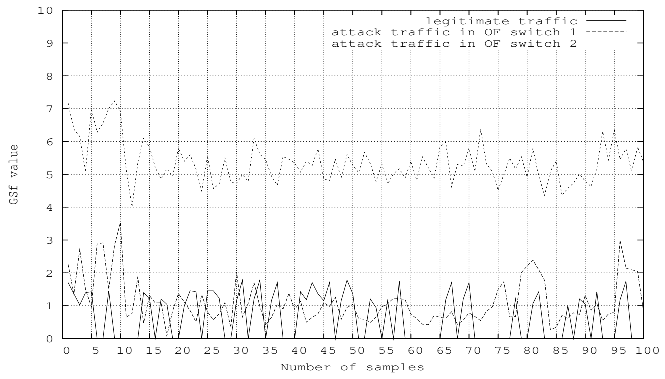

Figure 7. *Comparison of GSf traffic patterns between OF switches 1 and 2.*

Figure 6 depicts patterns of traffic between OF switch 1 and OF switch 2, corresponding only to the PPf feature. Notice that PPf values obtained from samples collected from OF switch 1 during a period of attack are in the same range (0.9 to 1.0) as the ones collected during a period of legitimate traffic. However, the samples obtained from OF switch 2 during the same period present values in different ranges (0.3 up to 0.6).

The same analysis can be made about the data in Figure 7, which depicts patterns of traffic between OF switch 1 and OF switch 2 for the same time interval, only now corresponding to the GSf feature. We can see that most of the sample values for attack traffic patterns obtained from OF switch 1 are similar to those of normal traffic patterns (varying 0 to 1.8). Values for the same feature obtained from OF switch 2 range from 4 to 7.2.

Most of the False Positive detections were obtained for periods in which the state of all switches changed from non-active to a low rate of legitimate traffic. The pattern of traffic registered during such time interval mirrors a DDoS traffic attack because features APf, ABf, and ADf end up with a range of values similar to those presented by legitimate traffic. The classification of attack traffic as normal leads SOM to draw wrong conclusions from some samples.

False Negatives often occurred when the Attack Rate parameter was set to a low value. However, an attack with a low rate of packet flow is less harmful to the victim's system.

The low Detection Rate values presented in Table V for OF switch 3 can be explained by the fact that no attack traffic passes through this switch. The values of Detection Rate for OF switch 3 (3.52 for 6-*tuple* and 3.27 for 4-*tuple*) correspond to traffic generated by the victim in response to the DDoS flooding attack.

During our experiments, we have also observed the linear growth of flow-initiations during attack traffic. This happens because packets arriving in OpenFlow switches during a DDoS attack–with IP and port spoofing–are very unlikely to match with any entry of the switch's Flow Table. Thus, for each non-matched packet, a flow-initiation is generated by the NOX controller in charge of the switch.

D. Method overhead

Considering that our approach makes use of flow-based information to classify patterns of traffic within a given time interval, and that samples of such information are collected every 3 seconds, there is a remarkable overhead reduction to the whole detection mechanism when compared to other approaches based on the KDD-99 dataset. Their additional overhead is caused by the need to collect every packet sent to a victim, and then pre-process this information to generate connection records. Besides, if we consider the worst case scenario (DDoS flooding attack with a very high flow of packets for the attack rate), then the overhead tends to be very high.

In order to compare our method to other KDD-99 dataset approaches we built another experiment in which a DDoS flooding traffic was generated with a high rate of attack and IP header spoofing (worst-case scenario). This experiment produced a huge number of flow entries that were supported by the OF switch and stressed our Collector and Extractor modules as they processed all flow entries.

Table VI shows a comparison of the CPU time to extract features needed for detection for KDD-99 dataset approaches and ours. The values for KDD-99 are those presented in [15], and reported as obtained from experiments run on a system with 2.66 GHz, dual core CPU, and 3.5 GB of RAM memory. Our values were obtained from experiments run on a system with 1.8 GHz, dual core CPU, and 2 GB of RAM memory. The time interval corresponds to generating 30,000 samples in both cases.

	Our method		KDD-99 based methods	
Number of features	6	4	9	41
CPU time(s)	154	109	237	2043

TABLE VI
COMPARISON OF OVERHEAD FOR EXTRACTION OF FEATURES

The results presented in Table VI demonstrate that our method is faster than the KDD-99 approach, even though our detection method receives data on-line! This is totally different from [15], which requires first the capture of packets from a probing point by means of a tcpdump tool. Moreover, only after storing the collected packets in a file does their process of extracting features begin.

E. Better deployment

A secondary contribution of this work concerns the great difficulty other methods have in deploying flow-based detection mechanisms. Usually, in order to access needed information from switches of interest one had to modify the software code controlling such switches. For instance, [5], installs filters within switches. Our approach takes advantage of NOX controlling the network in a centralized way. This mechanism allows us to extract flow-based information from all OF switches registered by NOX, since OpenFlow is a protocol that offers information needed in an efficient way.

Furthermore, our method also offers great flexibility for adding (or removing) OF switches to (or from) the detection loop. Another advantage is that if there is a change in the network topology, the administrator can easily adapt the detection loop to the new topology. For example, by adding switches more relevant to the detection or removing those less important.

VII. Conclusion and Future Work

This work presented a lightweight method for DDoS attack detection. We showed that our technique extracts features of interest with a low overhead when compared to approaches based on the KDD-99 dataset. It is also able to monitor more than one observation point.

The method is also very efficient at detecting DDoS attacks. It uses Self Organizing Maps, an unsupervised artificial neural network, trained with features of the traffic flow. The detection rate obtained is remarkably good as it is very close to other approaches.

As part of our future work, we plan to allow the communication between different detectors from different network domains. For example, if network A detects that an attack has been launched against a second network B, then A could inform B that it is under attack. The direction we shall take is that of the multiagent systems approach, where agents will behave as detectors and exchange information with other agents to facilitate the monitoring task of network resources.

We also intend to apply statistical methods to analyze ports of OF switches to precisely determine which hosts are launching attacks.

Acknowledgment

The first and third authors thank CAPES, the Brazilian Ministry of Education Research Agency, for grant support. We would like to thank the referees for their comments. We are most thankful to Monica Nogueira and Heloisa Nogueira for proofreading this final version.

References

[1] "The stacheldraht ddos attack tool," 2009. [Online]. Available: http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt

[2] "Twitter hit by denial-of-service attack," 2009. [Online]. Available: http://www.cnn.com/2009/TECH/08/06/twitter.attack/index.html

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev*, vol. 38, no. 3, pp. 105–110, 2008.

[5] Y. Feng, R. Guo, D. Wang, and B. Zhang, "Research on the Active DDoS Filtering Algorithm Based on IP Flow," in *2009 Fifth International Conference on Natural Computation*. IEEE, 2009, pp. 628–632.

[6] M. Kim, H. Kang, S. Hung, S. Chung, and J. Hong, "A flow-based method for abnormal network traffic detection," in *IEEE/IFIP Network Operations and Management Symposium*, 2004, pp. 599–612.

[7] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[8] M. Ramadas, S. Ostermann, and B. Tjaden, "Detecting anomalous network traffic with self-organizing maps," in *Recent Advances in Intrusion Detection*. Springer, 2003, pp. 36–54.

[9] H. Gunes Kayacik, N. Zincir-Heywood *et al.*, "A hierarchical SOM-based intrusion detection system," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 4, pp. 439–451, 2007.

[10] M. Li and W. Dongliang, "Anormaly Intrusion Detection Based on SOM," in *Proceedings of the 2009 WASE International Conference on Information Engineering-Volume 01*. IEEE Computer Society, 2009, pp. 40–43.

[11] D. Jiang, Y. Yang, and M. Xia, "Research on Intrusion Detection Based on an Improved SOM Neural Network," in *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security-Volume 01*. IEEE Computer Society, 2009, pp. 400–403.

[12] M. Alsulaiman, A. Alyahya, R. Alkharboush, and N. Alghafis, "Intrusion Detection System Using Self-Organizing Maps," in *2009 Third International Conference on Network and System Security*. IEEE, 2009, pp. 397–402.

[13] A. Mitrokotsa and C. Douligeris, "Detecting denial of service attacks using emergent self-organizing maps," in *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on*, 2005, pp. 375–380.

[14] C. Wang, H. Yu, and H. Wang, "Grey self-organizing map based intrusion detection," *Optoelectronics Letters*, vol. 5, no. 1, pp. 64–68, 2009.

[15] W. Wang and S. Gombault, "Efficient Detection of DDoS Attacks with Important Attributes ," in *CRISIS 2008, Third International Conference on Risks and Security on Internet and Systems, October 28-30, Tozeur, Tunisia*. RSM - Dépt. Réseaux, Sécurité et Multimédia (Institut Télécom-Télécom Bretagne), DREAM research group - Diagnosing, REcommending Actions and Modelling (INRIA /IRISA Rennes), 2008.

[16] S. Das, G. Parulkar, and N. McKeown, "Unifying Packet and Circuit Switched Networks," in *Globecomm Workshop on Below IP Networking*. IEEE, Nov. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5360777

[17] "Openflow switch specification," 2010. [Online]. Available: http://www.openflowswitch.org/wp/documents/

[18] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[19] Z.-P. Lo, M. Fujita, and B. Bavarian, "Analysis of neighborhood interaction in kohonen neural networks," in *6th International Parallel Processing Symposium*, Los Alamitos, CA, 1991, pp. 246–249.

[20] J. Mirkovic, S. Fahmy, P. Reiher, and R. K. Thomas, "How to test dos defenses," in *CATCH '09: Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 103–117.

[21] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "Seven years and one day: Sketching the evolution of internet traffic," in *INFOCOM 2009, IEEE*, April 2009, pp. 711–719.

[22] "Setting up a virtual testing environment," 2010. [Online]. Available: http://noxrepo.org/manual/vm_environment.html

[23] "Manual of wirefilter," 2010. [Online]. Available: http://www.linuxcertif.com/man/1/wirefilter/