Toward Network-based DDoS Detection in Software-defined Networks

Stefan Jevtic
Indiana University Purdue University Indianapolis
Indianapolis, IN
sjevtic@iupui.edu

Hamidreza Lotfalizadeh
Purdue University
West Lafayette, IN
hlotfali@purdue.edu

Dongsoo S. Kim
Indiana University Purdue University Indianapolis
Indianapolis, IN
dskim@iupui.edu

ABSTRACT

To combat susceptibility of modern computing systems to cyberattack, identifying and disrupting malicious traffic without human intervention is essential. To accomplish this, three main tasks for an effective intrusion detection system have been identified: monitor network traffic, categorize and identify anomalous behavior in near real time, and take appropriate action against the identified threat. This system leverages distributed SDN architecture and the principles of Artificial Immune Systems and Self-Organizing Maps to build a network-based intrusion detection system capable of detecting and terminating DDoS attacks in progress.

KEYWORDS

SDN, DDoS, AIS, NS-3, OpenFlow, RT, Bio-inspired

ACM Reference Format:
Stefan Jevtic, Hamidreza Lotfalizadeh, and Dongsoo S. Kim. 2018. Toward Network-based DDoS Detection in Software-defined Networks. In *Proceedings of ACM Langkawi conference, Langkawi, Malaysia, January 2018 (Langkawi'18),* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The size of the internet sphere has seen explosive growth in the last 10 years, and the rate of growth is expected to accelerate into the near future [46]. This accelerated growth is attributed to two main factors: the rise of the Internet of Things (IoT) and the emergence of cloud computing. The Internet of Things is the term attached to the growing network of small embedded devices that frequently serve the automation and security needs of homes and businesses. Designed to send and receive data with remote logging servers, or to be remotely controllable by a user's smartphone or web application software, these devices communicate over IP networks with or without requiring a human to be in the loop. While this offers increased consumer convenience and control, it also opens the door to new avenues of cyberattack, as the domain of potential targets grows and there is less human monitorization.

Much of this growth is attributed to consumer "Internet Connected Things" which include household appliances, security cameras, televisions, digital video recorders, and smart electric meters [46]. These devices are broadly connected to the internet through normal IP networks and thus can send/receive traffic to/from any device reachable from the internet. As of 2017, few of these devices possess adequate security measures, and thus are subject to infiltration [3]. Infiltrated or compromised devices can continue to provide expected consumer function and generally do not give any indication to the consumer that they have been compromised. Since these devices have the ability to connect to any remote internet endpoint and are very unlikely to be discovered once they are, internet connected things perfect candidates for inclusion into networks of malware infected devices (botnets).

At the other end of the spectrum, cloud computing has evolved as a resource sharing computing model in which large service providers provide infrastructre (infrastructure as a service), platforms, (platform as a service), software (software as a service), and everything in between as leasable services. These services are profitable for the service providers once a sufficient level of resource utilization is reached, and this gives rise to multiple customers sharing resource space at the CPU, storage, and network levels. This is known as multi-tenancy. This availability has driven a centralization of enterprise class computing into the cloud computing model, which offers convenience and quality for businesses. However, this also entails new challenges as cloud providers contend with the technical and security issues of multi-tenancy, which open up novel cyberattack methods as potential attackers may now reside within the same server or even the same physical CPU as their intended victims.

A common and problematic cyberattack is the denial-of-service attack. It is characterized by an attempt by a remote device to send more traffic to a target system than that system is able to handle. Once this threshold capacity is reached, normal/benign traffic can no longer be serviced, and the target becomes unavailable for use. The security community has been able to create effective firewall and filtering techniques to mitigate such DoS attacks, a state of affairs which then caused the attacks to evolve into distributed-denial-of-service (DDoS) attacks. DDoS attacks achieve the same end as their DoS ancestors, but do so by overwhelming the target with requests from a myriad of source devices called *bots*. In this way, the true source of the attack is very difficult to detect, as each attacking device does not send enough individual traffic to raise firewall/ filtering alarms. The effect, however, is the same: the summation of all the botnet traffic is enough to render the target resource unavailable. Effective protection against DDoS attacks remains an open problem in the security community.

To contend with the technical challenges of resource sharing, most cloud providers have developed concepts of software-defined networking (SDN) and network virtualization. In contrast with previous packet or circuit switching networks, software defined networks allow for software management of network traffic. This is enabled by centrally gathering network traffic information from a

Langkawi'18, January 2018, Langkawi, Malaysia
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

management network of distributed devices capable of monitoring and reporting traffic, then making intelligent decisions that lead to the highest resource utilization in the given situation. Because these networking models are associated with increased traffic metadata, they are well suited to tackling the problems of cyberattack[47].

This project proposes a system for recognizing and mitigating DDoS attacks in progress. The system's distributed detection and intervention strategies are bio-inspired and take inspiration from the function of human immune systems in detecting and disrupting unknown pathogens. Like its biological counterpart, this AIS is designed to be highly adaptive by using the distributed monitoring approach afforded by SDN systems with the malicious pattern generation techniques of a modified artificial immune system. The system is shown to be capable of being highly effective in monitoring, detecting, and mitigating unknown DDoS attacks in progress.

1.1 Proposed System

This work proposes a network-based[38] SOM-modified-AIS OpenFlow network application to meet the primary challenges of effective DDoS detection and mitigation in cloud computing environments. The system features collaborative profiling to meet the distributed monitorization needs of such environments, a time-series toxicity "overwhelm-factor" to model the effect a DDoS attack has on network terminal equipment, a network capable of simulating a DDoS attack in an SDN network environment, and a SOM-modified AIS network application to handle attack detection and the immune response. As a proof of concept, this work implements the SDN DDoS network, collaborative profiling monitorization scheme, and toxicity modeling AIS application while assuming a functional SOM block.

2 RELATED WORK

This section explores related work in developing SDN network-based[38] or AIS intrusion detection systems (IDSs).

The work of Jha and Archaya [24] builds an AIS-based IDS featuring network-based[38] monitorization and unsupervised learning they call I3DS. I3DS is built around the function of two modeling techniques, one probabilistic-based and one decision tree-based. The probabilistic technique uses a Hidden Markov Model[12] structure to adaptively learn and provide a preliminary self vs non-self classification on novel traffic. The decision tree structure then takes the non-self candidate data and performs a final self non-self classification. I3DS is then compared against other learning-based IDS techniques (k-means, SOM, etc.) in testing utilizing the KDD Cup 99 Training Dataset [43] from [44]. I3DS outperforms all competition. Though I3DS is purely theoretical, and thus does not offer a network monitoring scheme, it should be the goal of this work's SOM module to outperform I3DS in the same test conditions.

Hong [22] develops a network-based[38] malware detection method that detects anomalies in Domain Name System (DNS) traffic, rather than IP application traffic. The method (DTA) uses captured DNS traffic to build two graphs: DNS Lookup and DNS Failure. DNS Lookup is a weighted bipartite graph [16] whose edge set provides a mapping from a domain name to its corresponding IP address, and logs the query's frequency for a given time period. As DNS Lookup can only contain information on successful queries, DNS Failure is created to provide a mapping between failed DNS queries and the hosts originating the query, where the edge weight represents its frequency. Once created, the graphs are monitored for presentation of well-defined graph artifacts that are known to correspond with malicious network activity. However, graph presentation of these artifacts alone is insufficient to complete self non-self classification, so a decision tree module is included. While this method offers promise, it is a purely theoretical model and does not offer any kind of network monitoring scheme. DTA should ultimately be outperformed by the work of this project.

Braga[14] builds an OpenFlow-based DDoS detection method utilizing the SOM technique. Their SOM is trained using a 6-tuple of native OF switch statistics gathered from the NOX[10] SDN controller. While effective, [14] does not allow for communication between multiple networks nor statistical methods which analyze OF switch ports to delineate attacking hosts from benign hosts. Further, this paper does not address the effects that DDoS attacks have on SDN controllers themselves.

3 CHALLENGES AND HYPOTHESIS

This chapter articulates the general challenges associated with designing a system capable of detecting DDoS attacks in cloud computing environments, and the specific challenges in developing and implementing the primary work of the design. It concludes with a statement of the hypothesis the project intends to affirm.

3.1 Primary Challenges

The primary challenges are those identified as the basic and essential tasks a general system must perform in order to protect a cloud network from a DDoS attack.

3.1.1 Monitorization. Sufficiently monitoring network behavior is the first challenge to be met. The network monitoring scheme is the ultimate source of all information from which AIS decisions are made, so no data down the data pipeline may be of a finer resolution than the monitoring scheme itself supports. Further, the more network traffic monitoring data that can be generated, the more applicable the resulting model represents the network. As cloud virtual networks become more complex and more distributed, no single network device has a global network view, and destination-based monitoring becomes infeasible. Instead, network-based[38] monitoring methods composed of multiple distributed monitoring agents must be implemented.

3.1.2 Detection. To effectively secure a system from attack, the security system must be able to accurately and precisely predict when an attack is going to take place, or detect an attack in progress before damage is done. Due to the imperfect nature of prediction and detection, an amount of error will invariably exist. There are four types of detection events[24][22]:

(1) True Positive (T^+)– the system detected an event that occurred
(2) False Positive (F^+)– the system detected an event that did not occur
(3) True Negative (T^-)– the system did not detect an event that did not occur

(4) False Negative (F^-)– the system did not detect an event that occurred

The error types are related in a trade-off scheme. To achieve a higher T^+ rate, a system would likely experience an increase in F^+ rate as well. The situation is the same for F^+ and F^- rates. Based on the type of network traffic, some detection applications will be more tolerant of certain error types than others. For example, a government agency internal network may have access to sensitive data. Such an environment would have a very low tolerance for F^- type errors, with the understanding that an increase in F^+ type error tolerance would be necessary. Conversely, a public media server would seek to minimize F^+ errors, and would likely have a higher tolerance for F^- errors. Due to this reality, the detection module should be configurable such that it can be tuned to the performance requirements of the specific application implementation environment.

3.1.3 Intervention. An effective system must be able to take action against an identified threat. The action taken must be able to stop or mitigate the harmful effects of the attack and should provide some sort of attack memory should a similar attack occur in the future. Given a distributed environment, the action taken should include a broadcast of the attack signature to potentially affected network devices such that other network devcies may stop or mitigate the attack. The attack mitigation technique should maximize effect on the targeted threat while minimizing the disruption to normal (benign) traffic.

3.2 Hypothesis

This paper aims to meet all core challenges by developing a self organizing map (SOM) modified AIS OpenFlow network application with a monitorization module utilizing native SDN features in OpenFlow v1.3 to both monitor and intervene. This work hypothesizes that an SDN environment with OFv1.3 compliant switches is capable of monitoring a cloud environment's network to a level sufficient to enable a SOM based AIS network application to detect a DDoS attack in progress. The AIS module resides on top of the OF network controller as a network application. Upon successful detection, the AIS network application defines a new network policy capable of mitigating the attack and then instructs the OF controller to enforce the attack mitigation policy. Native OFv1.3 switches are sufficient to execute the monitorization scheme and mitigation policy without custom development.

4 PROPOSAL

This work seeks to support its hypothesis by building a proof-of-concept system. This system is intended to demonstrate the effectiveness of an OpenFlow v1.3 based network application in monitoring network traffic and detecting DDoS attacks in progress, assuming a suitably accurate detection signature is provided by the SOM module.

4.1 Collaborative Profiling

This project leverages an OpenFlow v1.3 switched network to build its network-based detection scheme. In contrast to traditional ingress switch or destination-based methods, network-based methods compile traffic data from many distributed traffic monitors. To take advantage of OpenFlow's strength in centrally handling distributed network architectures, it proposes the monitorization scheme be based on OpenFlow flow statistics collection native to the protocol.

The proposed framework develops a collaborative traffic monitoring model for attack detection, shown in figure 1. In it, each OF switch features a profile extractor that collects flow statistics for a subset of the network traffic flowing through the switch. The profile extractor is implemented as a low table ID, high priority flow table entry, and does not perturb any network traffic flowing through the switch. The OF switch collects the desired traffic data according to native OpenFlow matching rules on fields given in Section 4.3.2 on page 5. Periodically, a profile aggregation network application instructs the SDN controller to collect the captured data from each switch, then aggregates the data to obtain a global network view of network traffic. In this way, network-based monitorization is achieved and the monitorization primary challenge is met.

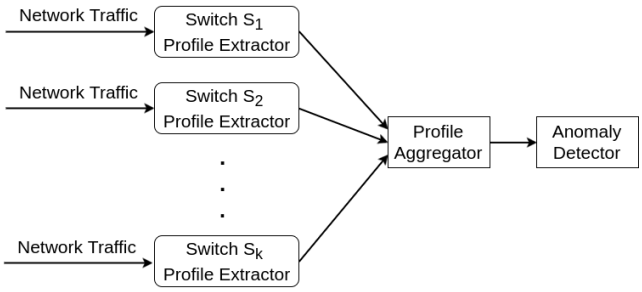

Figure 1: Collaborative traffic analysis model.

4.2 Toxicity

The nature of DDoS attacks is to overwhelm a computing resource. If a sufficient amount of traffic arrives at the attack target within a defined period of time, the destination becomes overwhelmed and inaccessible. This concept of time is very important, as 10Mb of network traffic in ten seconds is a very different task to handle than 10Mb in one millisecond. Native OpenFlow flow statistics are able to give precise packet counts, byte counts, and even average bitrates. While these figures may be able to give a good instantaneous snapshot of network traffic, they alone are not enough to track the overwhelming effects of time sustained abuse. To track a flow's capacity to overwhelm, we define a function of time: toxicity.

A single packet traveling through a network requires a non-zero amount of resources in order to be processed by network devices including switches, wires, routers, and destination servers. This amount varies based on packet size, L2, L3, and L4 protocols, etc, and may be quite small per packet, but can accumulate to an overwhelming amount very quickly. It becomes very useful to think of a packet as containing a fundamental amount of inherent toxicity. As more packets travel through a network en route to a final destination server, the general toxicity of the network increases. As more requests arrive at a server than it can handle per unit time, the toxicity level accumulates at a rate proportional to the number

and type of packets it is receiving. If traffic increases enough, the destination server becomes overwhelmed and becomes unreachable, an event analogous to poisoning. As network traffic subsides, the toxic effects subside as a function of time as the server is able to clear its queue. In seeking to protect a resource from attack, it becomes useful to think about the problem as trying to keep the resource's toxicity level from reaching a poisonous threshold. As general existing IP and SDN monitoring mechanisms are effective in keeping network infrastructure from being overwhelmed, our focus is on protecting the destination servers themselves.

Toxicity is modeled by considering how destination servers receive and process service requests. The amount of fundamental toxicity a flow accumulates with each packet reception is related to the server's maximum queue size, the packet size, and the L4 protocol, but not to the current size of its buffer or queue. Therefore, toxicity increases with each event. Further, toxicity levels are kept for both packet statistics and byte statistics, and thus we define the value τ_k, where k can take the value P for packet toxicity and B byte toxicity as,

$$\tau_k(t) = \tau_k(t - \Delta t) + \delta_k b_k(t) \tag{1}$$

where $\tau_k(t)$ is the toxicity level at time t, Δt is the elapsed time since the last toxicity update, δ_k is a tuneable damping constant, and $b_k(t)$ is the fundamental toxicity deposit amount for this flow. The deposit equations are given below.

$$b_k(t) = \alpha \frac{\eta_{k,t} - \eta_{k,t-\Delta t}}{\Delta t} \tag{2}$$

where α is a configurable constant and η is the number of packets or bytes.

Toxicity dissipates at a rate relative to the number of packets in the queue left to be processed and the time to service one request. Since the rate is relative to its current value, this process is modeled by the differential equation,

$$\frac{d\tau}{dt} = -\lambda\tau \tag{3}$$

with solution,

$$\tau(t) = \tau_0 e^{-\lambda t} \tag{4}$$

where λ is the exponential decay constant. Combining increasing and decreasing factors gives the composite function for a flow's toxicity as a function of time,

$$\tau_k(t) = \tau_k(t_0)e^{-\lambda(t-t_0)} + b_k(t) \tag{5}$$

where t_0 is the time of the most recent update. In this way, an accurate, configurable model capable of capturing the ability of a flow to overwhelm a destination server is developed.

4.3 System Architecture

The proposed system architecture is composed of an OF v1.3 SDN NS-3 network, a Floodlight SDN controller, and an AIS network application module. The OF v1.3 network simulates the behavior of a UDP flood DDoS attack on a single target, and includes normal background UDP traffic. The NS-3 network features ofswitch13 [15] OF v1.3 switches that are externally controlled by the Floodlight SDN controller [5] running in the same Linux user space as the NS-3 simulation. The NS-3 Floodlight interface is made through a Linux TAP interface [29], whose L2 and L3 addresses NS-3 associates with the controller through its native *ns3::TapBridge* module. The AIS network application module interfaces with the Floodlight controller via HyperText Transfer Protocol (HTTP) REpresentation State Transfer (REST) API to monitor the network and detect DDoS attacks in progress.

The AIS process is given in figure 2. The process is split into two logical entities, the central coordinator is composed of the network controller and network application (including SOM module), while the distributed detectors are the OF v1.3 switches. Potential detectors are initially generated by analysis of an initial dataset, and continually generated by analysis of network traffic. These potential detectors compose the self set. The self set undergoes the negative selection training process to become the antibody set, which is then distributed to the appropriate distributed detector(s). The detectors then collect flow statistics on incoming traffic, and match flow patterns against their antibody subsets. Flows that do not match antibody patterns are processed normally according to OpenFlow v1.3 processes, and flow statistics are periodically gathered and sent to the central coordinator for continued adaptive learning. Upon discovering a flow that does match an antibody, the detector alerts the central coordinator and the expressed antibody is added to the pathogen library.

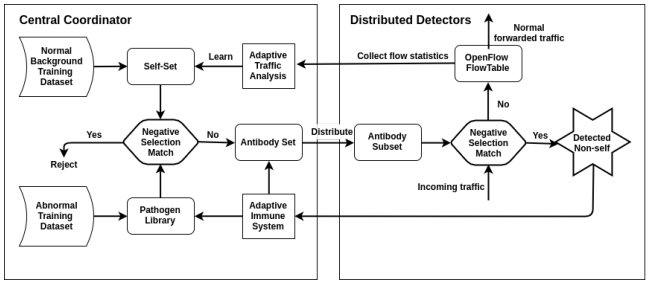

Figure 2: AIS process.

4.3.1 Network Architecture. The NS-3 network built for this project simulates the behavior of a UDP flood type DDoS attack converging on a single target. The network features ofswitch13[15] switches externally controlled by the Floodlight SDN controller. In keeping with general SDN practices, the data and control planes are logically separated into independent networks. This type of network in which control signaling is carried in a different network than user data is called an out-of-band network [34]. Out-of-band networks, while good general SDN practice, afford the additional benefit of control plane insulation against application-level DDoS attacks. An entire in-band control network could easily be taken down by an application-level DDoS attack, as the OF control messages would be dropped due to attack congestion. The control network, also called the OpenFlow channel, is configured as a single Carrier Sense Multiple Access (CSMA) ethernet [6] channel. The OF switch L2 and L3 addresses are configured within the NS-3 simulation, while the controller L2 and l3 addresses are configured by the Linux TAP interface, external to the NS-3 environment. Any changes to the OpenFlow channel topology must be reflected in both the TAP interface and the NS-3 environment.

The data plane network topology is contained entirely within the NS-3 environment. In it, each host is connected to a single switch by a dedicated CSMA ethernet channel. The dedicated channel allows for high traffic monitor resolution, suitable for this proof of concept system. The host switches are connected to the target switch in a star topology, again with dedicated ethernet channels. The target switch then uses singly dedicated ethernet channels to connect with the destination servers.

4.3.2 AIS Network Application Data Structure. The AIS Network Application is intended to enable decisions to be made on large networks which generate encumberingly large volumes of network statistics (metadata). As the entirety of this metadata must travel over the limited OpenFlow channel, care must be taken to minimize the channel bandwidth utilization by the module, or risk interfering with normal network operations. Further, as network, memory, and CPU resources are marketable commodities in cloud computing environments, the application's data and processing considerations must also be taken into account. For the end system, every effort should be made to reduce resource consumption wherever possible. As a result, the proof-of-concept AisTable and associated entries are developed with this in mind. The structure of the an entry in the AisTable is composed of the fields: *Signature*, *SwitchCountsEntry*, *Packet Tox*, and *Byte Tox*.

The AisTable is a hash table [16], keyed on a hashing of the *Signature*. Each network profile (see section 4.1) of interest generates a unique Signature and thus a unique AisTableEntry in the AisTable. Table 1 gives the possible fields a Signature might contain. These fields are developed from the OXM match fields required by OpenFlow v1.3 specification [41], and represent the maximum view resolution the AIS system has into the network. A Signature instance may contain as many Signature fields as desired, but must contain at least one.

Table 1: Signature fields (required OXM match fields).

Field	Description
OXM_OF_IN_PORT	Ingress port.
OXM_OF_ETH_DST	Ethernet destination address.
OXM_OF_ETH_SRC	Ethernet source address.
OXM_OF_ETH_TYPE	Packet payload ethernet type.
OXM_OF_IP_PROTO	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	IPv4 source address.
OXM_OF_IPV4_DST	IPv4 destination address.
OXM_OF_IPV6_SRC	IPv6 source address.
OXM_OF_IPV6_DST	IPv6 destination address.
OXM_OF_TCP_SRC	TCP source port.
OXM_OF_TCP_DST	TCP destination port.
OXM_OF_UDP_SRC	UDP source port.
OXM_OF_UDP_DST	UDP destination port.

Due to the collaborative nature of the network-based monitorization scheme, a single Signature may receive traffic data contributions from separate switches throughout the network. To allow for this, the AisTableEntry includes an array of SwitchCountsEntries, with a unique SwitchCountsEntry belonging to a single switch for the given profile. The SwitchCountsEntry keeps track of packet and byte counts for the individual flow, and the timestamp when these network statistics were last collected. These data are necessary and sufficient to calculate the toxicity contributions per time aggregation event as part of the calculation of Equation 5. The structure of the SwitchCountsEntry is composed of the fields: *SwitchID*, *Timestamp*, *Packet Count*, and *Byte Count*.

5 EXPERIMENT

The experimentation is designed to support the project's hypothesis. Included are descriptions of the experiment's topology, and specific limiting implementation challenges. Further, experiment results and analysis are presented that demonstrate support of the hypothesis.

5.1 Description

The project experimentation is designed to demonstrate the AIS application's nature as a network-based AIS and its capabilities of meeting the monitorization and detection primary tasks. To accomplish these, experimentation is built on the integration of several software modules within a single Linux userspace, as described in chapter 4. The experimentation consists of a series of separate 900 second network traffic simulations in which network configuration and traffic patterns are generated by the NS-3 network as input, and the AIS module toxicity and hazard message responses are recorded as output. The project hypothesis is supported if:

(1) *Monitorization*: the AIS is able to aggregate the distributed profile information into a network-based global view of traffic.
(2) *Detection*: the AIS is able to outperform other detection methods for a sufficiently realistic simulated DDoS attack in the measured performance indicator metrics.

The NS-3 component gives the ability to simulate representatively realistic network traffic in order to test network configurations, external SDN controllers, and SDN network applications. To test the monitorization and detection capabilities of the AIS network application, an NS-3 realtime network is created. The experimental traffic scenario is then run.

5.1.1 Profile Gathering Experiment Description. The experiment feeds a realistic traffic pattern to the system. In this experiment, a second destination is created, and is not the subject of an attack. Normal application UDP traffic flows from even numbered IP hosts to both the benign destination server and to the attack target. This traffic serves as background noise for the DDoS detection module, and should be neither detected as attack traffic nor disrupted as part of any attack mitigation efforts. This normal application traffic starts shortly after the simulation begins, and steadily continues through the simulation. Attack application UDP flood traffic flows from odd numbered IP hosts to the attack target. As all host clients generate the same amount of individual network traffic, no single IP source address can be categorized as malicious. These flows come online in regular intervals, and reach warning and alert level thresholds at well defined times. It is the end goal of the project to be able to classify subflows within the detected flows, and filter out specific IP sources as a result. It is this ability that the SOM module is intended to provide, and thus is not required here.

However, a demonstration of the subflow classification behavior is possible before the SOM module is developed. This behavior is available through the concept of bit-mapping. OpenFlow v1.3 flow tables support arbitrarily bit-masked IP address match fields[41]. This allows flow table entries to match IP source addresses to specific bit patterns, rather than specific or wildcarded full addresses. These bit patterns can be generated randomly, and can go through a negative-selection training process in a "pseudo-SOM" module, with the bitmasks taking the role of detector. In following along the SOM AIS process, the selected bitmasks can be given to the AIS module. Once there, the AIS module can instruct the SDN controller to monitor IP addresses matching the bit pattern, rather than monitoring specific IP addresses or subnetworks. In this experiment, the pseudo-SOM module has generated such a bitmask, and the results of its use are given in section 5.2.1.

5.2 Results

Results are now given which lend support to the hypothesis that a network-based[38] AIS is capable of detecting and mitigating DDoS attacks in cloud computing environments. These results center around the detection performance indicators[24][22] detection rate (D), accuracy (A), precision (P), and F-score (F). Where,

$$D = \frac{T^+}{T^+ + F^-} \tag{6}$$

$$P = \frac{T^+}{T^+ + F^+} \tag{7}$$

$$A = \frac{T^+ + T^-}{T^+ + T^- + F^+ + F^-} \tag{8}$$

$$F = 2 \times \frac{D \times P}{D + P} = \frac{2 \times T^+}{2 \times T^+ + F^+ + F^-} \tag{9}$$

$$\tag{10}$$

D represents the proportion of malicious packets the system is able to detect to all of the malicious packets in the system. Precision captures the likelihood that a packet detected as malicious truly is malicious. Accuracy represents the ability of the system to properly classify a packet as benign or malicious, and the F-score is a composite score that is used as a single "goodness metric". These performance metrics are composed of the four detection event types of section 3.1.2.

5.2.1 Profile Gathering Experiment. The profile gathering experiment features the same network-based[38] AIS, and largely the same network configurations as the baseline gathering experiment. However, it differs in that it includes a benign destination server. This new server receives a steady amount of benign UDP application traffic throughout the simulation from each of its hosts. The benign destination has the IP address 10.1.1.11, while the attack target has the IP address 10.1.1.10. This experiment examines the performance of two detection profiles on this new traffic scenario.

The first, WILDCARD, offers the simple network-based[38] detection featured in the previous experiment. Its IP source address field is wildcarded, resulting in this profile being unable to classify subflows or monitor specific IP source addresses. The second, BITMASK, offers the pseudo-SOM bitmask concept discussed in section 5.1.1, and is expected to outperform WILDCARD. The AIS is seeded with these detection profiles in turn and identical network simulations are run on each.

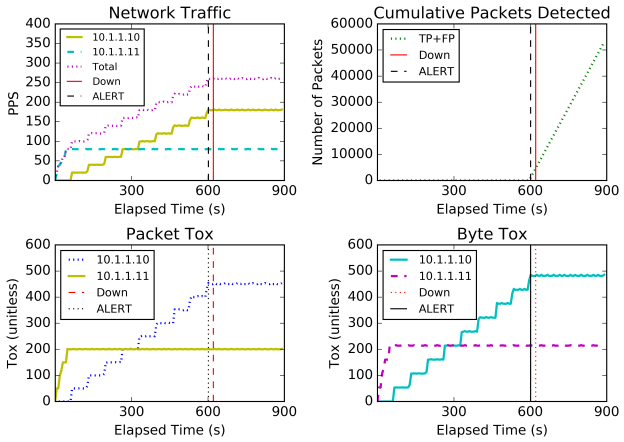

Figure 3: WILDCARD Single profile without bitmasking.

Figure 3 gives the details of the WILDCARD profile experiment. Total network traffic is shown, as is a breakdown of traffic by destination, at the furthest resolution offered by the profile. Each plot in the figure features ALERT and down lines, which illustrate the time at which the AIS detected an incipient attack and the time at which the target server was taken down by the attack. The accumulated total number of positive detection events is also shown. Packet and toxicity curves are given next, with the general patterns closely following the input traffic patterns.

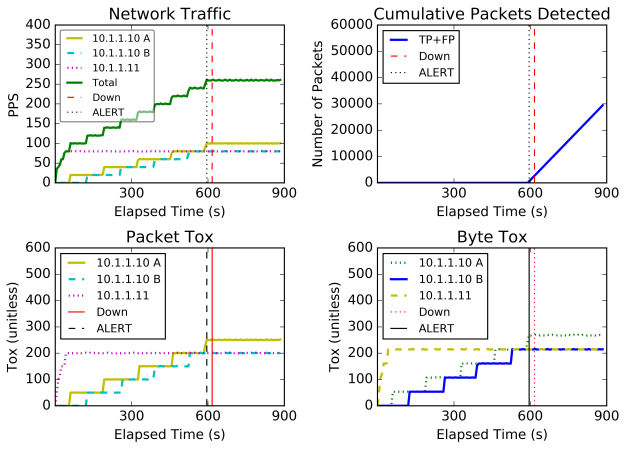

Figure 4: BITMASK Profile gathering experiment results with pseudo-SOM supplied profile bitmask.

The results of the BITMASK profile are given in figures 4 and 5. It runs an identical NS-3 network configuration, so the network input traffic is the same. In the same method as the previous, this traffic

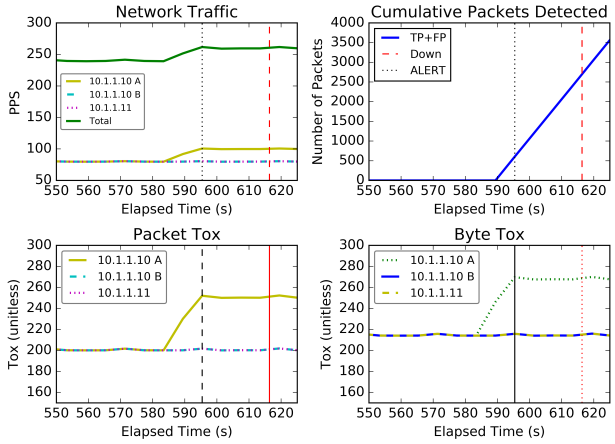

Figure 5: BITMASK Profile gathering experiment results with pseudo-SOM supplied profile bitmask, detail.

is broken down by destination at the furthest resolution offered by the profile. It should be noticed that BITMASK is able to distinguish subflows within the 10.1.1.10 flow: 10.1.1.10 A (attack) and 10.1.1.10 B (benign). The accumulated packet detections are given as well, and when compared against WILDCARD's result, show a more shallow linear slope as a result of its fewer detection events.

The toxicity figures illustrate the real benefit of this model. The benign subflow on the attack target does not surpass the alert threshold, and thus is not detected as malicious. The attack subflow, however, does, as is detected as malicious a result.

Table 2: Profile gathering performance indicators

	D	P	A	F
WILDCARD	96.329	55.555	85.902	70.469
BITMASK	96.452	100.0	99.381	98.194

This experiment's performance indicator results are given in table 2. As can be seen, the number of T^+ events is the same (the 6 packet difference is negligible) for the two detection profiles. The F^- events with both profiles are associated with the 10 second detection time lag as a result of the AIS's polling period and toxicity deposit amount configuration. The great benefit of the BITMASK profile lies in the number of F^+ events. WILDCARD's wildcarded IP source field results in over 23000 F^+ events, giving a precision of only 56%. This inadequate performance is expected here, as it cannot distinguish subflows with such a wildcard when traffic levels per host are identical. Thus, WILDCARD cannot distinguish an attacking host from a benign host within a flow. BITMASK, however, can, thanks to its inclusion of the bitmasking pseudo-SOM. As a result, it creates 0 F^+ events and properly classifies WILDCARD's 23000 F^+s as T^-. This proper classification gives BITMASK perfect precision and an accuracy limited only by the polling period, greatly supporting the project's hypothesis.

Other works by Hong [22] and Jha and Archaya [24] have measured the same metrics for similar work, which is compared in table 3. The highest obtained F-scores are included here for comparison. It should be noted that the experiments vary here, and therefore the results of comparison should be taken lightly.

Table 3: Related Work Performance Comparison

IDS	F-Score
Ais::BITMASK	98.2
DTA[22]	90.3
I3DS[24]	97.1

6 CONCLUSIONS AND FUTURE WORK

This work is intended to function as the laying of a foundational proof of concept for the proposed AIS network application system described in section 4. As demonstrated in section 5.2.1, when given a perfect profile, the AIS system can express perfect behavior, and thus is limited only by the strength of its detector generation method. Therefore, the hypothesis of this experimental proof of concept is well supported and should be extended.

The primary area of focus for future work should be on the development of the SOM module for the AIS network application. The development of this module would allow many of this project's assumptions to be removed and would allow the AIS to fully function as it was intended. The AIS network application is fully ready to accept network profiles from such an SOM module, so this development can focus on the SOM itself. The implementation details of the SOM module are not constrained by the AIS network application, and thus the SOM module can be implemented using any tool most suited to the task. The SOM generated profiles can be passed to the AIS network application through any usual application communication means, such as interprocess communication (IPC) [31]. If the developers of the SOM module choose to implement it in Python, minimal integration effort will be required, and the SOM can simply function as a separate thread/ process within the application.

Though the development of the SOM module will contribute mightily to the overall goals of the project, challenges remain that will limit its efficacy and accuracy. These challenges should be co-developed with the SOM module, as they are not logically connected and can be developed in parallel. These challenges include coordinating with the developers of the ofswitch13 module [15] to refine ns3::CsmaNetDevice modeling to solve the L3 Subnet Problem. Rectifying this problem will allow for a much more complex and realistic SDN network to be developed in NS-3. Not only will this result in more applicable real-world simulations, but it will also provide for a better training environment for the SOM module. It is possible the developers of the module will solve this problem themselves, but it is not guaranteed.

The final workstream to be extended in the future is the development of realistic and complete NS-3 service request modeling. The development of realistic μ values for destination terminal equipment packet service modeling would allow for the desired real-world effects of DDoS attacks on application servers to be modeled in NS-3 and a high analyzation resolution would result. This modeling must be able to account for load on the server, packet protocol, available server memory, etc. This sort of analyzation will

be necessary before any potential AIS system is fully tested, so future developers will need this capability in testing.

Future parallel development of these three workstreams is recommended, as the results of this paper demonstrate that a network-based AIS SDN network application is capable of detecting and mitigating DDoS attacks in progress in cloud computing environments.

REFERENCES

[1] 2016. *The Open Group Base Specifications Issue 7.* Technical Report. The Open Group and IEEE.

[2] 2017. Accessed: 6/24/2017. *What is NS-3.* http://www.nsnam.org

[3] 2017. Accessed: 6/25/2017. *2017 Study on Mobile IoT Application Security.* https://media.scmagazine.com/documents/282/2017_study_mobile_and_iot_70394.pdf

[4] 2017. Accessed: 6/25/2017. *Breaking Down Mirai: An IoT DDoS Botnet Analysis.* Technical Report. https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

[5] 2017. Accessed: 6/25/2017. *Floodlight SDN Controller.* http://www.projectfloodlight.org/floodlight/

[6] 2017. Accessed 6/25/2017. *IEEE 802.3 Ethernet Working Group.* http://www.ieee802.org/3/

[7] 2017. Accessed: 6/25/2017). *Python Programming Language.* https://www.python.org/

[8] 2017. Accessed: 7/1/2017. *ns3::CsmaNetDevice Class Reference.* https://www.nsnam.org/doxygen/classns3_1_1_csma_net_device.html

[9] 2017. Accessed: 7/4/2017. *Mininet: An Instant Virtual Network on your Laptop (or other PC).* http://mininet.org/

[10] 2017. Accessed: 7/4/2017. *NOX Network Control Platform.* https://github.com/noxrepo/nox

[11] 2017. Accessed: 7/4/2017. *The POX Controller.* https://github.com/noxrepo/pox

[12] Leonard E. Baum and Ted Petrie. 1966. Accessed 6/25/2017. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Ann. Math. Statist.* 37, 6 (12 1966. Accessed 6/25/2017), 1554–1563. https://doi.org/10.1214/aoms/1177699147

[13] Tamara Bowman. 2001. Accessed: 7/1/2017. *UDP Flood Denial of Service.* Technical Report. Global Information Assurance Certification. https://www.giac.org/paper/gcih/206/udp-flood-denial-service/101057

[14] Rodrigo Braga, Edjard Mota, and Alexandre Passito. 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference.* IEEE, 408–415.

[15] Luciano Jerez Chaves. 2017. *OpenFlow 1.3 Module Documentation.* Technical Report. University of Campinas.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. . MIT Press. 256–285 pages.

[17] D. Crockford. 2006. *The application/json Media Type for JavaScript Object Notation (JSON).* Technical Report. Network Working Group.

[18] Wesley M Eddy. 2007. *TCP SYN flooding attacks and common mitigations.* Technical Report.

[19] Stephanie Forrest, Alan S Perelson, Lawrence Allen, and Rajesh Cherukuri. 1994. Self-nonself discrimination in a computer. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium.* Ieee, 202–212.

[20] Fabio A González and Dipankar Dasgupta. 2003. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines* 4, 4 (2003), 383–403.

[21] Steven A. Hofmeyr and Stephanie A. Forrest. 2000. Architecture for an Artificial Immune System. *Evolutionary Computation* 8 (2000), 443–473.

[22] L. V. Hong. 2012. *DNS Traffic Analysis for Network-based Malware Detection.* Master's thesis. Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark.

[23] Kai Hwang, Geoffry C. Fox, and Jack J. Dongarra. 2012. *Distributed System Models and Enabling Technologies.* Morgan Kaufman. 5–30 pages.

[24] Manjari Jha and Raj Acharya. 2016. An immune inspired unsupervised intrusion detection system for detection of novel attacks. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference.* IEEE, 292–297.

[25] Zhou Ji and Dipankar Dasgupta. 2004. Real-valued negative selection algorithm with variable-sized detectors. In *Genetic and Evolutionary Computation–GECCO 2004.* Springer, 287–298.

[26] Zhou Ji and Dipankar Dasgupta. 2009. V-detector: An efficient negative selection algorithm with âĂIJprobably adequateâĂİ detector coverage. *Information sciences* 179, 10 (2009), 1390–1406.

[27] Jungwon Kim and Peter J Bentley. 2001. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation.* Morgan Kaufmann Publishers Inc., 1330–1337.

[28] Teuvo Kohonen. 1998. The self-organizing map. *Neurocomputing* 21, 1 (1998), 1–6.

[29] Maxim Krasnyansky. 2000. *Universal TAP/TUN Device Driver.* Technical Report. The Linux Foundation.

[30] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2015), 14–76.

[31] Leslie Lamport. 1986. On interprocess communication. *Distributed computing* 1, 2 (1986), 86–101.

[32] Avraham Leff and James T Rayfield. 2001. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International.* IEEE, 118–127.

[33] Alberto Leon-Garcia and Indira Widjaja. 2004. *Delay and Loss Performance.* McGraw-Hill. 845–859 pages.

[34] Alberto Leon-Garcia and Indira Widjaja. 2004. *End-to-End Digital Services.* McGraw-Hill. 249–250 pages.

[35] Sharon Lim, J Ha, H Kim, Y Kim, and S Yang. 2014. A SDN-oriented DDoS blocking scheme for botnet-based attacks. In *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conference.* IEEE, 63–68.

[36] Seyed Mohammad Mousavi and Marc St-Hilaire. 2015. Early detection of DDoS attacks against SDN controllers. In *Computing, Networking and Communications (ICNC), 2015 International Conference.* IEEE, 77–81.

[37] Jon Postel. 1981. Accessed: 6/25/2017. *Transmission Control Protocol.* https://doi.org/10.17487/RFC0793

[38] Orin Reams. 2005. Accessed: 6/27/2017. Distributed denial of service (DDoS) network-based detection. (2005. Accessed: 6/27/2017). https://www.google.com/patents/US20070130619

[39] Praneet Saurabh and Bhupendra Verma. 2016. An efficient proactive artificial immune system based anomaly detection and prevention system. *Expert Systems with Applications* 60 (2016), 311–320.

[40] D Senie and P Ferguson. 1998. *Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing.* Technical Report.

[41] OpenFlow Switch Specification. 2013. *Version 1.3. 2 (Wire Protocol 0x04).* Technical Report.

[42] William Stallings and Lawrie Brown. 2008. *Denial of Service.* Pearson Education Inc. 259–270 pages.

[43] Salvatore J Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K Chan. 1999. Accessed: 7/7/2017. *KDD Cup 1999 Data.* http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[44] Salvatore J Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K Chan. 2000. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, Vol. 2. IEEE, 130–144.

[45] J. Touch. 1997. *TCP Control Block Interdependence.* Technical Report. Network Working Group.

[46] Rob Van Der Meulen. 2017. Accessed: 6/26/2017. *Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016.* http://www.gartner.com/newsroom/id/3598917

[47] Qiao Yan, F Richard Yu, Qingxiang Gong, and Jianqiang Li. 2016. 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* 18, 1 (2016), 602–622.

[48] Susan Young and Dave Aitel. 2004. *The Hacker's Handbook.* Auerbach. 103–119 pages.