CAAMP: Completely Automated DDoS Attack Mitigation Platform in Hybrid Clouds

Nasim Beigi-Mohammadi, Cornel Barna, Mark Shtern, Hamzeh Khazaei and Marin Litoiu
Department of Computer Science,York University
Toronto, ON, Canada
Email:{nbm,cornel,hkh,mark,mlitoiu}@yorku.ca

***Abstract*—Distributed Denial of Service (DDoS) attacks are one of the main concerns for online service providers because of their impact on cost/revenue and reputation. This paper presents Completely Automated DDoS Attack Mitigation Platform (CAAMP), a novel platform to mitigate DDoS attacks on public cloud applications using capabilities of software defined infrastructure and network function virtualization techniques. When suspicious traffic is identified, CAAMP deploys a copy of the application's topology on-the-fly (a shark tank) on an isolated environment in a private cloud. It then creates a virtual network that will host the shark tank. Software defined networking (SDN) controller programs the virtual switches dynamically to redirect the suspicious traffic to the shark tank until final decision is made. If traffic is proved to be non-malicious, SDN controller installs flow rules on the switches to redirect the traffic back to the original application. Thus, CAAMP autonomically protects applications against potential DDoS threats and lowers the false positives associated with common detection mechanisms by leveraging resources from a private cloud.**

Keywords - DDoS Attacks, Hybrid Cloud, Software Defined Networking, Network Virtualization

I. Introduction

Denial of service attacks impose a serious threat on applications on cloud [1]. Although there are benefits to dynamic resource provisioning with pay-as-you-go billing in cloud, one disadvantage is the possibility of attacks designed to increase the costs without corresponding increase in the benefit of the application. Example of such attacks include Low-and-slow denial of service (LSDoS) attacks which are low-bandwidth application-layer DoS attack. The attacker is able to slowly degrade the performance of the application by exploiting weaknesses they have identified [2]. Distributed denial of service attacks (DDoS) is also one of the top nine threats to cloud computing environment according to Cloud Security Alliance [3]; out of all attacks in cloud environment, 14% are DoS attacks. As cloud is becoming more widespread, the rate of DDoS attacks is growing in parallel [4], [5]. DDoS attacks increase the workload on applications which would lead to adding more computational power to withstand the additional load. In addition, application owners are charged significantly for bandwidth usage caused by DDoS flooding raising the bill for cloud usage drastically.

Many companies use hybrid cloud deployment to allow them to scale computing requirements beyond their infrastructure capacity while still use their private cloud to reduce the cost. A hybrid cloud environment can also give rise to new opportunities to defend against DDoS attacks. Hence, in this paper, we introduce a mitigation solution to fully implement a defence strategy that makes use of private cloud to mitigate DDoS attacks targeting applications on public cloud.

We present **C**ompletely **A**utomated DDoS **A**ttack **M**itigation **P**latform (CAAMP)[1], a fully software defined solution that is cloud agnostic due to its virtualized and modular design. Resources from private cloud are used to handle suspicious traffic so that more time can be bought to distinguish attacks from non-malicious transient behavior. To achieve its objectives, CAAMP makes use of an autonomic manager that monitors the target application and initiates the mitigation process, SDN and network virtualization techniques.

When suspicious traffic is detected, CAAMP provisions a copy of the original application on a private cloud, referred to as *shark tank*, and dynamically redirects the suspicious traffic there. CAAMP protects stateless applications such as web applications where the transition to/from the shark tank does not interfere with user data. The shark tank provides the same functionalities as the original application, though with minimum amount of resources. The advantage of CAAMP to mitigate DDoS attacks is multi-folded; (1) DDoS attacks are hard to detect. Due to large false positives, non-malicious users may be filtered out as well [6]. Therefore, by using a shark tank, more time can be spent for further analysis over the suspicious traffic to make sure that the traffic is actually malicious and it is not some transient eccentric behavior. (2) By isolating the suspicious traffic on the shark tank, the original application is protected and serves only the traffic deemed legitimate. The users will experience normal behavior of the application and will not feel the negative effects of the attack. (3) The attacker that is sent to the shark tank has no indication that they have been discovered; they will experience high response time and the attack will look successful. As a consequence, they will not feel the need to adapt his strategy. (4) In case of adaptive applications, DDoS attacks may lead to excessive resources consumption (e.g., adding new virtual machines, increasing bandwidth, etc.) which will lead to higher costs. Therefore, by taking advantage of resources on a private cloud to host the shark tank, only the minimum amount of resources is allocated to handle suspicious traffic and reduce the associated costs.

Hence the original contributions of this paper are as fol-

[1]The name is inspired by Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA), whereas CAAMP helps to distinguish between malicious and non-malicious users by taking suspicious traffic to an isolated environment for further analysis.

lowing:

- We introduce and implement CAAMP which is an effective mitigation platform that protects applications against severe damages of DDoS attacks by taking advantages of virtualization and SDN technologies. CAAMP allows to have a more accurate detection consensus by providing more time for traffic analysis while maintaining service availability in case suspicious traffic is not malicious. CAAMP is cloud agnostic and can be applied in any cloud environment. Using the resources from an existing private cloud, CAAMP bears a cheap solution in isolating and identifying real attacks.
- We evaluate CAAMP on real hybrid cloud environment where Amazon is used as the public cloud and SAVI cloud [7] is leveraged as the private cloud. The results reveal promising functional and non-functional properties of CAAMP in real world scenarios.
- We performed extensive experiments to evaluate the efficiency and scalability of the SDN controller as the core component of CAAMP, for processing the mitigation commands. We particularly measure the response time of the SDN controller for processing mitigation requests. The results provide valuable insights in achieving instant mitigation actions with respect to SDN controller capacity.

The remainder of the paper is organized as follows: in Section II, we describe the architecture of CAAMP and explain its building blocks. Section III discusses the process of provisioning the shark tank. The design and architecture of SDN controller application is elaborated in Section IV. We then explain the experiment setup and present the results of our experiments on CAAMP in Section V. We overview the related work on DDoS attack mitigation in Section VI. Section VII concludes the paper and states future work.

II. CAAMP Architecture

A successful defense strategy against a DDoS attack has two major components: *detection* and *mitigation*. CAAMP is mainly a *mitigation* solution. However, for detection, CAAMP uses a number of sensors to detect DDoS threats. When DDoS threats are detected, CAAMP starts to mitigate the threats.

The mitigation mechanism makes use of the concept of *shark tank*. A *shark tank* is a functional copy of the application and application's topology that uses a minimum amount of resources, and handles the traffic identified as suspicious. While the application itself can reside in a public or private cloud, we assume that the shark tank is desired to reside in a private cloud. This way, the application owner has maximum amount of flexibility to deploy tools that further analyze the suspicious traffic, extract relevant attacking patterns, isolate and restore misidentified traffic. Also, because the *shark tank* has a reduced amount of resources at its disposal, the costs of mitigation are controlled.

Figure 1 shows an overview of CAAMP architecture. Main elements of CAAMP consist of target application, DDoS sensors, shark tank, an autonomic manager, shark tank orchestrator, and an SDN controller which will be described below.

Target Application: CAAMP protects the target application using a hybrid cloud environment where the target [2] application is hosted in a public cloud. Tailoring CAAMP to work on a pure public or private cloud would be a trivial task.

DDoS Sensors: The objective of this component is to detect the existence of DDoS threats. DDoS sensors are used on all traffic: *original application traffic* and *shark tank traffic*; in former they initially detect the suspicious traffic which leads to redirection to the shark tank, and in latter they determine if the suspicious traffic redirected to shark tank is actually malicious or not. If traffic is not malicious, it will be redirected back to the original application. In case of malicious traffic, two strategies can be taken: (1) the malicious traffic is kept in the shark tank so that the attacker feels that the attack has been successfully completed and as a result does not change its strategy, (2) simply block the attack.

Shark Tank: The purpose of a shark tank is to act as a restricted area for the attackers so that they can be placed under close surveillance. The shark tank absorbs the damage from the DDoS attacks (i.e., all suspicious traffic will be redirected to one instance of application in the shark tank) and allows the system to learn from these attacks for future reference. Moreover, the shark tank will be constantly monitored to investigate if the attacks are still taking place. The shark tank aids to reduce false positives, that is, if by mistake a user is redirected to the shark tank, they still get the requested services. Another reason for using a functional copy of the actual application on the shark tank is to prevent the shortcomings of the honeypot (the typical approach), that is, the malicious user might learn that they have been detected after they discover the honeypot has limited interactivity (simulation of the protected application). Although the same service is being offered, the amount of resources that will be used in the shark tank will be less than the original application.

Autonomic Manager: The autonomic manager uses a Monitor-Analyze-Plan-Execute (MAPE) loop [8] to monitor the protected application, plan and execute corrective actions so that desired operation conditions are met. For this purpose, CAAMP uses Hogna platform [9] to attain application inputs when creating the shark tank. In addition, Hogna monitors the application and adds or removes virtual machines (VMs) to meet the application criteria. For more detail on Hogna, reader is referred to [9].

SDN controller: It controls the routing of the traffic. Depending on the condition of the traffic, it programs the flow rules on the switches either to the target application or to the shark tank dynamically. The details and architecture of the SDN controller application is discussed in section IV.

Shark Tank Orchestrator (STO): It deploys the shark tank on private cloud for each application using *overlay networks*; this is one of the key points in making our solution cloud-agnostic. Particularly, STO employs Virtual Extensible LAN (VXLAN) technology which is a tunneling technology that can create arbitrary Layer 2 (L2) or Layer 3 (L3) networks by encapsulating L2 frames in L3 UDP packets. We employ Open Virtual Switch (OVS) [10] and OpenFlow initiatives to set up the overlay networks and program the flows respectively. Details of STO operation is discussed in Section III.

[2]We use target and original application interchangeably hereafter.

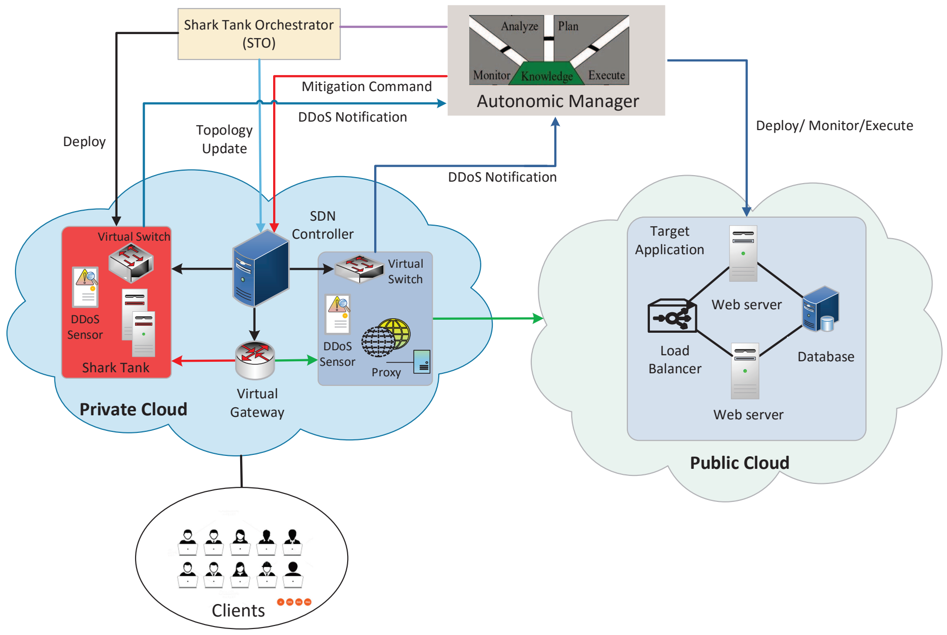


Figure 1: *CAAMP Architecture: The traffic first arrives at the gateway in private cloud. If it looks normal, it will be redirected to the public cloud through the application proxy. Otherwise, it will be redirected to the shark tank that is dynamically created on private cloud.*

Initially, CAAMP brings up a virtual switch that acts as an on-premise gateway on application owners' private cloud. The clients' traffic first reaches the gateway and then will be redirected to the public cloud through the application proxy (See Figure 1). DDoS sensors are installed on proxy to sniff the incoming traffic and detect suspicious activity. If suspicious traffic is detected, the DDoS sensors report the incidents to autonomic manager. The autonomic manager fires up the mitigation process resulting in the creation shark tank. It starts by instructing the STO to bring up the shark tank (i.e., the VMs), collects the IDs and IPs of the new VMs and configures them for their respective roles (i.e., database, web workers). Then, the autonomic manager will send the commands to the SDN controller so that the required rules are installed on the switches to dynamically redirect the incoming suspicious traffic to the shark tank. The autonomic manager also installs DDoS sensors on the shark tank so that they continuously monitor the suspicious traffic. When DDoS sensors determine that the suspicious traffic is not malicious and it has been some transient behavior, the traffic is sent back to the original application. In case of attacks, if the mitigation policy is to keep the attack in the shark tank, no further commands will be sent to the SDN controller. Otherwise, the autonomic manager sends commands to the SDN controller to block the attacks. Algorithm 1 summarizes the CAAMP DDoS mitigation actions.

Algorithm 1: The algorithm executed when a suspicious traffic is identified.

```
input  : $sources$ – a list of IPs that generate suspicious traffic
output : $\emptyset$
1 Deploy a minimal copy of the application's topology in the private
  cloud: $SharkTank$;
2 Connect all the new instances through a Virtual network: $virnet_{ST}$;
3 Deploy rules on the virtual switch to redirect all traffic coming from
  $sources$ to $SharkTank$ using $virnet_{ST}$;
```

III. Shark Tank Provisioning

STO creates the shark tank environment dynamically based on a *topology descriptor* file that is provided by the autonomic manager. The topology descriptor file determines the shark tank specifications including the number of VMs, corresponding images, and the topology.

STO uses virtual switches to build any type of topology over (L2) or (L3) networks. The image used to instantiate the VMs can be pre-configured with the required services. For example, a VM that is going to host the web application may use a pre-configured image in which a web server service such as Apache Tomcat has been already installed. When topology information is provided, STO starts to provision the shark tank. We developed STO such that it sends topology information to the SDN controller while the shark tank is being provisioned. Hence, the SDN controller is quickly updated with the latest network topology. Thus, three main steps are involved in provisioning the shark tank:

1) VM instantiation: when topology descriptor file is provided to STO, STO starts instantiating the required VMs based on the specified images. The specified images contain the required applications and services.
2) Overlay Network Establishment: after the VMs are up and running, the vxlan links are used to build the overlay networks and construct the topology. OVS software is pre-configured on the images and a script issues commands to each VM to establish VXLAN tunnels. This process is performed at two levels: host level and switch level. At the host level, the script

issues commands on each host to connect them to their corresponding virtual switches. At the switch level, VXLAN tunnels are constructed to connect the switches to create the desired topology. Fig. 2 illustrates the overlay network establishment in private cloud corresponding to Fig. 1. STO creates different overlay networks to ensure traffic separation between the networks. The reason for creating overlay networks only on private cloud is because after detection, only the safe traffic will reach the public cloud (i.e., mitigation happens before traffic reaches public cloud.)

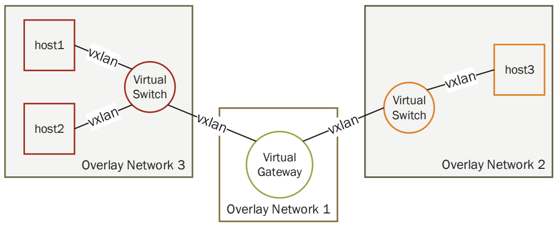


Figure 2: *Overlay network establishment in STO*

3) Topology Update: whenever there is a change in topology (adding/removing VMs, switches, links etc.), the changes are dynamically sent to the SDN controller. The details of this process is explained in Section IV.

IV. SDN Controller: Design and Architecture

The SDN controller acts as the brain of the network and it should always hold a global view of the network so that it is able to configure the switches properly [11]. Our SDN controller runs Ryu SDN framework [12] and programs the network flows using OpenFlow. Our SDN controller does not need a topology discovery module as opposed to other SDN controllers such as Floodlight [13] that involves overhead of topology discovery. Instead, when STO creates the topology, it sends the topology information to the SDN controller so that the SDN controller knows the topology a prior. Following, we explain how the topology information and mitigation actions are sent to the SDN controller in more details.

A. Global View of Network Topology

1) L2 and L3 Network Topology View: Shark tank orchestration tool, STO, builds the entire shark tank automatically through VM instantiation and overlay network establishment. We implemented STO such that once the VMs are instantiated and links are established, L2 (connection between hosts and their switch) and L3 topology (connections between switches in private cloud) information are sent to the SDN controller as JSON objects, stored in the topology file of the SDN Controller. RabbitMQ technology [14] is used for sending this information to SDN controller. When the SDN controller application is run for the first time, it reads in the topology, and builds topology data structures accordingly. Now when a switch forwards a packet to the SDN controller for flow rule installation, the SDN controller programs the switch immediately as it already holds a complete view of the network topology.

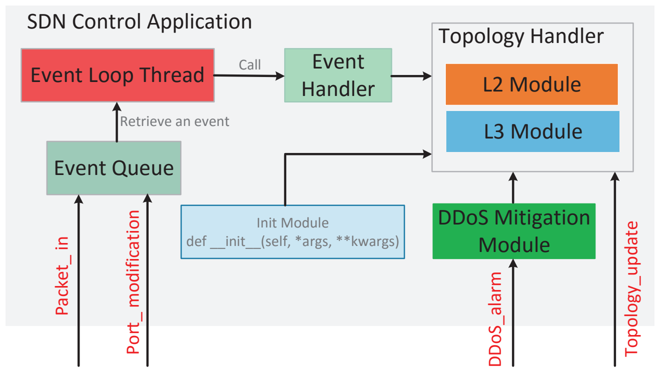


Figure 3: *SDN Controller Application Architecture*

2) Topology Update: When the topology is modified (i.e., a VM joins or leaves the shark tank or a VXLAN link is added/removed) at run-time, the topology information is updated and the controller is notified by STO about the change in the topology (topology update in Fig. 3). We have developed a background RabbitMQ service on SDN controller that always listens to topology update notifications so that the SDN controller continuously keeps the latest view of the network topology.

B. Proactive Flow installation for DDoS Attack Mitigation

We have developed a DDoS attack mitigation component run via an independent thread that is always listening for DDoS mitigation commands. When SDN controller receives the DDoS mitigation commands, it constructs a match field based on the received DDoS mitigation command. The match field includes the source and destination IP addresses of suspicious traffic. The SDN controller then builds an *action* which includes the mitigation policy (redirecting/blocking the traffic). Our mitigation policy also includes a rule to disguise the IP addresses of the shark tank web server and replace it with that of the original server. This way the suspicious user cannot know that it is actually interacting with the shark tank server (i.e., we do NATing in SDN way). After constructing the match field and action, SDN controller pushes `packet-out` commands to switches so that they forward the suspicious flows according to the action. In addition, the SDN controller modifies the rules on the switches' flow tables through `flow-modification` (proactive flow installation) which include the required actions that the switches have to take when encountering any match in the future.

V. Experiment Setup and Evaluation

We have performed a set of experiments to evaluate CAAMP on a hybrid cloud. More specifically, the target application that we would like to protect against DDoS threats is hosted on Amazon EC2 public cloud. The architecture of the target application is depicted in Figure 1. The application has as a three-tier cluster using Apache 2.0 as load balancer, an eStore web application in Tomcat 7, and a MySQL database.

We use Smart Applications on Virtual Infrastructure (SAVI) cloud as the private cloud to host the shark tank. SAVI is an OpenStack-based, multi-edge private cloud for academic research [7]. The specification of VMs used in our experiments are jointly presented in Tables I and II.

We developed a workload generator such that users periodically send HTTP requests to the application to select some items from the database and then wait for a random period of time between 500-950 ms and then send the next request. This way the normal users send approximately 1-2 requests/second. Furthermore, we used Hogna [9] to automatically deploy and monitor the application and act as the autonomic manager in CAAMP architecture. Hogna is an autonomic manager that follows the MAPE-K loop methodology [8] to monitor applications, analyze data, plan and execute corrective actions to meet application performance criteria. CAAMP takes advantage of Hogna to provide elasticity to the application. The elasticity is triggered when CPU utilization goes above 80% or below 40% where in former, an instance will be added as web worker while in latter, one instance will be removed from the original application cluster.

Flavor	VCPU	Memory (GB)	Disk (GB)
Amazon m3.medium	1	3.75	4
Amazon c3.large	2	3.75	32
Amazon c3.xlarge	4	7.5	80
SAVI Small	1	2	20
SAVI Medium	2	4	40

Table I: *Specification of VMs in experiments on Amazon and SAVI clouds*

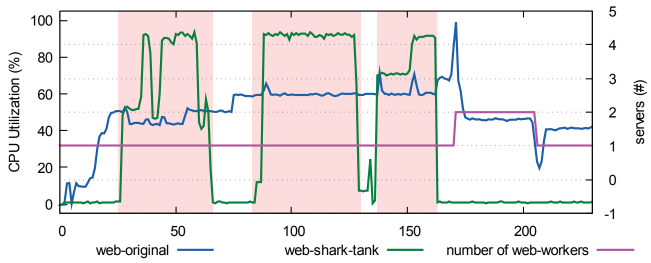
We set up DDoS sensors on both original application cluster and the shark tank to monitor the traffic. The DDoS sensors detect http flood attack which is a denial of service attack where malicious clients try to inundate the resources of applications by sending a large number of requests. DDoS sensors uses Snort to detect DDoS threats. Once suspicious clients are detected, they are reported to Hogna, shark tank will be created and the suspicious clients are then redirected to the shark tank.

We first demonstrate the DDoS mitigation and elasticity features of CAAMP under various traffic conditions. Second, we show the result of the performance analysis on our SDN controller as one of the core elements of CAAMP.

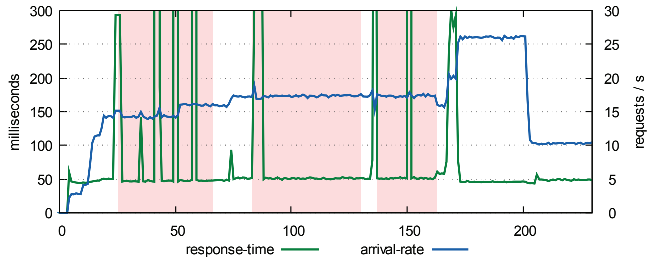
Before getting into the details of the first result, we explain the experiment main phases as follows:

- In the first phase, we introduce multiple attacks to the system and we see how multiple incidents of attack are handled by CAAMP;
- In the second phase, we introduce one intensive (i.e., large arrival rate) attack;
- In the third phase, we introduce two attacks;
- In the fourth phase, we increase the number of normal users (safe users) to trigger the elasticity feature of CAAMP.

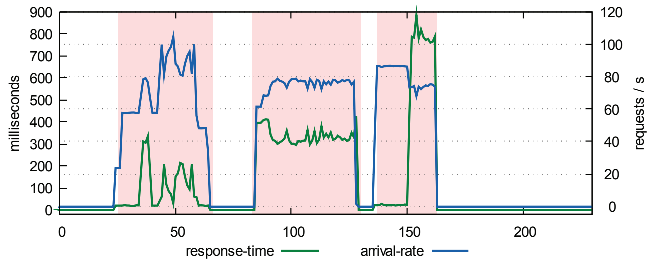
Figure 4 presents the result of our experiment with CAAMP in Amazon EC2 and SAVI hybrid cloud environment.



(a) *CPU utilization.*



(b) *Response time of the web servers in the original cluster.*



(c) *Response time of the web servers in the shark tank.*

Figure 4: *DDoS mitigation. Attack periods are shown in shaded background. CAAMP redirects the suspicious traffic to the shark tank on the fly.*

Figure 4a shows the CPU utilization of the original application web server (blue line), the shark tank web server (green line), and the number of web workers of the original application (purple line). The purple line shows the elasticity feature of CAAMP where web worker instances are added or removed to meet the required performance metrics. Figure 4b shows the arrival rate (blue line) and the response time (green line) of the original web server. The response time uses the left Y-axis while the arrival rate is shown on the right Y-axis. Figure 4c presents the arrival rate (blue line) and response time (green line) of the shark tank web server. The X-axis in all 3 graphs shows the experiment iteration number where each iteration takes approximately one minute. The shaded backgrounds in Figure 4 indicate the attack period.

In the beginning of the experiment shown in Figure 4, we only have normal traffic and the response time of original application server gets stable at around 50 ms. We initiate the first phase of the experiment where from iteration 23 to iteration 60, we start and stop 5 attacks one by one where the attackers send 50 to 100 request/second. Once each attack is started, it increases the response time of the original

Table II: *Experiment specification on SAVI cloud*

Cloud	No. of VMs	Flavor	Software
Original Application Cluster, Amazon	4-6	Load balancer (c3xlarge), web server(m3 medium), database(c3 large)	Apache Load Balancer, eStore Tomcat Web App, MySQL
Shark Tank, SAVI	2	load balancer (Large), Web server (Medium)	Apache Load balancer,eStore Tomcat Web App
Application Proxy, SAVI	1	Large	Custom Java application
Virtual Switches, SAVI	3	Small	OVS
SDN Controller, SAVI	1	Small/Medium	Ryu Custom Controller Application
STO, SAVI	1	Large	Python Scripts

application server (5 peaks in response time on the second plot). The response time during the attack period goes up to 450 ms in Figure 4b, however we did not show the peak response times in order to show the normal response time of original application server at around 50 ms. When first suspicious traffic is started, it triggers the Snort rule and, as a result, the creation of the shark tank is initiated. When the shark tank is ready, the redirection command will be sent to the SDN controller. The SDN controller programs the gateway to redirect the suspicious client to shark tank. Subsequent attack traffic will be redirected to the shark tank. It can be observed that the CPU utilization of shark tank increases from 0 up to around 90% and the total arrival rate to the shark tank raises up to 115 request/second (see Figure 4c). We then stop all the attack traffic at around iteration 60. We can see that the arrival rate and response time of the shark tank decreases accordingly.

We then increase the number of regular users, hence the arrival rate and the CPU utilization of the original web server increases from around 45% to 60% and the response time gets stable again at 50 ms. After some time, In the second phase of the experiment, around iteration 80, we start one attack where the attacker sends 80 request/second. We can see that the attack is detected and redirected to shark tank for further processing. We then do not stop the attack for a while, we can see that the response time and CPU utilization of shark tank stays up. This is an example where attack traffic is kept at shark tank so that attacker receives high response time and assume their attack has been successful and do not adapt their attack strategy. Then we stop the attack, and it can be observed that the arrival rate and response time of the shark tank decrease at around iteration 120 in Figure 4c .

In the third phase, at around iteration 132, we start another attack which results in a sudden increase in CPU utilization and response time of original web server. The Snort alert is triggered, and the attacker is redirected to shark tank. We can see how fast SDN controller installs the redirection rules on the gateway where the shark tank response time and arrival rate start to increase immediately. At iteration 148, we start another attack which is then mitigated and redirected to the shark tank. When the new attack traffic is routed to the shark tank, initially shark tank web server can handle the requests well and the response time is as low as 24 ms from iteration 143 up to iteration 150. But then at iteration 150 due to high arrival rate, the requests start to queue up and we can see the the response time of the shark tank web server increases further to 900 ms and its CPU utilization reaches from 60% to 80% (shown in Figure 4a). We then stop the attacks which reduces CPU utilization, arrival rate and response time of shark tank web server.

So far we demonstrated how CAAMP effectively mitigates the DDoS threats by dynamically redirecting the traffic to the shark tank. For testing the elasticity feature of CAAMP, we start the fourth phase where we increase the arrival rate of requests by adding more regular clients. Higher arrival rate raises the CPU utilization of original web server and the autonomic manager triggers the elastic behavior. In Figure 4a, we can see that up to around iteration 160 (the point that we increase the arrival rate of normal traffic), the number of web servers in public cloud stays at 1. However, when we increase the arrival rate, we can see in the second plot that the response time increases up to 300 ms and the CPU utilization gets to more than 80%. This situation triggers Hogna's elasticity policy and therefore one more instance is added to the original application cluster to keep the response time low (shown in purple in Figure 4a). After one more web worker is added, we can observe that the CPU utilization of original web server gets around 40% and the response time gets improved and stable at round 50 ms again. After a while, around iteration 200, we stop some of the clients which results in the reduction of CPU utilization of the application web server. The CPU utilization gets decreased below 40% which again triggers the elasticity policy and the extra web worker is removed.

The time to redirect the traffic between the original application and the shark tank is an important metric in CAAMP and it is a function of SDN controller response time and the switch's flow installation time. However, we observed that the redirection time is mainly associated with the response time of the SDN controller. Therefore, we carried out experiments to evaluate the SDN controller performance in processing the mitigation commands.

In order to increase the load on the SDN controller, we set a large number of clients to send traffic (attack and no attack). In doing so, we set up a client overlay network on SAVI separated from the rest of the deployment. We used 8 medium VMs on the client network and on each VM, we setup 40 VXLAN links. Each link is connected to a port with an IPv4 address representing a client (i.e., 320 clients in total). Then, we add routing entries to the Linux kernel routing table for each interface so that each interface (i.e., client) sends/receives traffic independently from other clients on the same VM to avoid ARP flux problem [15]. This way, the SDN controller is responsible for setting up flow rules for each client interacting with the application, in addition to processing the mitigation commands.

We performed two types of experiments by using small and medium size VMs on SAVI, specification of which is defined in Table I, to act as the SDN controller. Figure 5

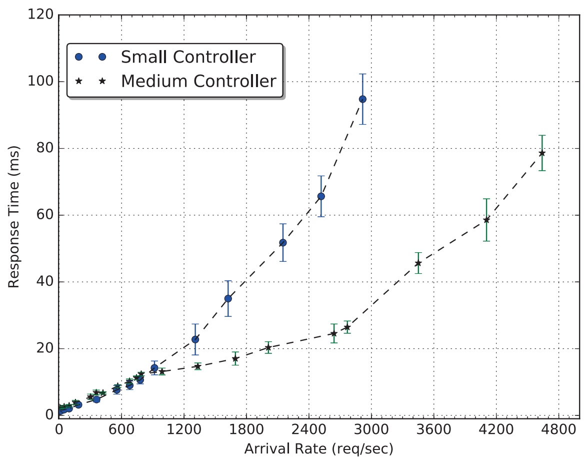


Figure 5: *SDN Controller Performance*

depicts the response time of the small and medium size SDN controller for processing mitigation commands (i.e., proactive flow installation). During this time, the SDN controller is also responsible for installing flow rules for application traffic reaching the switches on SAVI (i.e., reactive flow installation). We performed each experiment type 4 times and calculated the standard deviation as shown in Figure 5. The mitigation commands can be any of the following: (1) the redirection of traffic from the regular application on Amazon EC2 to the shark tank on SAVI; (2) the redirection of traffic from the shark tank on SAVI to the original application on Amazon EC2; (3) and blocking the traffic in case of real attacks.

As can be seen in Figure 5, the response time increases as the mitigation command arrival rates gets increased. For up to 1000 requests/second, we can see that the response time of both controllers are almost the same. Therefore, in terms of performance, for up to 1000 requests/second, a small size controller can handle the requests as good as a medium size controller. We can see that for up to 650 requests/second, the response time for both medium and small controllers is less than 10 ms. However, around arrival rate of 1200 requests/second, we can observe that the response time of the small controller starts to deviate from that of medium one such that the small controller gets saturated at around 3000 requests/second with response time of 94 ms. In medium controller, however, for up to 2100 requests/second, the response time is under 20 ms. The medium controller finally gets saturated at around 4800 requests/second where the response time reaches 88 ms.

Discussion: In CAAMP, when suspicious traffic is initially detected, a shark tank is dynamically instantiated on the private cloud, if it does not already exist. While shark tank is being provisioned, suspicious traffic will be reaching the original application. Hence, there is a trade off on the frequency of provisioning/deprovisioning the shark tank; if we aim to quickly mitigate the effect of potential DDoS attacks, shark tank will not be deprovisioned for some time, even though there is no suspicious traffic at the time. Therefore, once created, shark tank will be maintained for some longer time to serve potential incoming suspicious traffic (i.e., deprovisioning once every 6 hours). The amount of time, the shark tank will be maintained for, once created, depends on the likelihood of the application being under attack. On the other hand, if we are more concerned about the resource consumption on private cloud, shark tank will be deprovisioned after some short amount of time if there is no suspicious traffic. As a result, the discussed trade off has to be taken by the application owners on the frequency of shark tank decommissioning.

CAAMP mainly focuses on the *mitigation* mechanism, therefore, it will be an effective solution to complement detection techniques to reduce large false positives; shark tank provides longer time for attack analysis while it still maintains the service availability until final decision is drawn. Another advantage of CAAMP is that since all the traffic first goes to the private cloud and then to the public cloud, all the redirection between shark tank and the original application happens on the private cloud. As a result, no charge will be applied from the public cloud provider to application owners as all the mitigation process happens before the traffic reaches the public cloud.

VI. Related Work

Arbor Networks [16] surveyed how DDoS attacks evolved over the past decade, and the current threats that data centers have to handle. The survey found that the largest attack in 2014 has peaked at 400 Gbps, and more than 75% of data centers were targeted. It also reveals that the number of attacks and their intensity continues to increase.

Some experts argue that the only sensible solution is to improve security of all internet hosts at the same time [17] and prevent the attacks to become damaging, while other doubt that such mechanisms can be widely adopted [18]. Other researchers claimed that DDoS attacks are not a security problem, but a scalability one [19]. Because the attackers will attempt to make their requests indistinguishable from the rest of the traffic, they can defeat the detection mechanisms. In this case, the only solution to maintain service is to increase the quantity of resources (and this is an expensive solution).

Barna et al. [20][21] investigate mitigation techniques at the application level. They use a performance model to analyze the impact of the traffic on some key performance metrics. The traffic that is identified as undesirable is redirected (using the HTTP redirect header) to a secondary system where the users are challenged with a CAPTCHA. The requests that are part of a DoS attack will be discarded (since the CAPTCHA is not solved) and the misidentified requests are going to be slowed down (but not dropped). These results are further extended in [22], where the authors also use cost of resources when the decisions are made. Kalliola et al. [23] use machine-learning-based DDoS defense mechanism and use SDN techniques to maintaining service quality by detecting and blocking the attacks upon detection. They evaluate their defense strategy using testbed experiments. Wang et al. [24] propose a DDoS attack detection and reaction solution. In their work, the DDoS mitigation strategy depends on public cloud provider to take actions against threats while CAAMP uses overlay networks which makes the solution independent of cloud providers. In addition, Wang et al. [24] do not define where the suspicious traffic is hosted after being detected. Fung et al. [25] propose a dynamic traffic engineering solution to perform DDoS mitigation by assigning suspicious traffic to lower priority tunnels. They evaluated their solution using simulation.

Our work is different from above work on several dimensions: in our solution, we do not block suspicious traffic once it is detected; instead, we dynamically scale out and create a shark tank which offers the same service as the target application on private cloud; CAAMP is an autonomic and versatile solution that isolates the suspicious traffic to protect the application and creates the environment for further analysis of traffic. Moreover, by leveraging resources from private cloud, our solution attains a cost-effective mitigation strategy. We evaluate our solution on a real environment using Amazon as public cloud and SAVI as private cloud.

In previous work [6], a conceptual model of shark tank was introduced. However, in the current paper, we present a concrete architecture and mitigation platform for hybrid cloud environment. In addition, we fully implement the solution and evaluate it on real cloud environment. Furthermore, we present the design details and evaluated the performance of the SDN controller on SAVI cloud.

VII. Conclusion

In this paper, we presented CAAMP, a software defined mitigation platform that dynamically mitigates the DDoS threats on applications on public cloud using private cloud. We demonstrated the features of CAAMP through extensive experiments on Amazon and SAVI clouds.

Our results showed that CAAMP effectively mitigates the severe effects of DDoS attacks. In addition, CAAMP yields elasticity feature to applications on cloud to maintain the desirable performance. CAAMP provides the same functionality to suspicious clients until final decision is made. This way, on one hand, the original application is protected against damages of DDoS threats, and on the other hand, it lowers the rate of false positive alarms which results in customer dissatisfaction and decline in service revenue.

As future work, we plan to investigate the end-to-end performance of CAAMP. Also, we intend to examine various policies for provisioning/deprovisioning the shark tank and the impact of those policies on the applications in terms of performance and economics. We are also going to include an in-house detection mechanism in CAAMP where the SDN controller uses network statistics to detect suspicious/malicious traffic.

Acknowledgment

This research was supported by the Natural Sciences and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network and NSERC CGSD.

References

[1] R. Dobbins, C. Morales, D. Anstee, J. Arruda, T. Bienkowski, M. Hollyman, C. Labovitz, J. Nazario, E. Seo, and R. Shah, "Worldwide Infrastructure Security Report," http://www.arbornetworks.com/dmdocuments/ISR2010_EN.pdf, Arbor Networks, Tech. Rep., 2010.

[2] M. Shtern, M. Smit, B. Simmons, and M. Litoiu, "A runtime cloud efficiency software quality metric," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. ACM, 2014, pp. 416–419.

[3] M. Jensen, J. Schwenk, N. Gruschka, and L. Iacono, "On technical security issues in cloud computing," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, Sept 2009, pp. 109–116.

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

[5] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "SDNIPS: Enabling Software-Defined Networking based intrusion prevention system in clouds," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, Nov 2014, pp. 308–311.

[6] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou, "Towards mitigation of low and slow application ddos attacks," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, March 2014, pp. 604–609.

[7] SAVI, "Smart applications on virtual infrastructure," http://www.savinetwork.ca/.

[8] IBM, "An architectural blueprint for autonomic computing," IBM, Tech. Rep., 2005.

[9] C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern, "Hogna: A platform for self-adaptive applications in cloud environments," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2015 IEEE/ACM 10th International Symposium on*, May 2015, pp. 83–87.

[10] "Open vSwitch," online, Access date: 11 Feb. 2016. [Online]. Available: http://openvswitch.org/

[11] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. IEEE Computer Society, 2015, pp. 50–56.

[12] "Ryu SDN Framework," online, Access date: 28 Dec. 2015. [Online]. Available: http://osrg.github.io/ryu/

[13] "Project floodlight," Online:http://www.projectfloodlight.org/projects/, 2012, access date: April 6,2015.

[14] "RabbitMQ," online, Access date: 28 Dec. 2015. [Online]. Available: http://rabbitmq.com

[15] "ARP Flux issue in Linux," online, Access date: 19 Nov. 2015. [Online]. Available: http://serverfault.com/questions/336021/two-network-interfaces-and-two-ip-addresses-on-the-same-subnet-in-linux

[16] Arbor Networks, "Worldwide infrastructure security report," 2015.

[17] F. Kargl, J. Maier, and M. Weber, "Protecting web servers from distributed denial of service attacks," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. ACM, 2001.

[18] M. Sachdeva, G. Singh, and K. Kumar, "Deployment of Distributed Defense against DDoS Attacks in ISP Domain," *International Journal of Computer Applications*, vol. 15, no. 2, pp. 25–31, February 2011, published by Foundation of Computer Science.

[19] Y. Chung, "Distributed denial of service is a scalability problem," *CoRR*, vol. abs/1104.0057, 2011.

[20] C. Barna, M. Shtern, M. Smit, V. Tzerpos, and M. Litoiu, "Mitigating dos attacks using performance model-driven adaptive algorithms," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 1, pp. 3:1–3:26, Mar. 2014.

[21] ——, "Model-based Adaptive DoS Attack Mitigation," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '12. IEEE Press, 2012, pp. 119–128.

[22] C. Barna, M. Shtern, M. Smit, H. Ghanbari, and M. Litoiu, "Model-driven elasticity and dos attack mitigation in cloud environments," in *11th International Conference on Autonomic Computing (ICAC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 13–24.

[23] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding DDoS Mitigation and Traffic Management with Software Defined Networking," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, Oct 2015, pp. 248–254.

[24] B. Wang, Y. Zheng, W. Lou, and Y. Hou, "DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, Oct 2014, pp. 624–629.

[25] C. J. Fung and B. McCormick, "Vguard: A distributed denial of service attack mitigation method using network function virtualization," in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 64–70.