Spring 2017

Intelligent Threat-Aware Response System in Software-Defined Networks

Kunal Ketan Goswami
San Jose State University

INTELLIGENT THREAT-AWARE RESPONSE SYSTEM IN SOFTWARE-DEFINED NETWORKS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kunal Goswami

May 2017

The Designated Thesis Committee Approves the Thesis Titled

INTELLIGENT THREAT-AWARE RESPONSE SYSTEM IN SOFTWARE-DEFINED NETWORKS

by

Kunal Goswami

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2017

Dr. Younghee Park	Department of Computer Engineering
Dr. Xiao Su	Department of Computer Engineering
Dr. Hyeran Jeon	Department of Computer Engineering
Dr. Youngsoo Kim	Department of Electrical Engineering

ABSTRACT

INTELLIGENT THREAT-AWARE RESPONSE SYSTEM IN SOFTWARE-DEFINED NETWORKS

by Kunal Goswami

Software-defined networks decouple the control plane from the data plane, enabling researchers to evaluate protocols and network configurations through the centralized point of control, the controller. They provide easy management and automation, scalability, and flexibility in the traditional computer network. In spite of these advantages, software-defined networks fall prey to various denial-of-service attacks specific to network protocols and applications despite their simplicity. There is a need to implement intelligence in the controller as a countermeasure for not only the various types of denial-of-service attacks but also the increasing sophistication involved in them. In this paper, an intelligent threat-aware response system is proposed for defending against any attack by using reinforcement learning. Reinforcement learning can acquire intelligence for detection and reactive actions through experience with various attacks. This experience is obtained from interactions with the computer network through the controller. With the combination of reinforcement learning and the software-defined networking controller, the goal of the autonomous threat response system can be achieved.

ACKNOWLEDGMENTS

I would like to thank Dr. Younghee Park for her continuous guidance and support throughout my time as a graduate student. I am greatly indebted to Dr. Park for all the technical concepts and research styles that I learned from her. Another person who had an impact on my work is Dr. Chungsik Song. I would like to thank him for his valuable insights and comments on the algorithmic approach for this work. I offer my gratitude to the committee members for their expert advice on how to formulate the thesis, evaluation, and presentation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Software-defined networking (SDN) provides an abstraction of programmability in traditional computer networks. Computer networks can be simulated virtually and interfaced through the SDN controller from topology configurations to protocol-specific behavior. This abstraction is facilitated by OpenFlow [1], a protocol which decouples the data plane from the control plane on the switch. OpenFlow protocol also makes the computer network more flexible. In addition, since the control plane is separated from the data plane, it is possible for researchers to deploy and test various protocols in real-time.

However, along with all the advancements in the field of computer networks, there have been advances for compromising the networks as well. Protocol features such as three-way handshake in TCP, HTTP GET request processing, and ICMP response can be exploited for selfish gains. The same features which provide a secure connection for communication, availability, and hassle-free data transfer lead to the misuse. A few examples of such misuse are TCP SYN flood attack, HTTP GET Request flood attack, and Ping flood attack. Recent history of network attacks [2, 3] brings to light the innovative approaches taken by adversaries in terms of denial-of-service (DoS) attacks. This calls for a measure which can surpass the same innovation and counter the attacks.

The other concept which enters the picture is one of reinforcement learning [4]. It is a paradigm which closely imitates the human way of learning. The human brain learns with the help of interaction with and feedback from a corresponding environment. Given a state of the environment, the human brain evaluates the best possible action to interact with it and gain the best possible outcome. The definition of the word “best” is highly subjective. However, the same learning procedure helps us understand concepts from natural language [5] to aeronautics [6]. This paper presents an approach in which a reinforcement learning paradigm is combined with SDN to create an intelligent threat-aware framework which is able to take responsive actions, given access to network behavior information.

The SDN architecture enables the underlying computer network to be managed with the help of the controller, the logical centralized point of access [1, 7]. A lot of information in the form of network traffic data propagates through the computer network. This information encodes the user-specific behavior, IP specific behavior, and usage patterns. With the introduction of reinforcement learning, this information can be represented in a succinct form of a “state”. The state can then be utilized in a state-based learning process to train an agent for taking reactive action against network threats.

The proposed framework implements the above discussed principles with traffic analyzer, reinforcement learning agent, and threat response modules. The traffic analyzer module gathers information through the controller and represents it into a form suitable for the reinforcement learning agent. The agent computes the likelihood of the action with the maximum long-term benefit in terms of the computer network. The selected action becomes the reaction of the controller for the corresponding network threat.

Intelligent intrusion detection has been an area of research for quite some time; many approaches have been developed using reinforcement learning [8, 9]. Some of these approaches include providing a response to a network attack. Examples include router throttling [10] and Q-learning [11] applied to routing. However, there are other approaches to network attack prevention as well which solely focus on actions provided by software-defined networks [12]. Frameworks such as AVANT GUARD [13] and FLOW GUARD [14] focus on changing the flow rules in the data plane to make software-defined networks more secure and developing a proxy firewall to detect the changes in the network traffic, respectively. These approaches focus on developing an intelligent response to networks or using software-defined networks to prevent a network attack. This thesis focuses on delivering the best of both reinforcement learning and software-defined networks.

The unique contributions of this work include a novel architecture consisting of reinforcement learning combined with SDN to address network threats, network condition monitoring through network congestion function, and implementation of this framework integrated into an open-source SDN Controller RYU [15]. The scope of network attacks covered by this framework at present includes various DoS attacks.

The rest of this document is organized as follows: Chapter 2 provides a brief survey of similar work in this domain. Chapter 3 provides the details of the system architecture and other implementation details. The experimental results of the prototype are discussed in Chapter 4, followed by discussion in Chapter 5. Chapter 6 provides the future scope of this research and contains the conclusions derived from it.

CHAPTER 2
Related Work

There are many approaches one can take to defend a computer network against threats, some of which involve the use of sophisticated hardware to detect and nullify threats. Others make use of the programmability offered by SDN. Recently, the concept of combining intelligence with network security has been gaining in popularity. This intelligence is of the form of machine learning applied to various network statistics for threat detection and/or prevention. This chapter discusses various approaches applied to security in SDN.

2.1 Software-Defined Network based Network Security

Padekar, Park, Hu and Chang [16] propose the AEGIS framework to protect the SDN Controller and the network from erroneous network applications that misuse controller APIs. The AEGIS framework follows a three-step process to secure the controller: The first step is to identify all of the APIs that modify the internal data for the network, the second step is to generate access rules which govern the behavior of a network application, and the third step involves the decision engine which intercepts and validates the API calls in real time to ensure safe operations. These authors have developed a prototype and tested it against six network attacks for evaluating the system.

Wang, Zheng, Lou and Hou [17], on the other hand, propose an enterprise network security framework which utilizes the monitoring capabilities of SDN to provide a flexible interface for taking actions against network attacks. They address denial-of-service (DoS) attacks against cloud infrastructures which are facilitated by SDN. The authors argue that, with the help of a properly designed framework, one can counter such network attacks solely with the help of SDN. To the best of their knowledge, the authors were the first to propose a network defense architecture such as DaMask for DoS and Distributed DoS (DDoS) attack mitigation.

Lim, Ha, Kim, Kim and Yang [18] discuss a SDN-based defense against DoS and DDoS attacks by botnets. Due to the difficulties of identifying these attacks using network monitoring, the responsibility lies on the victim server or host. The authors propose a defense framework using the standard OpenFlow implementation to thwart DDoS attacks. Their implementation relies on the victim server to determine whether or not it is under a DDoS attack. Once the server confirms it, the proposed solution suggests a redirection URL to all incoming requests and continues blocking subsequent requests to the same IP address. The intuition followed by the authors is simple; the preprogrammed bots cannot comprehend or change the address of the victim immediately after the URL redirection is applied.

Wang, Xu and Gu [19] address the control plane to data plane saturation attack on SDN controllers. They propose a framework called FLOODGUARD to analyze the flow rules pushed on switches with the controller and prevent this attack. Another framework which addresses security in SDN is AVANT-GUARD [13], which addresses the attacks focused on the SDN controller itself. The authors propose an extension to the data plane in the form of "connection migration" and "actuating trigger." These are novel contributions to the software-defined network environment and help in minimizing the probability of saturation attacks on the controller. Braga, Mota and Passito [20] propose a lightweight DoS detection framework using NOX [21] controller. They use the concept of IP flows [22] to detect network attacks. These frameworks certainly are well developed but can be extended when combined with the capacity to take action against an attack.

Another approach to protecting the network involves the help of host IP address mutation. Jafarian, Al-Shaer and Duan [23] propose a random host mutation approach which assigns virtual IP addresses to all the hosts present in the network.

The virtual IP addresses are changed at random intervals of time. The benefit is that the adversaries cannot discover these hosts through scanning or worm propagation. It is an ingenious way of defending against network attacks. However, it might fall prey to an attacker who compromises an authorized system, who can then use the actual IP addresses of these hosts to communicate.

2.2 Machine Learning based Network Security

Malialis and Kudenko [10] discuss an approach to intrusion detection using reinforcement learning. The focus of their work is to inhibit DDoS attacks. They use a multi-agent router throttling approach to regulate the traffic towards a victim in such a way that it does not collapse. This approach proposes "coordinated team learning" as an extension to the original multi-agent router throttling approach. Coordinated team learning is decentralized, which is one of the biggest advantages it has over the traditional approach. They experimented with up to 100 reinforcement learning agents against dynamic attacks and managed to outperform state-of-the-art router throttling procedures.

Another approach to intrusion detection in networks has been along the lines of Q-learning [11], a novel approach indeed. Detecting network intrusion with the help of a modified version of the traditional Q-learning algorithm, this framework achieves an accuracy of 98%. The authors combine the Q-learning algorithm with rough set theory to achieve higher classification results. Reinforcement learning is popularly implemented for interacting with the system rather than classifying the state of the system. Sengupta, Sen, Sil and Saha [11] developed an online system in which the network packets become the actions in the system, and the traffic is categorized as either "normal" or "anomaly."

Chung, Khatkar, Xing, Lee and Huang [24] propose the NICE framework for securing a virtual network environment. They focus on securing cloud systems which provide virtual machines to users. Their framework prevents these virtual machines from being zombie machines by using network intrusion detection. In their approach, they classify network intrusion by using the similar activities of the users and analyzing them. They also scan the virtual machines for vulnerabilities to check for compromised machines.

Restricted Boltzmann Machines (RBMs) [25] have also been used for network anomaly detection. The work primarily revolves around classifying network traffic as hostile or normal. The authors gather network data for training the RBM. However, they state that "network traffic is very complex and unpredictable, and the model is subject to changes over time, since anomalies are continuously evolving." The network traffic data are converted into a time series, and the algorithm is trained based on a discriminative approach and a generalized approach. The generalized approach trains on normal data, considering each data point by how well it fits with the others, whereas the discriminative approach focuses on learning the difference between hostile data points and normal data points. Cannady [26] developed an intrusion detection system with the help of adaptive neural networks. The neural network received many more inputs along with feedback from the system. The trained system had an error rate of around 2% against known attacks. The author also demonstrated that the neural network adapts very well to the new attack vectors upon feeding them as input.

Wang, Chao, Lin, Lin and Lo [27] proposed an attack detection and mitigation system in SDN. This system made the use of support vector machine (SVM) algorithm to detect network attacks. The authors analyzed the features generated from the network with decision tree algorithm and trained the SVM classifier to detect threats. The mitigation action was taken through the northbound REST API of the controller with the help of a flow rule. Although this work has promise, it has an inherent need of labeled data. For a dynamic environment such as SDN, it is not really feasible to generate labels to depict all scenarios. Colbaugh and Glass [28] propose a proactive framework for defense against cyber attacks. The authors develop this framework by "modeling attacker / defender interaction as a stochastic hybrid dynamical system (S-HDS)" to counter against more complex or synthetic attacks. The system is evaluated on the KDD cup 1999 dataset and Ling-Spam dataset. The authors created a synthetic attack space to train and evaluate the algorithm. This approach can be extended by providing a semi-supervised or an unsupervised solution combined with measures which qualify as the proactive measures against network threats.

An early work for intrusion detection is by Xu and Luo [29]. The authors modeled network intrusion detection with the help of reinforcement learning. The intrusion detection problem is modeled in the form of a Markov decision process (MDP) by using host-specific features in the network. Though the work is different in nature from the one proposed in this paper, it is crucial to highlight the approach used by the authors because it acts as an inspiration to model the problem of reactive action against network threats as a MDP. The various frameworks presented in this section prove to be an inspiration to learn from and extend the same for achieving better network security.

This paper proposes a framework which intends to combine the programmability and flexibility of SDN with the learning nature of reinforcement learning to not only detect network threats but also autonomously take countermeasures against such threats. It encompasses the MDP and Q-learning algorithm to develop intelligence integrated with the SDN controller.

CHAPTER 3

System Architecture

This chapter presents the system architecture for the intelligent threat-aware response system in software-defined networks. It gives an explanation of the different components of the framework in a unique combination of northbound and southbound interfaces for knowledge discovery and data mining (KDD) process, reinforcement learning, and the reactive actions. The three main components are as follows:

- Traffic Analyzer

  A southbound interface for monitoring the network traffic propagating through each of the switches connected in the network. The same implementation facilitates the initial steps of the KDD process.

- Reinforcement Learning Agent

  An additional component developed in the SDN controller operating system for the implementation of the Q-learning algorithm.

- Threat Response

  Another southbound interface which provides APIs to communicate with OpenFlow and ovsdb to manage the bandwidth of the switches and also to update flow tables on the switch.

Figure 1 depicts the component-based block diagram of this framework. In the following sections, each component of the framework is discussed in detail.

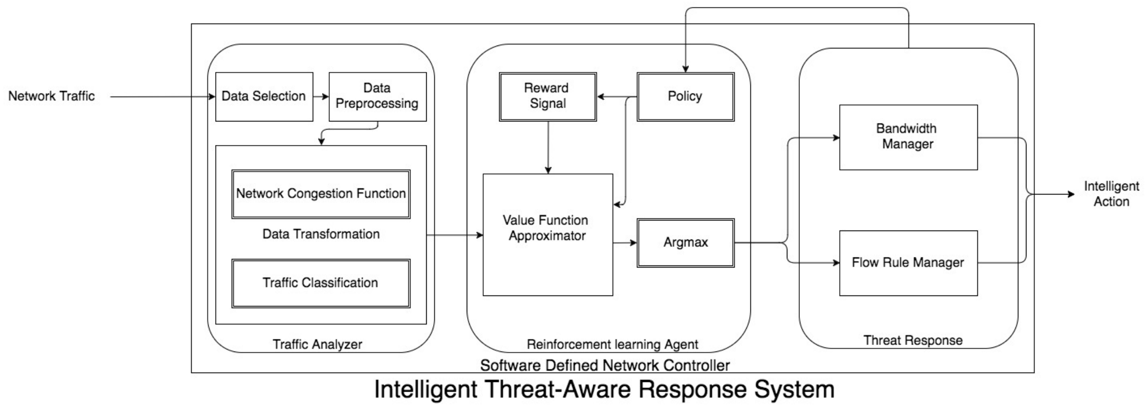


Figure 1 – The component-based architecture and representation of control flow through the system

3.1 Traffic Analyzer

A threat-aware system needs to continuously monitor the network to detect any unusual behavior. The traffic analyzer is the module responsible for the network monitoring in the proposed framework. It collects information about the total number of packets sent and received along with the total number of bytes sent and received through the network. However, the total number of packets and bytes alone do not provide much of an insight to discriminate between various types of network threats. To achieve that, traffic analyzer module collects information specific to different protocols such as TCP, UDP, HTTP, etc.

The traffic analyzer module is a southbound interface in the SDN controller operating system that monitors network traffic. The OpenFlow protocol APIs allow the controller to check for flow-specific and port-specific statistics. This framework is implemented as a part of the RYU [15] SDN controller. RYU is an event-driven controller wherein the southbound APIs are invoked whenever there is an appropriate event. For example, one could set up an event listener for an OpenFlow packet and define an API to be invoked whenever there is an incoming OpenFlow packet. In a similar way, there are listeners for port-specific statistics on the switch. The traffic analyzer sends a stats request to the switch, invoking the corresponding API for replying with the real-time statistics.

The other perspective to network monitoring addressed in this work is one from the KDD process. The statistics lack contextual information to derive conclusions. This contextual information can be captured by accumulating the statistics over time or combining together two or more values to obtain a third one. This module deals with the initial phases of the KDD process to obtain a representation of the network through the information collected by monitoring it.

3.1.1 Data Selection

Just as the name suggests, the data selection process involves the determination of the respective data type and source. As discussed earlier, the selected statistics comprise the total number of packets, the total bytes of data being transferred, and protocol-specific information with respect to the overall values. For example, let us suppose that 10% of the bandwidth capacity is under use in the network. The first set of features would determine how much of that 10% is occupied by the respective switches present in the network and then calculate what percent of the overall usage belongs to a specific protocol. The features are computed with the help of the network congestion function and traffic classification while transforming the data into an appropriate representation for the reinforcement learning agent.

DoS attacks mainly focus on exhausting the bandwidth of the network or the resources of the switch and/or a host. The information about the total number of packets and the total size allows one to estimate the bandwidth usage to some extent. Resource exhaustion is achieved by sending spoofed or fake requests to the victim; therefore, protocol-specific statistics are selected to address these type of attacks. Also, if a network is under attack, the network latency would be relatively high.

Hence, network latency is also one of the features included during the data selection phase.

3.1.2 Data Preprocessing

The data that exist in the natural state or exist in the world are not necessarily in the best possible format. First of all, it is not necessary for the data to have all the uniform values as an ideal data set. For this research, an ideal data set is one where there are no missing or invalid values for any considered parameter. The challenge to solve in this phase is generating the features in a synchronized manner. It is important to measure the number of packets, the total number of bytes exchanged, and the network latency at the same time in order to represent the network state for that time. Even a slight shift in the measurement would lead to confusion in the network state. For example, if the number of packets and the number of bytes are recorded at different times, one could end up with many more bytes for a relatively small number of packets and vice versa. Such a discrepancy in the statistics would mislead the reinforcement learning agent and cause errors while it selects a reactive action. During the data preprocessing step, the data are filtered to form the features that will be used later for further computation.

3.1.3 Data Transformation

The information obtained so far lacks context. The context information allows the framework to understand the need for taking a reactive action. That is, the context information helps to derive some conclusion from the data. With respect to the framework, the context information summarizes the overall behavior of the network at a given point in time. The aim of this phase is to provide concrete evidence that the network is congested or is under an attack. This evidence contributes towards the intelligence of the proposed framework. The intelligence is not only in taking the right action given a threat, but also on the right understanding of the network behavior. For example, if the framework scales up the bandwidth at a time when the network does not need more bandwidth to function, it defeats the purpose of an intelligent system. The context information is provided with the help of network congestion function and traffic classification. Network congestion function is one of the contributions of this research to identify the network behavior given limited information. Traffic classification is an extension to the network congestion function for obtaining protocol-specific congestion across the network.

3.1.4 Network Congestion Score

The very first requirement of a threat-aware response system is to know if there is any threat to the computer network. A network threat can be identified as an unusual behavior of the network, whether it is through the traffic or by the switches. This unusual behavior needs to be captured through some measure to determine if there is an active threat to the computer network. The network congestion function carries that purpose in the framework. This function returns a congestion score which helps the reinforcement learning agent distinguish between the normal and abnormal network behavior. That is, the congestion score becomes the "observation" and input to the reinforcement learning algorithm. Hence, congestion score allows the intelligence module to take reactive action in case of a threat and scale up the network resources when needed.

The network congestion score is defined as a function of throughput and network latency. The throughput of the network indicates the number of bytes or the rate of bytes being exchanged through the network per unit time. The network latency on the other hand indicates the round trip-time (RTT) that a packet would take for traveling across all the nodes of the network and returning to the origin.

That is, the packet will originate from the switch and hop onto each of the hosts present in the network and back to the switch. The following equation can be used to calculate network congestion score:

$$\text{Congestion Score} = \left\{ \frac{\triangle B}{B} * 100 \right\} + \left\{ \frac{1}{k * n} * \frac{\triangle L}{L} \right\} \tag{1}$$

$\triangle B$ is the difference between the current speed of the port and the default speed of the port B, the current speed is the byte rate with which the port is transmitting data, and the default speed is the maximum speed with which the port can transmit data. The value of n indicates the total number of links in the local network and $\triangle L$ is the difference between the network latency and the ideal latency L of the local network. The total number of links helps to determine the ideal network latency. The importance of latency for the congestion can be specified with the constant k. A relatively smaller value of k would increase the emphasis of network latency in the congestion score and vice versa. The network latency has a penalty measure just to assign a higher importance to the bandwidth usage, as most network attacks focus on exhausting the network bandwidth.

The intuition behind this equation is very simple and can be understood with the help of simple questions. First, how does one determine if the network is congested? It is by the amount of data that is exchanged in the channel. The available bandwidth of the channel indicates the maximum amount of data that can be exchanged through the channel, and the throughput indicates the real time data exchange. The second question to ask is, how fast does a packet travel from one end to the other end of the network? The answer is network latency. It is necessary to determine, given n links how much time it takes for the packet to make a round trip of the local network. If it is an ideal network, it should be less than 1 ms per link. Any congestion in the links would indicate the extent to which the links are compromised. A congested link would result in a relatively higher value of network latency.

3.1.5 Traffic Classification

The traffic classification is an intuitive extension to the network congestion function discussed previously. The overall network usage can be estimated through the congestion score. The protocol-specific network usage can be estimated through the traffic classification, that is, the number of packets and number of bytes per packet per protocol. The term traffic classification originates from the question that, out of the total used bandwidth, how much is occupied by a certain protocol P? For example, if the total bandwidth usage comes out to be 70%, traffic classification for a protocol P measures the portion of bandwidth occupied by P.

As discussed earlier, because most of the DoS attacks are protocol-specific, the intuition behind traffic classification is to narrow down the protocol involved in the attack. Network congestion function checks if the network is congested or not. Similarly, traffic classification checks for the protocol-specific congestion. For the scope of this research, HTTP, TCP, DNS, UDP, and ICMP protocols were considered. These protocols are largely used in day-to-day applications for complex communication carried out over the network. A southbound interface using tshark [30] was developed to gather the protocol-specific statistics, which is then combined with the statistics obtained from OpenFlow specific southbound APIs.

3.2 Reinforcement Learning Agent

This section explains the reinforcement learning paradigm as applied to network security. This module is developed as a part of the RYU [15] SDN controller operating system. Formally speaking, a reinforcement learning problem consists of the following sub-elements:

- Policy
- Reward Signal
- Value function

The reinforcement learning problem is the selection of the right reactive action given the current network behavior. Let us go over each of the sub-elements of reinforcement learning. This approach not only provides an overview of reinforcement learning but also provides context information that facilitates the relationship between the concept of reinforcement learning and its practical implementation for network security.

3.2.1 Policy

According to Sutton and Barto [4] "A policy is defined as a function which determines the action taken by a reinforcement learning agent given some state S_i."

The policy can choose any action from the set of available actions. It is formally represented with the help of $\pi^*(S)$. The problem is represented in a state based system. That is, there is a set of states in which the system can be and there is a set of actions which make the system transition from one state to another. With reference to the learning problem in this paper, the actions would be the sub-modules present in the threat response module.

3.2.2 Reward Signal

Reinforcement learning paradigm is reward driven. A reward can be defined as a favorable condition being met upon taking some action. This action would essentially make the system transition from one state to other. For example, when a child learns to walk, the reward is moving from one place to another without being hurt, or taking each step without falling. It could also be possible that the reward is considered for both the situations, not falling on each step as well as not falling even once while walking from the starting point to the destination. For the reinforcement learning problem addressed by this paper, the reward signal is determined from the input state at time t, S_t and the input state at time t+1, S_{t+1}. With the help of the congestion score and protocol-specific congestion, it is possible to determine whether the congestion increased or decreased after taking the particular action. The same measure becomes the key to the reward signal for the reinforcement learning agent. If the congestion of the network at time $t+1$ did not improve by some particular action taken at time t, the action was not really the best action that could have been taken. The reward signal specifies the immediate reward received upon a transition. However, there is a greater part of the reward obtained by taking the value of the action into consideration. It is also considered the goal of a reinforcement learning problem. After all, every problem is solved with the motivation of obtaining the maximum possible reward.

3.2.3 Value Function

Consider a system in which the number of states, the reward of each state, and the value of the action taken from a given state are known beforehand. This is the ideal reinforcement learning problem that can be solved by planning or dynamic programming. However, in the real world, even if the number of states and the immediate reward of each state are known, the value (or long-term benefit) of an action given a particular state is unknown. This gives rise to the concept of the value function in the reinforcement learning problem. The overall reward for taking any action is the sum of the immediate reward obtained from the state and the value of the action taken with respect to the previous state.

Suppose that in a reinforcement learning problem, the agent has to find the path from one state to another state. If the agent keeps selecting the path which gives the highest immediate reward, it is possible that the agent might not devise the most optimal path to the destination. However, it is the optimal solution which is very much desired, not just a good enough solution. The value function in a reinforcement learning problem helps the agent determine which action or actions correspond to the maximum long-term reward which can be obtained. The learning problem which arises is how to determine the value of an action. It is not really possible to estimate the value, given some state S_i and some action A_i.

The value of an action is captured through the experience obtained by the reinforcement learning agent over time as it interacts with the environment. The more experience that a reinforcement learning agent has with each respective action, the more likely it is that the agent can select the best action in case of a complex situation. That is, if the network is currently under attack, the experience of the reinforcement learning agent with each possible reactive action determines the likelihood of the agent to make a wise decision. The value function is the learning challenge in the reinforcement learning problem. It is not known beforehand, nor can it be guessed at random. However, there are approaches with which the value function can be approximated such as Monte Carlo simulations, Temporal difference (TD) learning, Q-learning, etc. The approach used for this framework is Q-learning, the working of which is explained in one of the following sections.

3.2.4 Overview: Markov Decision Processes

Markov decision processes (MDPs) are used to model decision making problems when the outcome cannot be predetermined completely. The outcome usually has some degree of uncertainty associated with it. Covering the concept of MDPs in depth is beyond the scope of this paper, but this section attempts to introduce it briefly.

According to Sutton and Barto [4], "A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP." The Markovian property informally specifies that only the present matters. Consider a scenario wherein there is some decision to be taken with respect to some parameters. If this decision can be taken by considering only the last few values of the parameters rather than the complete history, this system is a MDP. More formally, MDPs prove that taking into account the entire history of parameters is equivalent to taking into account only the recent history of the environment. Again, the proof of this is beyond the scope of this paper. With reference to network attacks, it can be said that only the present network behavior information or the network behavior information collected for the past few seconds matters. If the network is under attack, the present information collected from the network would reflect the attack, not the information which was gathered from an attack which occurred ten days ago. With this argument, one can say that the reinforcement learning problem that this framework is trying to solve is a MDP. Therefore, there exists a set of actions which can solve the problem optimally.

3.2.5 Q-learning

The reinforcement learning problem consists of the state space and the action space. The state space consists of the information available to the agent at each point in time t. The action space is the set of all possible actions the agent can undertake from a particular state S_i. As discussed earlier, the long-term benefit of each action with respect to the state S_i is not known beforehand. Let us assume that the function $Q^*(S_i, A_i)$ represents the long-term benefit of the action A_i taken in the particular state S_i. There is also an immediate reward R_i for the action and state pair. Assume that for all $A_i \in A$, where A is the set of all possible actions, the function $Q^*(S_i, A_i)$ outputs real values indicating the long-term benefit of taking action A_i in state S_i. In this case, it is possible to select the action that corresponds to the maximum long-term benefit from the action space.

However, the $Q^*(s_i, a_i)$ function is not always known beforehand in a reinforcement learning problem. This $Q^*(s_i, a_i)$ function is more commonly known as the value function in a reinforcement learning problem and as the Q-function for Q-learning. As discussed previously, the value function can be approximated by accumulating experience over time. The reinforcement learning agent accumulates experience through the interaction with the computer network. Depending on the action, the reinforcement learning agent would receive more information from the network. This information would then be passed to the reward signal to obtain the immediate reward for the action. The immediate reward and the estimated long-term benefit of the action taken form the learning process for Q-learning. The intuition behind Q-learning is that given enough tuples of states, actions, and rewards, it is possible to approximate the value function accurately.

Q-learning approximates this function by using a non-linear curve-fitting approach such as neural networks. The value function is a pattern which can be approximated by taking into account the present network behavior information and comparing it with the previous network behavior information to obtain the reward. The reward acts as an implicit label, and the algorithm can be trained upon it to obtain an approximation of the value function.

The problem now becomes more of a learning through experience. As the number of interactions with the environment increases, the Q-learning algorithm accumulates more information. Suppose the approximated value function is represented by $Q^{'}(s_i, a_i)$, Now as $time \to \infty$ the approximation of the value function $Q^{'}(s_i, a_i) \to Q^*(s_i, a_i)$. It is an approach which can be summarized as pattern recognition through experience. The function approximation is facilitated by neural networks, or an architecture better known as Q-networks. Q-networks learn through experience, considering the information obtained from the environment in the next state as an implicit label.

Another concept to be considered is one of creativity. Reinforcement learning agents are creative by nature in terms of the developed strategies. With enough experience, the agent can propose a novel strategy to reach the goal state or to maximize the value obtained in the process. The fundamental idea behind this creativity is the exploration vs. exploitation for the action space.

Given a set of actions A, it is very important that the reinforcement learning agent evaluates each action A_i from it. The above mentioned idea is more formally known as exploration. Suppose that the reinforcement learning agent chose an action A_j from the action space A, and the action turned out to be something beneficial. Now, the value of this action would be increased as compared to the other actions as it returned a good reward. Also, in the future, if a scenario similar to the current scenario arises, the likelihood of choosing the same action A_j increases. This concept is called exploitation. A balance is needed between these two concepts to solve the problem. The goal is to find the best possible actions from the action space A, and it is necessary for the agent to try out different actions from the action space.

The agent can be forced to take up some action from the action space by just selecting a random value at random time intervals. Let us say that for some ϵ and a random real value $P \in [0, 1]$, if $P > 1 - \epsilon$, the reinforcement learning agent chooses a random value and determines the reward received from it. Otherwise, it selects the action with the maximum value of $Q^{'}(S_i, A_i)$. The value of epsilon decreases with the number of interactions carried out by the reinforcement learning agent. The ideal initial value of ϵ is 1, as $time \rightarrow \infty$, $\epsilon \rightarrow 0$. This methodology is known as $\epsilon - greedy$ approach in reinforcement learning. Now that the foundation of how different approaches work for value function approximation has been developed, the concepts can be combined to present the algorithm that updates the value of $Q^{'}(S_i, A_i)$, making it closer to the ideal $Q^{*}(S_i, A_i)$ function. The algorithm used to achieve it is known as Q-learning and is represented by the following equation, where the next state of the system is $S_i^{'}$:

$$Q^{'}(S_i, A_i) = R + \gamma max_{A'}(Q^{'}(S_i^{'}, A_i^{'})) \tag{2}$$

R is the immediate reward obtained by the agent upon taking the action A_i. γ is the discount factor which avoids the risk of infinite reward values when there is no termination state in the system. The ideal value of γ is taken as 0.9 [31]. With the above mentioned equation, the value of each action is updated. For a large enough number of iterations, this value function approximates the ideal value function.

3.3 Threat Response

The threat response module interacts with the ovsdb and OpenFlow southbound APIs of the SDN controller to deploy network configurations from the framework. The configurations are selected with the help of reinforcement learning and are deployed in real-time. As shown in Figure 1, the threat response consists of two sub-modules: bandwidth manager and flow rule update manager. Both of these sub-modules represent the actions which the reinforcement learning agent can take against a network threat. This module can be easily extended with more choices for actions against the network threat.

3.3.1 Bandwidth Manager

The bandwidth manager can be thought of as an interface which provides a southbound API to modify the bandwidth of the switch. The value of the bandwidth is determined by another reinforcement learning agent, making it possible for the system to scale the bandwidth up and down as needed. The state space and action space for this agent are not so different from the agent that determines whether to update bandwidth or update a flow rule. The bandwidth allocation agent takes as input the congestion score defined in the traffic analyzer section and selects the optimal bandwidth. The congestion score provides the estimate of the bandwidth usage to the agent. With the help of the bandwidth usage, the agent can determine whether the suggested bandwidth value was a profitable decision or a loss. The profit and loss here are used in the context of reward. If the suggested bandwidth maintains the optimal network operation, it is a positive reward. However, if the suggested bandwidth does not affect the network congestion much, it is a negative reward.

The same learning paradigm helps the reinforcement learning agent to scale up or scale down the bandwidth depending on the network usage on a normal basis. This feature allows the agent to work as a resource allocation algorithm as well. The agent is given a set of values which can be deployed as the network bandwidth, and each one of these values becomes an action. Since the state space and action space have been defined, the agent learns the value of each corresponding action over time and determines the relationship between the actions and states, which maximizes the reward given a particular state.

There are many network attacks which can be countered by simply modifying the network bandwidth to some extent. There certainly is a large number of attacks which can exhaust the bandwidth even after scaling, but for them, the other action deals with the protocol-specific traffic rates in the network.

3.3.2 Flow Rule Manager

The flow rule manager is another interface which provides a southbound API to update flow rules on the switch. This API sends an OpenFlow message to the particular switch to indicate the update in the flow table. With the help of this API, the framework can add, modify, and remove rules corresponding to various packets. There are many cases wherein just updating the bandwidth of the network does not solve the problem. For example, in the case of HTTP GET Request Flood, it is not the bandwidth which is exhausted in the network but the resources of the victim server. The action could be to limit or stop the incoming HTTP traffic to a particular switch or host in the network.

The learning switch deployed with the help of the SDN controller has an inbuilt add_flow API to make changes in the flow table. The parameters include a match object to a protocol, the specific actions, and the datapath of the switch. The framework checks for the protocol-specific information which is gathered in the traffic analyzer section. The protocol-specific features help to determine which flow rule needs to be deployed on the corresponding switch. The protocol having a higher density in terms of bandwidth and packets is the one which needs to be moderated.

3.4 Layer-wise grouping of DoS attacks

Denial-Of-Service (DoS) attacks focus on disrupting the victim service or the victim network for the legitimate users or legitimate hosts, respectively. These attacks exploit the features of different protocols and misuse them, and can be classified based on the various layers of the TCP/IP model as follows:

3.4.1 Attacks on the IP Layer

A basic network attack targeting the IP layer is MAC flooding. This attack involves flooding the switch with data packets, which draw out the legitimate MAC address and imitate a unicast behavior by sending traffic to the areas of network where it is not intended to go. The methods to mitigate this attack might involve limiting the number of MAC addresses learned through a particular port, authentication of MAC addresses, etc. Another attack on this layer involves ICMP flooding. This attack is a volumetric attack which focuses on overloading the network bandwidth. The adversary floods the network with ICMP echo requests until the server or the machine exhausts all the resources in replying to the message. A typical way to mitigate this attack is by filtering ICMP echo requests or rate limiting them. It is also known as the Smurf Attack.

3.4.2 Attacks on the Transport layer

The attacks on the transport layer exploit the TCP and UDP protocols. Most of the communication is carried out through the transport layer, as many of the applications use TCP/UDP in the underlying process. The two main attacks on the transport layer are the TCP SYN Flood attack and the UDP Flood attack.

SYN Flood: The TCP protocol is a connection oriented protocol and this connection is facilitated through a three-way handshake. SYN flood attack exploits the three-way handshake to hinder the services provided by any server by sending spoofed or fake SYN requests to the server. Upon receiving a SYN request, a network server would simply acknowledge it and wait for a reply. However, in case of a SYN flood, the reply never arrives, thereby hindering the functionality of the server. This attack can be mitigated by limiting the number of SYN requests, filtering the SYN requests, or even a timer to SYN RECEIVED.

UDP Flood: As the name suggests, this attack involves sending a huge number of UDP packets to the victim. The victim cannot process the large influx of packets all at once and hence remains occupied. Because of this, it cannot process the legitimate traffic or requests. The victim would simply reply with the destination host unreachable ICMP reply, as the packet would be spoofed. This attack can be mitigated with the help of appropriate firewalls.

3.4.3 Attacks on the Application Layer

The attacks on the application layer exploit the vulnerabilities in the applications, such as an HTTP server, DNS and Ping. Each application has some procedure to deal with communication, whether it is in request and response form or echo and reply form. The same procedure which allows the applications to work perfectly, is also the one which leads to exploitation.

HTTP Flood Attacks: As the name suggests, these attacks involve sending a large number of HTTP POST requests to the server. By nature, this attack is a resource consumption attack, as the victim cannot process the large number of POST requests in time, resulting in high utilization of system resources or a crash in the worst case. Similarly, one could achieve the same goal with the help of an HTTP GET Flood, which involves the influx of a large number of HTTP GET requests. A reasonable way to mitigate this attack is by monitoring the application which is accepting such requests.

NTP Amplification Attack: NTP is a UDP based protocol which can be exploited to return a large reply to a small request. Using the same property, a resource consumption attack can be launched against a server. The NTP protocol has a “monlist” command which returns the total number of servers which the NTP server has contacted. Given a number of open NTP servers, one could easily accomplish an amplification attack resulting into a DDoS attack. This attack is an amplification attack, as the response packet size is much larger than the request packet size. A typical way to mitigate this attack is by securing the NTP client.

DNS Amplification Attack: A DNS server upon receiving a "ANY" request returns all information about that host to the source IP address, thereby making the response packet size much larger than the request packet size. In a DNS amplification attack, the adversary spoofs DNS lookup requests with the IP address of the victim and sends "ANY" requests to open DNS servers. All these servers would then respond by sending out all the information they know about the requested IP address, and this information would be sent to the victim. This attack can be mitigated by reducing the number of recursive resolvers. This work covers ICMP flooding, TCP SYN flooding, UDP flooding, HTTP GET request flood, and DNS amplification attack. The performance of the framework with respect to each attack scenario is discussed in the next chapter.

CHAPTER 4

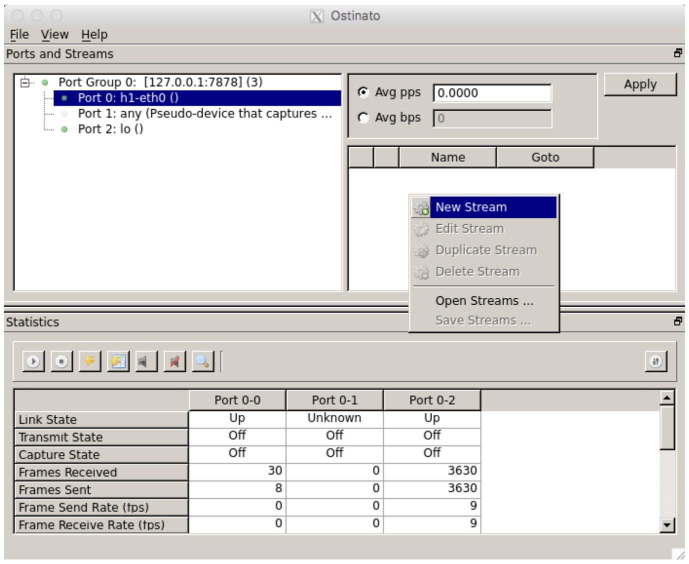
Evaluation

The proposed framework is evaluated by deploying it with a network topology. An SDN controller can best be tested by allowing it to handle the operations for which it was developed. In the same way, the proposed framework can be best tested by allowing it to protect a network against attacks. The network topology can be simulated with the help of mininet [32]. It is a network simulation tool that makes use of Linux containers to simulate switches and hosts on the system.

The other aspect to testing is simulating normal network traffic. In case of a real network, there are various users using their systems to fetch different sorts of information. For testing purposes, it is possible to simulate usual or normal traffic using traffic generators. A traffic generator is a tool that takes in the type of packets to generate, the number of packets to generate, and the destination IP addresses. One such traffic generation tool is Ostinato [33]. It can be used through the GUI or the Python API. The following steps show the working of the tool and the equivalent actions on the GUI are demonstrated in Figure 2, Figure 3, Figure 4, Figure 5, and Figure 6.

1. Create a new stream under the interface.
2. Select the Layer 3 protocol version and Layer 4 protocol type.
3. Enter the custom protocol data. [If needed]
4. Enter the total number of packets to send, that is, the number of packets per second.
5. Choose the source and destination IP addresses; it is possible to make them both random.

6. Apply the stream on the port and start it.

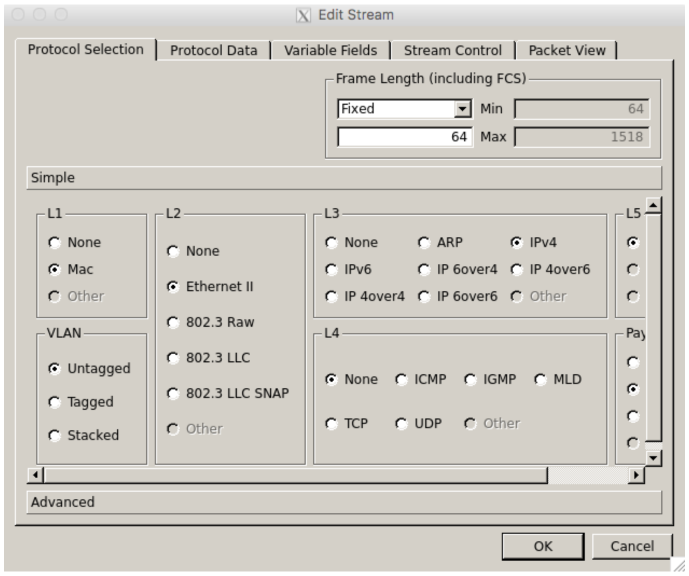


Port Group 0: [127.0.0.1:7878] (3)
Port 0: h1-eth0 ()
Port 1: any (Pseudo-device that captures ...
Port 2: lo ()

(●) Avg pps 0.0000
( ) Avg bps 0

	Port 0-0	Port 0-1	Port 0-2
Link State	Up	Unknown	Up
Transmit State	Off	Off	Off
Capture State	Off	Off	Off
Frames Received	30	0	3630
Frames Sent	8	0	3630
Frame Send Rate (fps)	0	0	9
Frame Receive Rate (fps)	0	0	9

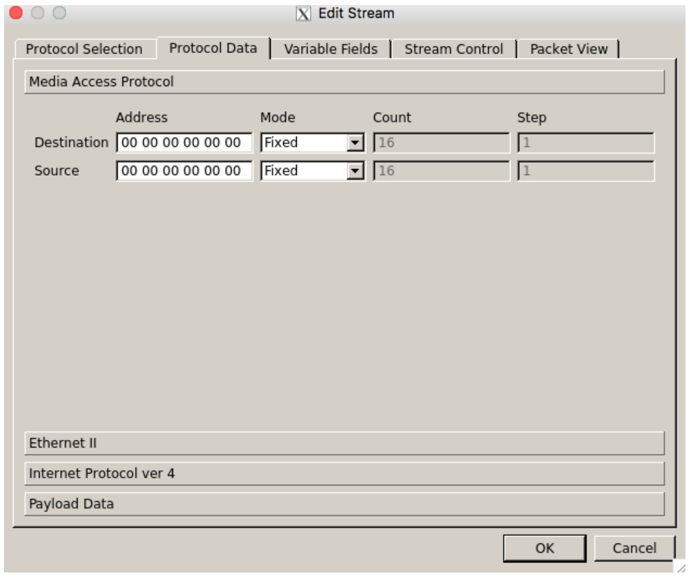
Figure 2 – Adding a new stream to Ostinato



Frame Length (including FCS)
Fixed 64 Min 64 Max 1518

L1: ( ) None [✓ Mac] ( ) Other
VLAN: [✓ Untagged] ( ) Tagged ( ) Stacked
L2: ( ) None [✓ Ethernet II] ( ) 802.3 Raw ( ) 802.3 LLC ( ) 802.3 LLC SNAP ( ) Other
L3: ( ) None ( ) ARP [✓ IPv4] ( ) IPv6 ( ) IP 6over4 ( ) IP 4over6 ( ) IP 4over4 ( ) IP 6over6 ( ) Other
L4: [✓ None] ( ) ICMP ( ) IGMP ( ) MLD ( ) TCP ( ) UDP ( ) Other
L5
Pay

Figure 3 – Configuring the stream with L3 and L4 protocols


Edit Stream

Protocol Selection | Protocol Data | Variable Fields | Stream Control | Packet View

Media Access Protocol

	Address	Mode	Count	Step
Destination	00 00 00 00 00 00	Fixed	16	1
Source	00 00 00 00 00 00	Fixed	16	1

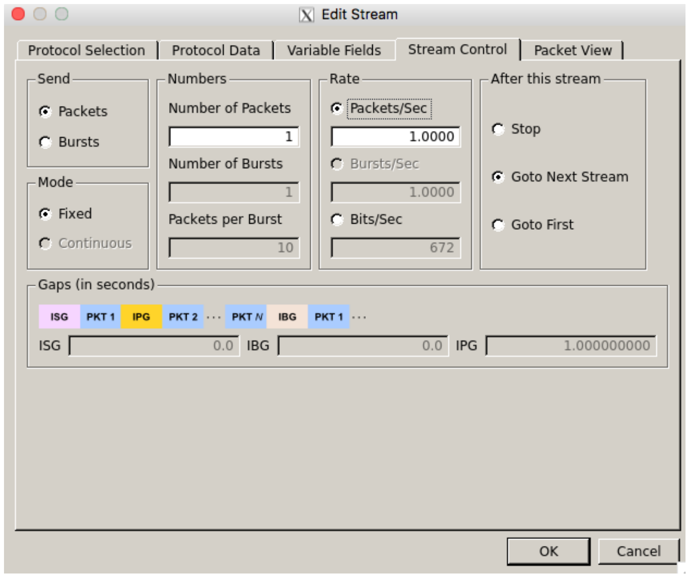
Ethernet II

Internet Protocol ver 4

Payload Data

OK Cancel

Figure 4 – Configuring the protocol data


Edit Stream

Protocol Selection | Protocol Data | Variable Fields | Stream Control | Packet View

Send: [✓ Packets] Bursts

Mode: [✓ Fixed] Continuous

Numbers:
- Number of Packets: 1
- Number of Bursts: 1
- Packets per Burst: 10

Rate:
- [✓ Packets/Sec] 1.0000
- Bursts/Sec 1.0000
- Bits/Sec 672

After this stream: Stop [✓ Goto Next Stream] Goto First

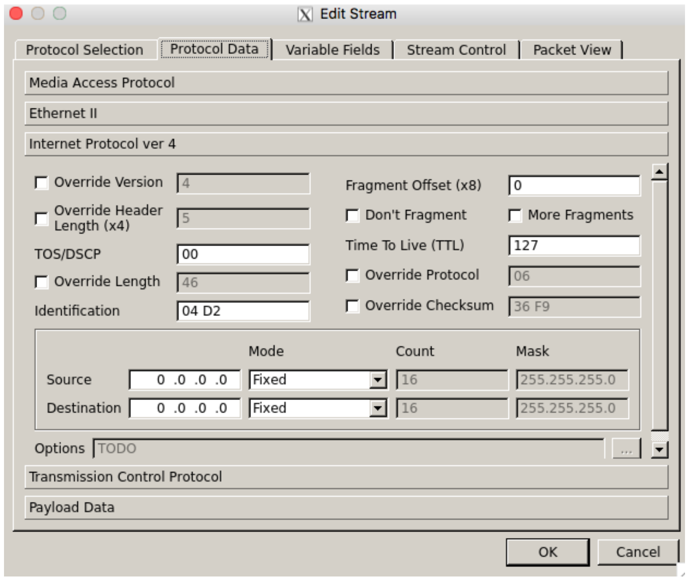
Gaps (in seconds)

ISG PKT 1 IPG PKT 2 ··· PKT *N* IBG PKT 1 ···

ISG 0.0 IBG 0.0 IPG 1.000000000

OK Cancel

Figure 5 – Configuring the values in the stream



☐ Override Version: 4
☐ Override Header Length (x4): 5
TOS/DSCP: 00
☐ Override Length: 46
Identification: 04 D2
Fragment Offset (x8): 0
☐ Don't Fragment
☐ More Fragments
Time To Live (TTL): 127
☐ Override Protocol: 06
☐ Override Checksum: 36 F9

		Mode	Count	Mask
Source	0 .0 .0 .0	Fixed	16	255.255.255.0
Destination	0 .0 .0 .0	Fixed	16	255.255.255.0

Options: TODO

Figure 6 – Configuring the source and destination for the packets

4.1 Generating the Attacks

The framework is evaluated against UDP flood, TCP SYN flood, ICMP flood, HTTP GET request flood, and DNS amplification attack. These attacks are some of the most well-known DoS attacks, and despite the simplicity, each attack is prevalent. The first step is to isolate the adversary machine. For evaluating this framework, a host from the available hosts is selected from the simulated topology, and the IP address of the host is changed so that it belongs to a different subnet. The host IP address can be preset if the mininet topology is created using the Python API or it can be set using the *ifconfig* and *netmask* commands as follows:

h1 ifconfig <interface> <new IP address> netmask <appropriate netmask>

The above example can be used with a host *h*1 in mininet. The new IP address can be set to 10.0.1.1 from 10.0.0.1 by using a different netmask, per the need. The adversary has been identified and so has the normal traffic flow. The next step is to generate the DoS attacks. These attacks involve a large number of packets, and there are many sophisticated tools with which one can carry them out for research purposes. These tools involve hping3 [34], Metaspolit [35], and Scapy [36]. A simple UDP attack can also be implemented by using Python socket programming APIs. The TCP SYN flood, UDP flood, and ICMP flood were generated with the help of hping3 tool. The following command can be used to deploy a TCP SYN flood attack on the victim of choice:

```
hping3 -S -P -U --flood -V --rand-source <victim-IP/URL>
```

By default, this command would run infinitely, but the number of packets can be limited by using the *-c* option and specifying the total number of packets to send. The port can be specified as well with the *-p* option of the command. One more attack that can be carried out using hping3 is the UDP flood attack. It is possible to specify the bytes per packet using the *-d* option followed by the size of each packet in bytes. The following command carries out a UDP flood on port 53.

```
hping3 -udp -p 53 -flood-rand-source <victim-IP/URL>
```

An alternate way to carry out a DoS attack is by wrapping up the hping3 system call in the Python programming language and setting up a loop against different port numbers. This attack would be difficult to manage, as multiple threads would be flooding the network. The ICMP attack can also be generated with the help of hping3. There are ICMP attacks with various codes, and for the testing of this framework it is ICMP echo flood. The following command is used to generate the flood:

```
hping3 --icmp -C 8 -K 0 --flood <victim-IP/URL>
```

The ICMP code and ICMP type are specified using the *-K* and *-C* options in the attack. The HTTP GET request flood and DNS amplification attacks were simulated with the help of Python scripts. Using BaseHTTPServer module in Python, it is very easy to create a victim server that can respond to GET requests. With the help of an infinite loop or multiple threads, one can send a large number of HTTP GET requests with spoofed IP addresses. The DNS amplification attack is carried out with an automated Python script as well. A large list of domain names to be resolved is prepared for the DNS server, and each one of the requests is sent through the script by spoofing the source address as the victim's IP address.

4.2 Network Topologies

The reinforcement learning agent learns through experience with the computer network. The goal is to make the agent understand the various scenarios of network attacks, so that it can learn to take complex decisions in real-time. This section explains the testbeds used for evaluating the framework. These testbeds are simulated with the help of mininet [32].

Figure 7 showcases a very simple network topology generated with the help of mininet [32]. This simple topology is a linear topology of a network consisting of one switch and three hosts connected to the switch. It is a learning switch facilitated with the help of OpenFlow [1] protocol, and simulated with the help of OpenVSwitch. The initial evaluation of any framework is easier with the help of a small scale network topology instead of a large scale one. The small scale network topology allows for easy debugging in terms of an error, and the results can be seen immediately. Also, with a smaller number of hosts connected to the switch, the network monitoring becomes simple as well. Considering the perspective of reinforcement learning agent, the agent first learns the complexity of a small scale environment, thereby making it possible to derive a conclusion whether or not it is feasible to deploy the agent on a relatively complex network topology. The results obtained with this network topology align with this view.

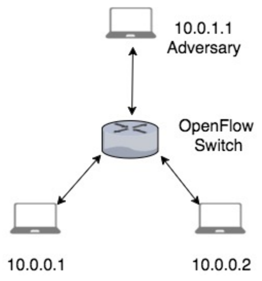


Figure 7 – A linear network topology with 3 hosts and one switch

Another viewpoint is, when the agent is first deployed and tested with a simple network topology, it is possible to also make a conclusion on how well the framework scales with a more complex network topology. Figure 8 shows a linear topology with six switches and twelve hosts. It allows for the reinforcement learning agent to be a part of a more diverse environment. With an increase in the number of hosts, the normal network usage increases as well. The first learning curve for the agent is to accommodate itself with the change in the environment. The next section presents the experimental results obtained with the help of the proposed framework.

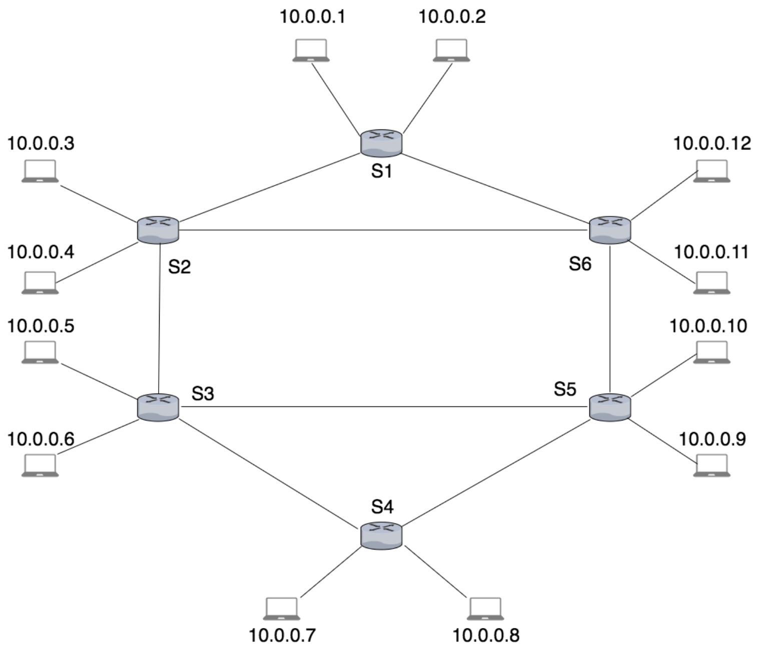


Figure 8 – A more complex linear network topology with six switches and twelve hosts

4.3 Experimental Results

The framework was tested iteratively against the different DoS attacks. This stepwise training of the reinforcement learning agent helps build a more robust understanding of various reactive actions in various scenarios. At first, the agent was trained against a UDP flood DoS, and then the other type of DoS attack was simulated on the network. This new attack would appear to be a zero-day attack to the agent, and it would adjust the value for each reactive action to counter the attack. This process was repeated with a set of five DoS attacks to evaluate the overall performance of the framework.

The framework was evaluated on the basis of percentage packet loss for an intended user of the service or a legitimate request made to the victim host. A basic UDP server was deployed on the victim host, and the client was deployed on one of the benign hosts to calculate the packet loss. The client tracks the total number of packets sent and received from the server. For every packet that the server does not respond to, it accumulates the packet loss with respect to the total number of packets sent. It is evident from the results obtained, that the framework reduces the packet loss by at least 70-80% as compared to the test results on RYU SDN controller without using the framework.

4.3.1 UDP Flood Attack

As the name suggests, a UDP flood attack douses a victim server with UDP packets. For the purpose of this attack on both testbeds, a simple UDP Server was set up on the victim host. The flood attack was carried out with the help of hping3 tool. The flood size determines the number of UDP packets which were used in the attack. For example, an attack of 500 Megabytes indicates a total of 5000000 packets sent, each having a size of 100 bytes. Figure 9 and Figure 10 show the results obtained on testbed 1 and testbed 2 respectively.

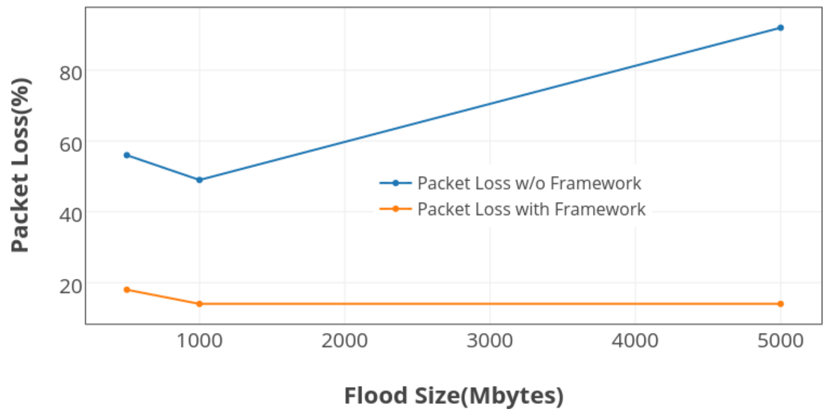


Figure 9 – Packet loss with and without framework in case of UDP flood on testbed 1

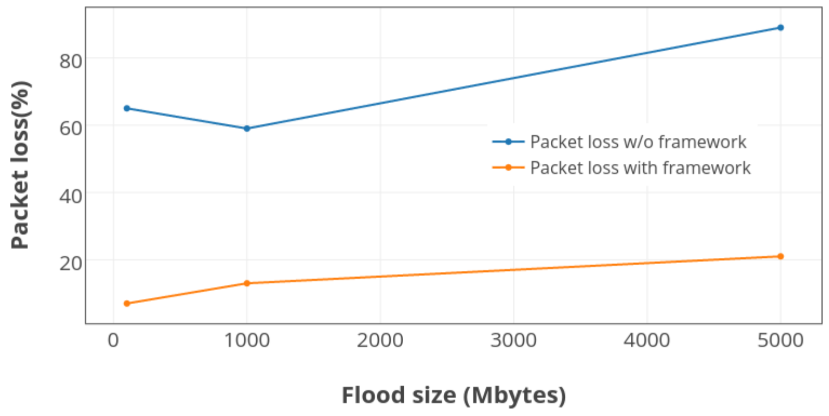


Figure 10 – Packet loss with and without framework in case of UDP flood on testbed 2

It is evident from the results that the packet loss is quite high when the framework is not deployed in the network. However, once the framework is deployed, the SDN Controller manages to thwart the attack to a great extent. Even though there is some packet loss after that too, it is not as significant as without using the framework.

4.3.2 TCP SYN Flood Attack

Next, the reinforcement learning agent or rather the framework was tested with a TCP SYN flood attack. Since it was the first time the framework comes across a TCP SYN flood attack, it could be considered as a zero day attack. The reinforcement learning agent does learn how to tackle it, and the number of interactions it took to do so are depicted in the latter part of this chapter. However, the motive of training the framework like this was to inspire the reinforcement learning agent to explore more actions. The TCP SYN flood attack is focused more on exhausting the resources of the victim host rather than exhausting the bandwidth. Therefore, an appropriate action in this case would be to moderate the incoming TCP SYN Requests.

This attack focuses on sending spoofed TCP SYN packets to the victim. It exploits the three-way TCP Handshake. Since the IP addresses are spoofed, the handshake is never completed and the victim keeps waiting for the sender to respond. In this way, the resource exhaustion is achieved in a TCP SYN flood attack. Figure 11 and Figure 12 show the packet loss with and without the framework on testbed 1 and testbed 2, respectively.

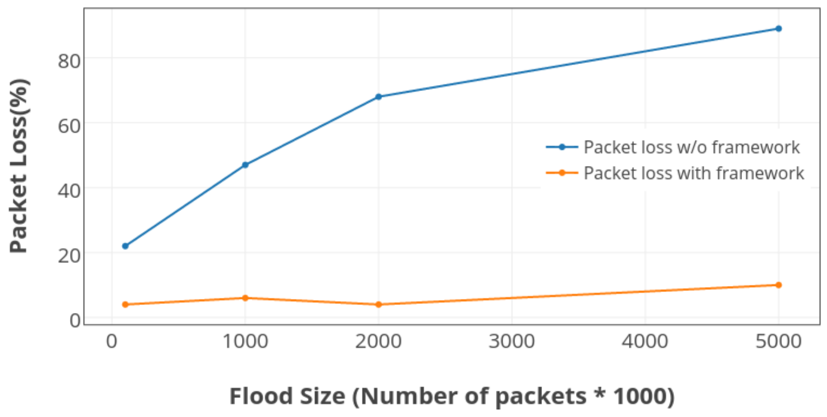


Figure 11 – Packet loss with and without framework in case of TCP SYN flood on testbed 1

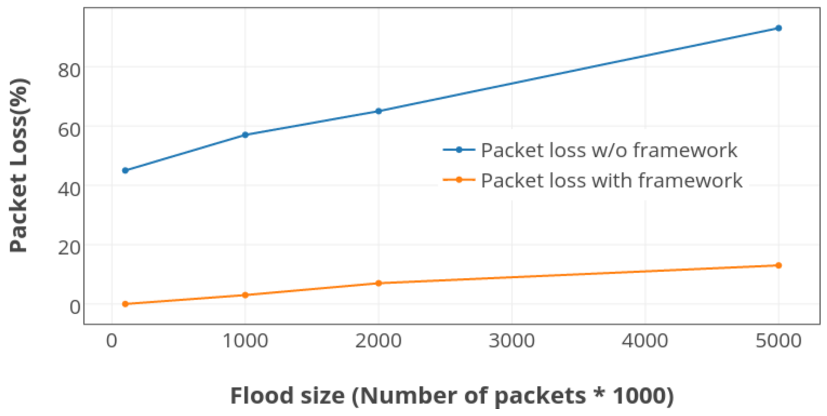


Figure 12 – Packet loss with and without framework in case of TCP SYN flood on testbed 2

Packet loss is measured as the number of legitimate TCP SYN requests being refused as the victim is too busy. This attack is simulated using hping3 tool and the legitimate TCP SYN packets are generated through a Python script, which also keeps an account for the packet loss.

4.3.3 ICMP Flood Attack

Ping application is used to check if there is an active host for the corresponding IP address or URL. The same utility is exploited to send a multitude of ICMP requests that force the victim to respond with an ICMP packet. It is a classic resource consumption attack wherein the victim server becomes too busy in handling all the incoming ICMP requests. However, an ICMP packet does not have a higher priority such as HTTP or TCP, so it might get ignored. The main targets of these types of flood attacks are the switches because a switch is responsible to forward all such packets to the respective hosts. Figure 13 and Figure 14 show the results obtained on both testbeds under an ICMP flood attack.

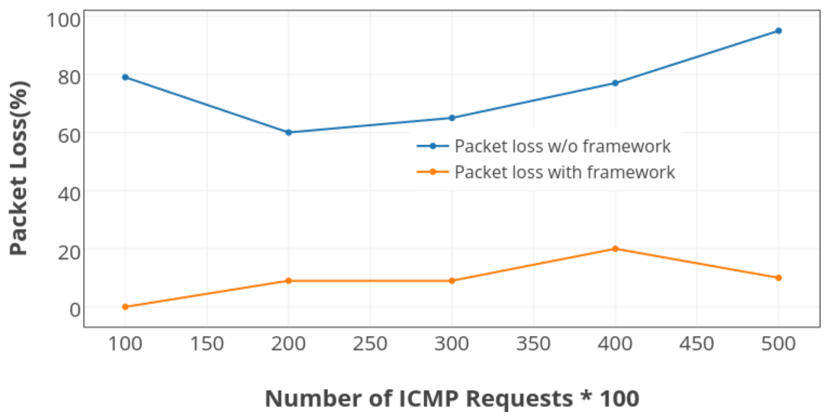


Figure 13 – Packet loss with and without framework in case of ICMP/Ping flood on testbed 1

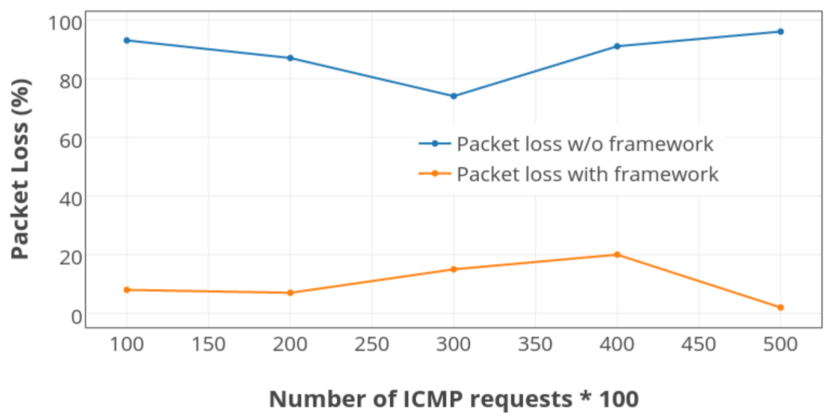


Figure 14 – Packet loss with and without framework in case of ICMP/Ping flood on testbed 2

The goal of ICMP flood attack might be to disconnect the victim from the network by making it busy. Once the victim is busy enough, it might not respond or collect information necessary to stay connected in the network. The reinforcement learning agent iteratively learns the type of attack and also learns the optimal action necessary to counter it.

4.3.4 HTTP GET Request flood Attack

The HTTP GET Request flood attack is an application layer DoS attack targeted on an HTTP Server. The target is to flood the server with enough fake requests that it cannot handle legitimate requests. All different kinds of DoS attacks share essentially the same goal. The packet loss is taken into account as the number of legitimate HTTP GET Requests that were refused service. Figure 15 and Figure 16 showcase the results on testbed 1 and testbed 2 with and without the framework respectively.

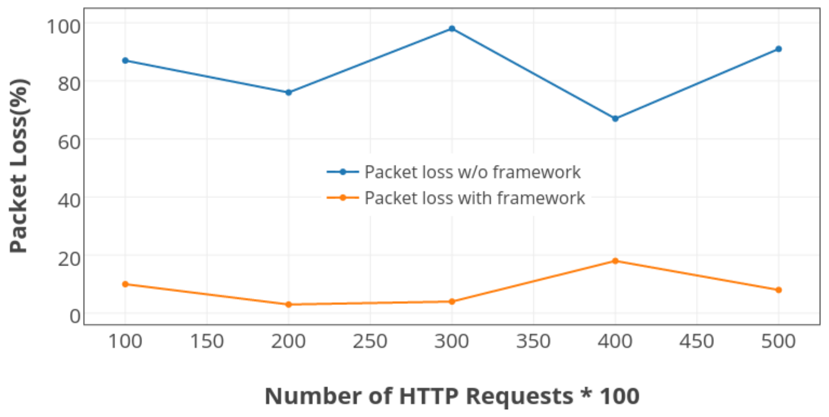


Figure 15 – Packet loss with and without framework in case of HTTP GET Request flood on testbed 1

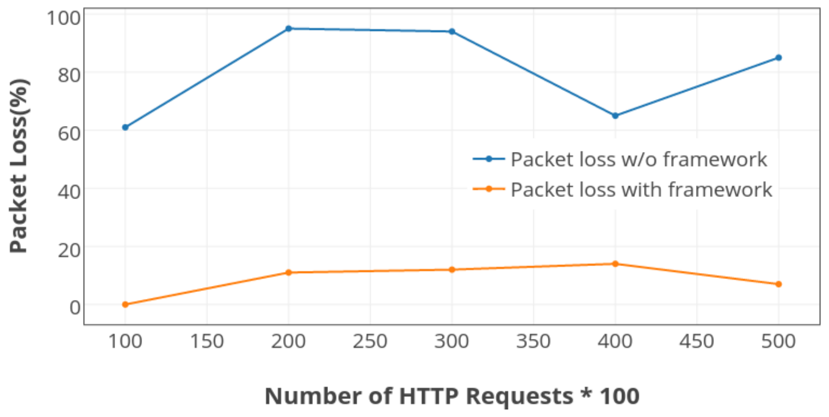


Figure 16 – Packet loss with and without framework in case of HTTP GET Request on testbed 2

4.3.5 DNS Amplification Attack

The last attack which the reinforcement learning agent is trained against is a DNS Amplification Attack. In this attack the adversary sends spoofed DNS requests to DNS servers. The DNS Servers resolve the request and send their responses back to the origin. The origin address is the address of the victim. This attack can be carried out on a large scale as the number of Open DNS Servers available is high and most of these DNS Servers have a very high bandwidth. It might be perceived as though these DNS servers are executing a DDoS attack on the victim.

However, the DNS protocol does contain a UDP packet underneath it. These UDP packets can be regulated in the network in order to thwart the amplification attack. This attack is not focused only on bandwidth exhaustion but also on resource exhaustion for the victim. Whenever the victim receives a DNS response it starts processing it, and with enough DNS responses, the victim would be so busy processing these responses that the more important traffic might be overlooked.

A simple Python script is used to simulate a DNS amplification attack. The packet loss is accounted for by using simple UDP Client and server programs across the network. Figure 17 and Figure 18 show the results with the help of the framework on testbeds 1 and 2, respectively.

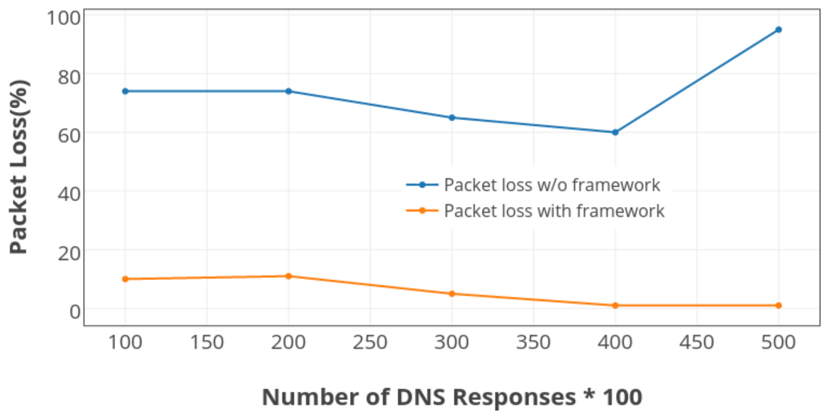


Figure 17 – Packet loss with and without framework in case of DNS Amplification on testbed 1

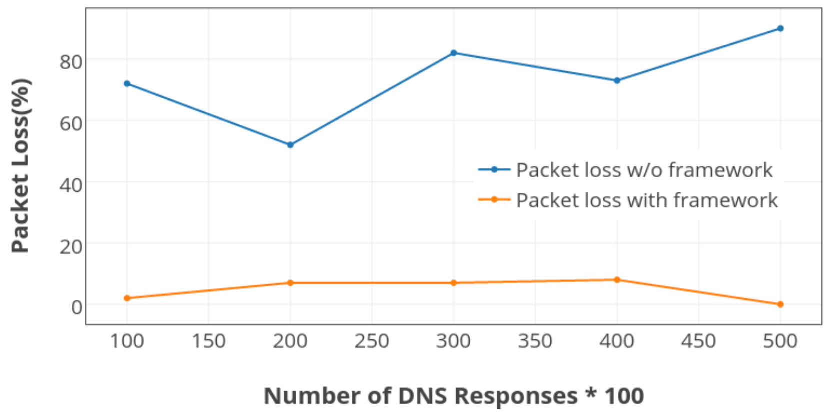


Figure 18 – Packet loss with and without framework in case of DNS Amplification on testbed 2

Apart from the iterative training of the reinforcement learning agent on various types of DoS attacks, the other aspect of evaluation is how quickly the reinforcement learning agent learns to select the optimal or the right action as per the standard strategy. For example, the standard strategy for a UDP flood might be increasing the bandwidth, but for a UDP flood with a higher intensity, it would be to moderate UDP traffic using a flow rule.

Figure 19 shows the number of interactions the reinforcement learning agent made before choosing the correct action. Each attack is repeated ten times to get an estimate of the learning curve for the reinforcement learning agent. There is a lot of variance in the number of interactions made, and the reinforcement learning agent does not take too many interactions as well. The main reason for it is the number of actions in the framework; at present, the framework supports two major actions. The first one is bandwidth management in the computer network and the other one is flow rule management. There are different flow rules under the flow rule management action, and each of these flow rules are selected by looking at the protocol-specific features obtained during traffic classification. It is possible that, with the increase in the number of actions, the reinforcement learning agent takes more interactions to arrive at the correct action for the particular scenario.

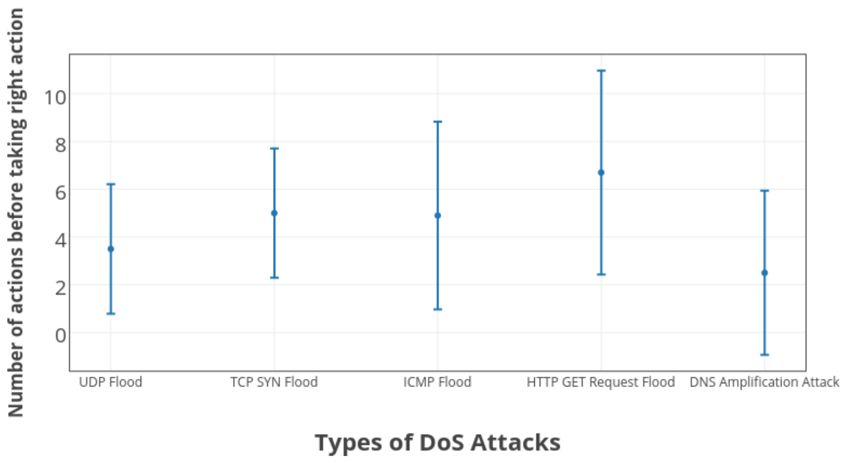


Figure 19 – The learning curve of the agent w.r.t. various network attacks

The last test which was carried out on the framework was one of bandwidth allocation. It is possible that at some point in time, the computer network is busier than usual. Maybe all the users were streaming live videos or playing games. This framework was developed with a major focus on handling different types of network attacks, but it also has an additional feature of bandwidth management. It can be configured to act like a resource allocation agent as well. Table 1 shows the various values of the bandwidth suggested by the reinforcement learning agent under different network congestion scenarios.

Table 1 – Evaluating Optimal Bandwidth Allocation

Case	Network Congestion(%)	Suggested Bandwidth(Mbps)
1	99.98	92-96
2	85.47	79-83
3	50.32	41-43
4	10.39	4-7

It is clear from all the results that were obtained, that the framework works reasonably well against various DoS attacks. However, there were challenges faced during the process to obtain such results. These challenges include simulation techniques for the various attacks to the deployment of the framework and the learning of the reinforcement learning agent. Overall, the agent counters the network attacks to a greater extent. Hence, it gives sufficient background on further implementation of an intelligent network security module with SDN.

CHAPTER 5

Discussion

The goal of this framework is to protect the network from DoS attacks. These attacks can target the hosts and/or the switches that make up the network. The controller handles topology management, provides an interface to collect switch statistics, and uploads appropriate flow rules on the switches. With the same features and an added southbound interface to push changes in the switch configurations, the framework protects the entire network against attacks. The framework forms a successful relationship between the network behavior and the reactive action space which helps to counter various network attacks.

5.1 Memory Usage Comparison

The controller loads the reinforcement learning agent along with the analyses modules in memory to make a decision against network attacks. The decision specifies the reactive action which needs to be taken for thwarting the network threat or stabilizing the network operation. The reactive actions are deployed on the switches individually. Thus, it is important to evaluate memory usage of the controller with and without the framework. There is an increase in memory usage as the framework loads the reinforcement learning agent and the tshark [30] library in memory. However, the usage stays constant even with an increase in the number of switches connected to the controller. Figure 20 shows the memory usage comparison of the controller with and without the framework.

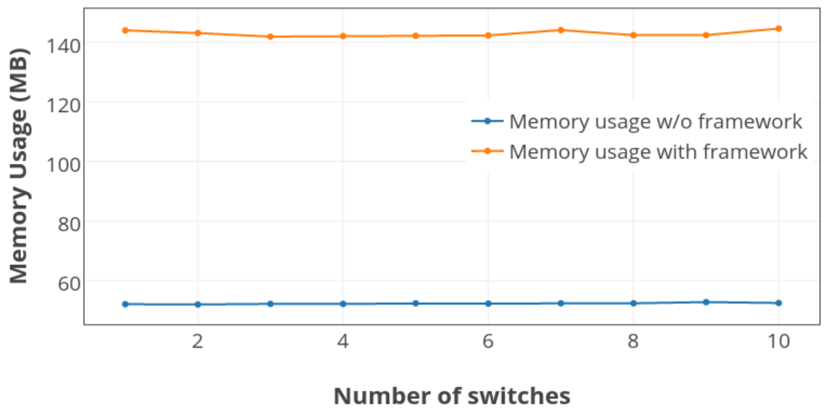


Figure 20 – Comparison of memory usage

The memory usage statistics were obtained using the "top" terminal utility command. The graph shows the physical memory used by the controller when it is deployed in network topologies with a different number of switches. Each one of these switches is monitored for collecting statistics, and the reactive actions are pushed switch-wise; thus the memory usage is taken with reference to the number of switches in the network. The increasing number of switches does not have an impact on the memory used by the framework. The memory used by the framework is nearly constant as the number of switches increases. The difference in memory usage with and without the framework suggests that the reinforcement learning agent needs a certain amount of memory to function. However, it is a memory difference which can be accommodated by the present day switches with ease.

The experimental results discussed in Chapter 4 prove that the congestion score and other protocol-specific features calculated in the traffic analyzer module are effective representations of the network behavior. The effectiveness is proven through the actions taken by the reinforcement learning agent. There is no accuracy which can be measured in terms of such an agent, but the impact on the network turns out to be a valuable metric in its performance. The frequency with which the agent arrives at the correct action shows the exploration vs. exploitation concept in real-time. The actions selected by the agent are executed in less than one second. Overall, the proposed framework asserts that with reinforcement learning the capabilities of SDN can be increased significantly, especially in the area of network security.

From an implementation perspective, RYU [15] is a component-based, event-driven controller which allows for fast prototyping. The main advantage of using RYU is its smaller code density and modular implementation. The security measures can very well be integrated with an enterprise-grade controller, but the importance lies in evaluation of the framework's usefulness. This paper covers DoS attacks for evaluating the framework. However, there are many other network attacks that target software-defined networks. It is possible to develop a multi-agent reinforcement learning solution to cover different scenarios across the network. Other approaches as well as extensions to the framework are discussed in Chapter 6.

The reinforcement learning agent first starts off with a random action given the $\epsilon - greedy$ approach. It solves the cold start problem, wherein the agent does not have prior experience with the environment. With time, the agent accumulates experience for each reactive action and the understanding of network state to make an intelligent decision. The average number of iterations it takes from switching over to an intelligent action from a random action is three to four. However, the number of iterations depends on how well the random action worked for the network environment. If the action worked out well for the network, the value for that action would increase automatically. In the upcoming iterations, the likelihood of the reinforcement learning agent choosing that action increases.

There are other approaches which can be used to train the agent. One of these approaches is to train the network in a live environment. The reinforcement learning agent can accumulate more information about the network behavior and take an even better action against a network attack, as the dynamics of a live environment are much different. However, in a simulated environment such as the one presented in this paper, the agent can acquire enough prior experience to handle the live network properly. The agent can also be trained with different network attacks to incorporate more actions in the action space.

CHAPTER 6

Conclusion and Future Work

This thesis proposed a framework that can autonomously protect the network against attacks. The framework does so by forming a successful non-linear relationship between the state space and the action space. The state space is the information collected to represent network behavior and the action space is the set of reactive actions defined to protect the network. The framework has a traffic analyzer module which calculates the state space and a threat response module that forms the action space. Both of these modules make use of the southbound interfaces and provide southbound APIs to monitor and protect the network. The reinforcement learning agent forms the non-linear relationship between network behavior and reactive action space through the experience gathered from the interactions with the network. With a large number of interactions, the agent accumulates enough information to even thwart unknown attacks on the network in the future. The iterative evaluation of the framework was carried out to estimate the behavior in case of a previously unknown attack. As shown by the results, the framework performs quite well in defending the network.

However, this implementation supports only the bandwidth management and traffic moderation through flow rule management action. More actions such as setting up VLANs, restricting the traffic specific to the port, and routing path changes can be incorporated to make an even more complete solution. Also, this framework can be extended by using other enterprise-grade SDN controllers. The framework prototype focuses on defending the network against DoS attacks; future work will focus on implementing a generic framework for all types of network attacks. It is hoped that this work would serve as an inspiration for other researchers to pursue intelligent security measures in software-defined networks and extend the same to an even higher level.

LIST OF REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, ‘‘Openflow: Enabling innovation in campus networks,’’ *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69--74, Mar. 2008. DOI: 10.1145/1355734.1355746. [Online]. Available: http://dl.acm.org/citation.cfm?id=1355746

[2] L. H. Newman, ‘‘What we know about friday’s massive east coast internet outage,’’ Oct 2016, as accessed on 04/05/2017. [Online]. Available: https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/

[3] D. Bisson, ‘‘The 5 most significant ddos attacks of 2016,’’ Nov 2016, as accessed on 04/05/2017. [Online]. Available: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/5-significant-ddos-attacks-2016/

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* Cambridge, MA: MIT press, 1998, vol. 1, no. 1. [Online]. Available: https://mitpress.mit.edu/books/reinforcement-learning

[5] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, ‘‘A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies,’’ *Knowl. Eng. Rev.*, vol. 21, no. 2, pp. 97--126, Jun. 2006. DOI: 10.1017/S0269888906000944. [Online]. Available: http://dl.acm.org/citation.cfm?id=1166054

[6] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, ‘‘Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning,’’ in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug 2005. DOI: 10.1109/IROS.2005.1545025. ISSN 2153-0858 pp. 3712--3717. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/1545025/

[7] ‘‘Software-defined networking: The new norm for networks,’’ Open Networking Foundation, Apr. 13 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[8] A. Servin and D. Kudenko, ‘‘Multi-agent reinforcement learning for intrusion detection,’’ in *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning: 5th, 6th, and 7th European Symposium, ALAMAS*

2005-2007 on Adaptive and Learning Agents and Multi-Agent Systems, Revised Selected Papers, K. Tuyls, A. Nowe, Z. Guessoum, and D. Kudenko, Eds. Berlin, Heidelberg: Springer, 2008, pp. 211--223. ISBN 978-3-540-77949-0. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-77949-0_15

[9] N. Sengupta and J. Sil, ''Comparison of supervised learning and reinforcement learning in intrusion domain,'' in *Wireless Networks and Computational Intelligence: 6th International Conference on Information Processing, ICIP 2012, Bangalore, India, August 10-12, 2012. Proceedings*, K. R. Venugopal and L. M. Patnaik, Eds. Berlin, Heidelberg: Springer, 2012, pp. 546--551. ISBN 978-3-642-31686-9. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-31686-9_63

[10] K. Malialis and D. Kudenko, ''Distributed response to network intrusions using multiagent reinforcement learning,'' *Eng. Appl. Artif. Intell.*, vol. 41, pp. 270 -- 284, 2015. DOI: 10.1016/j.engappai.2015.01.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095219761500024X

[11] N. Sengupta, J. Sen, J. Sil, and M. Saha, ''Designing of on line intrusion detection system using rough set theory and q-learning algorithm,'' *Neurocomputing*, vol. 111, pp. 161 -- 168, 2013. DOI: 10.1016/j.neucom.2012.12.023. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121300060X

[12] P. J. Chen and Y. W. Chen, ''Implementation of sdn based network intrusion detection and prevention system,'' in *2015 International Carnahan Conference on Security Technology (ICCST)*, Sept 2015. DOI: 10.1109/CCST.2015.7389672 pp. 141--146. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7389672/

[13] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, ''Avant-guard: Scalable and vigilant switch flow management in software-defined networks,'' in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013. DOI: 10.1145/2508859.2516684. ISBN 978-1-4503-2477-9 pp. 413--424. [Online]. Available: http://dl.acm.org/citation.cfm?id=2516684

[14] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, ''Flowguard: Building robust firewalls for software-defined networks,'' in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014. DOI: 10.1145/2620728.2620749. ISBN 978-1-4503-2989-7 pp. 97--102. [Online]. Available: http://dl.acm.org/citation.cfm?id=2620749

[15] ‘‘Ryu sdn framework[software],’’ 2014. [Online]. Available: https://osrg.github.io/ryu/

[16] H. Padekar, Y. Park, H. Hu, and S.-Y. Chang, ‘‘Enabling dynamic access control for controller applications in software-defined networks,’’ in *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT ’16. New York, NY, USA: ACM, 2016. DOI: 10.1145/2914642.2914647. ISBN 978-1-4503-3802-8 pp. 51--61. [Online]. Available: http://dl.acm.org/citation.cfm?id=2914647

[17] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, ‘‘{DDoS} attack protection in the era of cloud computing and software-defined networking,’’ *Computer Networks*, vol. 81, pp. 308 -- 319, 2015. DOI: 10.1016/j.comnet.2015.02.026. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615000742

[18] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, ‘‘A sdn-oriented ddos blocking scheme for botnet-based attacks,’’ in *Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2014. DOI: 10.1109/ICUFN.2014.6876752. ISSN 2165-8528 pp. 63--68. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6876752/

[19] H. Wang, L. Xu, and G. Gu, ‘‘Floodguard: A dos attack prevention extension in software-defined networks,’’ in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015. DOI: 10.1109/DSN.2015.27. ISSN 1530-0889 pp. 239--250. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7266854/

[20] R. Braga, E. Mota, and A. Passito, ‘‘Lightweight ddos flooding attack detection using nox/openflow,’’ in *IEEE Local Computer Network Conference*, Denver, CO, Oct 2010. DOI: 10.1109/LCN.2010.5735752. ISSN 0742-1303 pp. 408--415. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5735752/

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, ‘‘Nox: Towards an operating system for networks,’’ *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105--110, Jul. 2008. DOI: 10.1145/1384609.1384625. [Online]. Available: http://dl.acm.org/citation.cfm?id=1384625

[22] Y. Feng, R. Guo, D. Wang, and B. Zhang, ‘‘Research on the active ddos filtering algorithm based on ip flow,’’ in *Fifth International Conference on Natural Computation*, vol. 4, Tianjin, China, Aug 2009. DOI: 10.1109/ICNC.2009.550. ISSN 2157-9555 pp. 628--632. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5363277/

[23] J. H. Jafarian, E. Al-Shaer, and Q. Duan, ‘‘Openflow random host mutation: Transparent moving target defense using software defined networking,’’ in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN ’12. New York, NY, USA: ACM, 2012. DOI: 10.1145/2342441.2342467. ISBN 978-1-4503-1477-0 pp. 127--132. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342467

[24] C. J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, ‘‘Nice: Network intrusion detection and countermeasure selection in virtual network systems,’’ *IEEE Trans. Dependable Secur. Comput.*, vol. 10, no. 4, pp. 198--211, July 2013. DOI: 10.1109/TDSC.2013.8. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6419708/

[25] U. Fiore, F. Palmieri, A. Castiglione, and A. D. Santis, ‘‘Network anomaly detection with the restricted boltzmann machine,’’ *Neurocomputing*, vol. 122, pp. 13 -- 23, 2013. DOI: 10.1016/j.neucom.2012.11.050 Advances in cognitive and ubiquitous computing. Selected papers from the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231213005547

[26] J. Cannady, ‘‘Next generation intrusion detection: Autonomous reinforcement learning of network attacks,’’ in *Proceedings of the 23rd National Information Systems Security Conference*, 2000, pp. 1--12. [Online]. Available: http://csrc.nist.gov/nissc/2000/proceedings/papers/033.pdf

[27] P. Wang, K. M. Chao, H. C. Lin, W. H. Lin, and C. C. Lo, ‘‘An efficient flow control approach for sdn-based network threat detection and migration using support vector machine,’’ in *IEEE 13th International Conference on e-Business Engineering (ICEBE)*, Nov 2016. DOI: 10.1109/ICEBE.2016.020 pp. 56--63. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7809901/

[28] R. Colbaugh and K. Glass, ‘‘Proactive defense for evolving cyber threats,’’ in *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, July 2011. DOI: 10.1109/ISI.2011.5984062 pp. 125--130. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5984062/

[29] X. Xu and Y. Luo, ‘‘A kernel-based reinforcement learning approach to dynamic behavior modeling of intrusion detection,’’ in *Advances in Neural Networks -- ISNN 2007: 4th International Symposium on Neural Networks, ISNN 2007, Nanjing, China, June 3-7, 2007, Proceedings, Part I*, D. Liu, S. Fei, Z.-G. Hou, H. Zhang, and C. Sun, Eds. Berlin, Heidelberg: Springer, 2007, pp. 455--464. ISBN 978-3-540-72383-7. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-540-72383-7_54?LI=true

[30] ‘‘tshark: Dump and analyze network traffic[software].’’ [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[31] E. Even-Dar and Y. Mansour, ‘‘Learning rates for q-learning,’’ *J. Mach. Learn. Res.*, vol. 5, pp. 1--25, Dec 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1005333

[32] B. Lantz, B. Heller, N. Handigol, and V. Jeyakumar, ‘‘Mininet: An instant virtual network on your laptop[software],’’ 2016. [Online]. Available: http://mininet.org

[33] P. Srivats, ‘‘Ostinato: Network traffic generator and analyzer[software],’’ N.D. [Online]. Available: http://ostinato.org

[34] S. Sanfilippio, ‘‘Hping - active network security tool[software],’’ 2006--2013. [Online]. Available: http://www.hping.org

[35] ‘‘Metasploit.’’ [Online]. Available: https://www.metasploit.com/

[36] ‘‘Scapy[software].’’ [Online]. Available: http://www.secdev.org/projects/scapy/