Distributed Reinforcement Learning for Network Intrusion Response

KLEANTHIS MALIALIS

Doctor of Philosophy

UNIVERSITY OF YORK
COMPUTER SCIENCE

September 2014

For my late grandmother Iroulla

Abstract

The increasing adoption of technologies and the exponential growth of networks has made the area of information technology an integral part of our lives, where network security plays a vital role. One of the most serious threats in the current Internet is posed by distributed denial of service (DDoS) attacks, which target the availability of the victim system. Such an attack is designed to exhaust a server's resources or congest a network's infrastructure, and therefore renders the victim incapable of providing services to its legitimate users or customers.

To tackle the distributed nature of these attacks, a distributed and coordinated defence mechanism is necessary, where many defensive nodes, across different locations cooperate in order to stop or reduce the flood. This thesis investigates the applicability of distributed reinforcement learning to intrusion response, specifically, DDoS response. We propose a novel approach to respond to DDoS attacks called Multiagent Router Throttling. Multiagent Router Throttling provides an agent-based distributed response to the DDoS problem, where multiple reinforcement learning agents are installed on a set of routers and learn to rate-limit or throttle traffic towards a victim server. One of the novel characteristics of the proposed approach is that it has a decentralised architecture and provides a decentralised coordinated response to the DDoS problem, thus being resilient to the attacks themselves.

Scalability constitutes a critical aspect of a defence system since a non-scalable mechanism will never be considered, let alone adopted, for wide deployment by a company or organisation. We propose Coordinated Team Learning (CTL) which is a novel design to the original Multiagent Router Throttling approach based on the divide-and-conquer paradigm, that uses task decomposition and coordinated team rewards. To better scale-up CTL is combined with a form of reward shaping. The scalability of the proposed system is successfully demonstrated in experiments involving up to 1000 reinforcement learning agents. The significant improvements on scalability and learning speed lay the foundations for a potential real-world deployment.

Contents

List of Tables

List of Figures

Acknowledgements

A big thank you to my supervisor Dr. Daniel Kudenko for his guidance and support throughout these four years, and for all the freedom he has given to me. It was a privilege working with you.

I wish to express my gratitude to my examiners Prof. John A. Clark and Dr. Michael Rovatsos for their invaluable feedback which has shaped and considerably improved my thesis.

Special thanks to Dr. Sam Devlin for the great collaboration, and to Logan Yliniemi and Prof. Kagan Tumer for their invaluable suggestions on Chapter 5. I am grateful to Gareth A. Lloyd for his great help; I learnt something new from you in every single conversation we have had. I would also like to thank Kyriakos Efthymiadis, Peter Scopes, Hanting Xie, Christos Evripidou, Simos Gerasimou and Yu Chan for all their help and the numerous fruitful discussions we have had. Last but not least, I would like to thank Dr. Paul Cairns for his help on statistics, Alex Cooper and David Hull for the technical support, and the Head of the Artificial Intelligence group, Dr. Alan M. Frisch, and the Department of Computer Science for the generous financial support.

I would like to thank my friends - you all know who you are - for all the good times.

I am very grateful to Koulla for her love, support and encouragement through the ups and downs of this journey. Thank you for your understanding.

Above all, I would like to thank my family, Andreas, Evita and Iro for their endless love and support. Nothing would have ever been possible without you.

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

Some of the material contained in this thesis has appeared in the following published or awaiting publication papers:

Journal Articles

1. K. Malialis and D. Kudenko, Distributed Response To Network Intrusions Using Multiagent Reinforcement Learning, In *Engineering Applications of Artificial Intelligence*. (In press)

2. K. Malialis, S. Devlin and D. Kudenko, Distributed Reinforcement Learning for Adaptive and Robust Network Intrusion Response, In *Connection Science*. (In press)

Conference Papers

1. K. Malialis, S. Devlin and D. Kudenko, Coordinated Team Learning and Difference Rewards for Distributed Intrusion Response, In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, Prague, Czech Republic, August 18-22, 2014, pp. 1063-1064.

2. K. Malialis and D. Kudenko, Multiagent Router Throttling: Decentralized Coordinated Response against DDoS Attacks, In *Proceedings of the 25th Conference on Innovative*

Applications of Artificial Intelligence (AAAI / IAAI), Bellevue, Washington, USA, July 14-18, 2013.

Workshop Papers

1. K. Malialis, S. Devlin and D. Kudenko, Intrusion Response Using Difference Rewards for Scalability and Online Learning, In *Proceedings of the AAMAS Workshop on Adaptive and Learning Agents (ALA)*, Paris, France, May 5-6, 2014.

2. K. Malialis and D. Kudenko, Large-Scale DDoS Response Using Cooperative Reinforcement Learning, In *11th European Workshop on Multi-Agent Systems (EUMAS)*, Toulouse, France, December 12-13, 2013.

3. K. Malialis and D. Kudenko, Reinforcement Learning of Throttling for DDoS Attack Response, In *Proceedings of the AAMAS Workshop on Adaptive and Learning Agents (ALA)*, Valencia, Spain, June 4-5, 2012, pp. 111-117.

CHAPTER 1

Introduction

This chapter introduces the topic of this thesis and describes the motivation of our research. It discusses the scope of the research and sets out the central research direction or hypothesis to be investigated. It then presents an overview of the thesis and discusses how the rest of the chapters are organised.

1.1 Motivation

The increasing adoption of technologies, the wide spread application of computers and the exponential growth of networks has made the area of information technology an integral part of our lives. This gave birth to numerous applications such as social networking, online shopping, smart metering and cloud computing where computer and network security plays a vital role.

A rapidly evolving threat and one of the most serious in the current Internet is posed by distributed denial of service (DDoS) attacks, which target the availability of a system (Mirkovic & Reiher 2004; Douligeris & Mitrokotsa 2004). A DDoS attack is a highly coordinated attack where the attacker takes under his control a large number of hosts, called the botnet (network of bots), which start bombarding the target when they are instructed to do so. Such an attack is designed to exhaust a server's resources or congest a network's infrastructure, and therefore renders the victim incapable of providing services to its legitimate users or customers.

Numerous incidents have been reported over the years against some high profile companies such as Yahoo, eBay, Amazon, Twitter, Facebook, CNN, Visa, MasterCard and PayPal (Greene 2011). Specifically, the Arbor Network's worldwide security survey (Anstee et al. 2013) conducted among more than 290 companies and organisations reveals that 50% of the participants see

1-10 DDoS attacks per month, while 12% experience more than 100. Furthermore, the survey reveals that the 62% of the participants have seen increased demand for DDoS protection from their customers. Beyond the financial loss caused by DDoS attacks, victims also suffer loss to their reputation which results in customer dissatisfaction and loss of trust. Common motivations are political and ideological hacktivism, vandalism, rivalries and extortion (Anstee et al. 2013).

As a result of the Internet's popularity and usefulness, there are many "interesting" targets and malicious (or simply ignorant) users that the DDoS threat is not going to disappear on its own (Keromytis et al. 2002). DDoS attacks grow in sophistication, frequency and size, and this demonstrates the need to adopt novel approaches to rapidly respond to DDoS attacks.

Machine learning (Mitchell 1997) is the study of how to construct computer agents that can improve their behaviour with experience. Machine learning is a multidisciplinary concept highly influenced by ideas from mathematics, psychology, biology, neuroscience and others.

As computing evolves, the overlapping connections, dependencies and interacting applications increase (Horn 2001). The need of learning new behaviours emerges from the fact that the complexity of the environment may not allow good behaviours to be pre-designed. Furthemore, in dynamic environments where the environment changes frequently over time, even a good behaviour may become inappropriate because it is static. Also, these complex applications require for administrative decision-making and responses faster than any human can deliver (Horn 2001). Machine learning distinguishes three different learning processes; these are, supervised learning, unsupervised learning and reinforcement leaning (RL) (Mitchell 1997).

Reinforcement learning is defined as the goal-directed learning from interaction between an active decision-making agent and its environment (Sutton & Barto 1998). The agent maps environmental input to local state information in discrete time steps. It then selects and executes an action. The environment presents the new state and also responds with a feedback, in the form of a positive or negative reward, evaluating the quality of the action performed. The agent finally updates its policy in such a way to optimise future rewards.

A multiagent system deals with the construction of a system involving multiple autonomous and interacting agents. Multiagent systems can offer many benefits including among others, better robustness to individual agent failures, better scalability, taking advantage of geographic distribution and sharing experiences for faster and better learning (Stone & Veloso 2000). Multiagent systems often need to be very complex, and multiagent reinforcement learning (MARL) is a promising candidate for dealing with this emerging complexity (Stone & Veloso 2000).

1.2 Scope and Hypothesis

The DDoS threat is challenging for many reasons, including the following (Mirkovic et al. 2003). Firstly, the traffic flows originate from host machines spread all over the Internet, which they all aggregate at the victim. Furthermore, the volume of the aggregated traffic is really large which is unlikely to be stopped by a single defence point near the victim. Also, the number of compromised host machines is large, thus making an automated response a necessary requirement. Moreover, DDoS packets appear to be similar to legitimate ones, since the victim damage is caused by the total volume and not packet contents. A defence system cannot make an accurate decision based on a packet-by-packet basis. It requires the defender to keep some statistical data in order to correlate packets and detect anomalies, for example, "all traffic directed towards a specific destination address". Lastly, it is very difficult to traceback, that is, to locate the botnet machines responsible for the attack, let alone discover the actual attackers who infected them.

It is evident that, to combat the distributed nature of these attacks, a distributed and coordinated defence mechanism is necessary where many defensive nodes across different locations cooperate in order to stop or reduce the flood.

There is an extensive literature regarding the application of machine learning to intrusion *detection*, specifically anomaly detection where no action is performed beyond triggering an intrusion alarm when an anomalous event is detected.

This thesis investigates the applicability of multiagent systems and machine learning to intrusion *response*. Specifically, the thesis investigates the applicability of multiagent reinforcement learning to respond to DDoS attacks. We focus on distributed rate-limiting and the research pursued in the thesis investigates the following hypothesis:

> Distributed reinforcement learning will create an automated and adaptive response, thus producing dynamic and robust behaviours to effectively mitigate the impact of DDoS attacks and improve recovery rates over existing state-of-the-art rate-limiting approaches.

1.3 Thesis Overview

Chapter 2 provides the background and literature review necessary for the reader to understand the contributions made by this work. Chapters 3 to 5 present the contributions made in the thesis. Chapter 3 provides the basis of this work, and the rest of the contributing chapters build upon their previous one. Each contributing chapter contains its own experimental evaluation. The thesis concludes in Chapter 6. Specifically, the organisation of the thesis is as follows.

Chapter 2
This chapter provides the background and a comprehensive review of the research necessary to make later chapters accessible to the reader. It provides a technical introduction to reinforcement learning and emphasises multiagent reinforcement learning. It also covers network attacks emphasising the existing use of machine learning. Lastly, it covers DDoS attacks and reviews current DDoS defence techniques.

Chapter 3
This chapter introduces the concept and the basic design of our proposed approach called Mul-

tiagent Router Throttling. Multiagent Router Throttling provides an agent-based distributed response to DDoS attacks, where multiple individual reinforcement learning agents learn to rate-limit traffic towards a victim server. One of the novel characteristics of our proposed approach is that it has a decentralised architecture and provides a decentralised coordinated response to the DDoS problem, thus being more resilient to the attacks. This initial design forms the basis on which the next couple of chapters build upon.

Chapter 4

This chapter tackles the scalability challenge. Scalability is one of the most important aspects of a defence system since a non-scalable mechanism will never be considered, let alone adopted, for wide deployment by a company or organisation. The chapter proposes Coordinated Team Learning (CTL) which is a novel design to the original Multiagent Router Throttling approach based on the divide-and-conquer paradigm. The CTL approach proposes several mechanisms, namely, hierarchical team-based communication, task decomposition and coordinated team rewards and can successfully scale up to large network scenarios.

Chapter 5

Online learning is the focus of this chapter; the previous two chapters deal with offline learning. Unlike non-learning approaches, one of the core advantages of reinforcement learning is its capability for online learning. To enable online learning we incorporate into CTL a form of reward shaping called difference rewards. The combination of CTL and difference rewards improves the scalability, learning speed and final performance. Specifically, the scalability of the proposed approach is demonstrated in experiments involving up to 1000 reinforcement learning agents. The significant improvements on scalability and learning speed lay the foundations for a potential real-world deployment. The proposed approach provides an adaptive response and meets the real-world requirements of robustness to agent failures and measurement noise.

Chapter 6

The thesis concludes in this chapter summarising all the contributions made, the deployment issues of our proposed approach and provides directions for future work on how to resolve these issues and extend our approach.

CHAPTER 2

Background and Literature Review

This chapter provides the background and current state of research related to this thesis. Section 2.1 covers the fundamentals of reinforcement learning and emphasises multiagent reinforcement learning and approaches to scalability. Section 2.2 covers the background on network attacks, emphasises defence approaches and discusses existing research on the use of machine learning. Section 2.3 provides a technical description of distributed denial of service (DDoS) attacks, discusses the characteristics of DDoS defence and reviews the current defence techniques.

2.1 Reinforcement Learning

A computer agent or agent is any entity that perceives its environment through sensors and acts upon it through actuators (Russell & Norvig 2003). Artificial Intelligence is defined as "the study of the design of intelligent agents" (Poole et al. 1998). An intelligent or rational agent is the agent that acts in such a way as to achieve the best possible outcome. Russell & Norvig (2003) formally defines a rational agent as follows:

> "For each possible percept sequence, a rational agent should select an action that is expected to maximise its performance measure, given the evidence provided by the percept sequence and whatever built-in prior knowledge the agents has."

Wooldridge (2009) suggests that an intelligent agent should be reactive, that is, to respond in a timely fashion to environmental changes; proactive, that is, to exhibit goal-directed behaviour by taking the initiative; and finally to have social abilities, that is, to interact (cooperate, coordinate, negotiate) with other agents or even humans.

Other definitions of intelligence or rationality exist. According to Russell & Norvig (2003) a rational agent should be capable of information gathering, capable of learning and should be autonomous. Information gathering refers to the execution of actions provided by exploration in order to modify future percepts. A rational agent should be able to learn from its perceptions, that is, to modify or augment its configuration as it gains experience. The agent should also be autonomous. An autonomous agent is the one which does not rely solely on its designer's prior knowledge. An autonomous agent should learn what it can to compensate in situations where partial or incorrect prior knowledge is provided.

Machine learning is the study of how to construct computer agents that can improve their behaviour with experience. Machine learning is a multidisciplinary concept highly influenced by ideas from philosophy, mathematics, psychology, biology, neuroscience and others. Mitchell (1997) formally defines Machine learning as follows:

> "A computer is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The questions that now arise are in which cases and why is machine learning preferred over pre-designed static behaviours? The answer is explained along four directions, namely, complexity, automation, adaptation and cost. As computing evolves, the overlapping connections, dependencies and interacting applications increase (Horn 2001). Therefore, the complexity of the environment may not allow good behaviours to be pre-designed. Also, these complex applications require for administrative decision-making and responses faster than any human can deliver (Horn 2001). Moreover, in dynamic environments where the environment changes frequently over time, a static behaviour may become inappropriate. Finally, machine learning can be used in cases where manual processing of enormous amount of data is difficult and expensive.

Machine learning distinguishes three different learning processes; these are, supervised learning, unsupervised learning and reinforcement learning (RL) (Mitchell 1997). Supervised learning involves learning a function from labelled examples of inputs and outputs provided by a domain expert (the teacher or supervisor). The learnt function can be then applied to new situations with unseen input data in order to predict the output. Unsupervised learning involves learning patterns in the input data and clusters relevant data when no output values are provided.

In RL the agent learns by interacting with its environment, specifically, it learns from reinforcement i.e. environmental feedback which indicates whether an action was right or wrong. This feedback is less informative than in supervised learning, but more informative than in unsupervised learning (Buşoniu et al. 2008).

According to Stone (2007) the premise of RL matches the agent paradigm exactly (although this is not say that classical learning is irrelevant to agents). As we will shortly explain, despite the elegant theory behind temporal difference algorithms, there have only been a limited number of

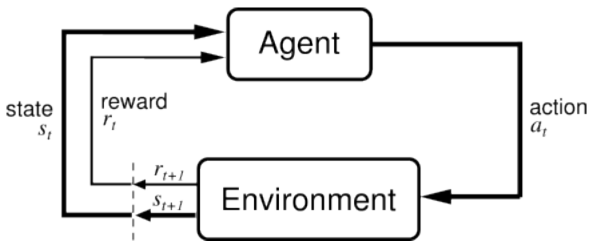


Figure 2.1: The Agent-Environment Interaction in RL (Sutton & Barto 1998)

successful large-scale applications such as playing backgammon (Tesauro 1994), robotic soccer (Stone et al. 2005) and air traffic management (Tumer & Agogino 2007).

Reinforcement learning is defined as the goal-directed learning from interaction between an active decision-making agent and its environment (Sutton & Barto 1998). The agent maps environmental input to local state information in discrete time steps. It then selects and executes an action. The environment presents the new state and also responds with a feedback, in the form of a positive or negative reward, evaluating the quality of the action performed. The action selection mechanism is based on the maximisation of a specific quantity such as the cumulative reward or average reward per time step. The agent finally updates its policy in such a way to optimise future rewards. This agent-environment interaction is shown in Figure 2.1.

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) provides a mathematical framework which is widely used for modelling the dynamics of an environment under different actions, and is useful for solving reinforcement learning problems. A state is said to be Markov or have the Markov property if its state signal includes a compact summary of all previous sensations, but in such a way that all essential information is kept (Sutton & Barto 1998). This enables the prediction of the next state and expected next reward according to the current state and action; that is, the Equation 2.1 holds.

$$\begin{aligned} &P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0) \\ &\qquad = P(s_{t+1} = s', r_{t+1} = r | s_t, a_t) \end{aligned} \tag{2.1}$$

Therefore, a Markov state makes available to the agent all the necessary information in order to make an optimal decision. It should be noted that a state may not fully satisfy the Markov property, but it is often useful to assume that it does. A reinforcement learning problem is called an MDP if it satisfies the Markov property (Sutton & Barto 1998). An MDP is defined by a 4-tuple $< S, A, R, P >$ where:

- S represents the set of all possible states

- A represents the set of all possible actions
- $R : S \times A \times S \to \mathbb{R}$ is the reward function which returns the expected reward received in state $s_{\mathrm{t}+1}$ when action a is executed in state s as shown in Equation 2.2.

$$R^a_{ss'} = E\left\{r_{\mathrm{t}+1} | s_t = s, a_t = a, s_{t+1} = s'\right\} \tag{2.2}$$

- $P : S \times A \times S \to [0, 1]$ is the transition probability function which returns the probability of reaching state s_{t+1} when action a is executed in state s as shown in Equation 2.3.

$$P^a_{ss'} = Pr\left\{s_{\mathrm{t}+1} = s' | s_t = s, a_t = a\right\} \tag{2.3}$$

$R^a_{ss'}$ and $P^a_{ss'}$ are called the environment's dynamics. A finite MDP is the one with a finite set of states and actions. MDPs assume that each action requires the same length of time. A Semi-Markov Decision Process (SMDP) relaxes this assumption by allowing actions to have different durations. MDPs also assume that the agent can observe the whole environment accurately at all times. This assumption is relaxed by considering the Partially Observable Markov Decision Process (POMDPs). SMDPs and POMDPs are outside the scope of our work and the interested reader is directed towards (Puterman 1994) and (Kaelbling et al. 1998) respectively.

2.1.2 Value Functions

A policy π defines a state-action mapping to the probability of selecting action a in state s, that is, $\pi : S \times A \to [0, 1]$. The notion of "how good" a particular state is, or "how good" would it be to perform an action in a particular state, is given by the value functions.

The state-value function $V^\pi(s)$ is the value of a state under a given policy (Sutton & Barto 1998). It is defined as the expected return when starting from state s and then following policy π as shown in Equation 2.4.

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t | s_t = s\right\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned} \tag{2.4}$$

R_t is also called the cumulative future discounted reward because future rewards are discounted by a factor of $0 \leq \gamma \leq 1$. The discount factor γ is introduced to deal with the problematic formulation for continuing tasks, because for the final time step ($k = \infty$) the return could be infinite. If $\gamma < 1$ the infinite sum has a finite value (as long as the reward is bounded). If $\gamma = 0$ the agent is called "myopic" and is concerned only with maximising immediate rewards. A myopic agent could maximise the return by separately maximising each immediate reward, but typically, doing that reduces access to future rewards and therefore the return may actually be reduced. As

γ approaches 1, the agent becomes more “farsighted”.

The action-value function $Q^{\pi}(s,a)$ is the value of a state-action pair under a given policy (Sutton & Barto 1998). It is defined as the expected return when starting from state s, performing action a and then following policy π as shown in Equation 2.5.

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \end{aligned} \tag{2.5}$$

The value function satisfies a particular recursive relationship shown in Equation 2.6 which is known as the Bellman equation for $V^{\pi}(s)$. It expresses a relationship between the state value and the values of its successor states i.e. the value of the start state must be equal to the discounted value of the expected next state, plus the expected future reward.

$$V^{\pi}(s) = \sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{2.6}$$

The goal of a reinforcement learning agent is to find the optimal policy π^*, that is, the policy which maximises the expected discounted reward over time. There always exists one policy which is better or equal to the rest. The state-value function and action-value function under optimal policy π^*, are called the optimal state-value function $V^*(s)$ and optimal action-value function $Q^*(s,a)$ respectively. They are defined in Equations 2.7 and 2.8.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.7}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.8}$$

The Bellman optimality equations for $V^*(s)$ and $Q^*(s,a)$ are shown in Equations 2.9 and 2.10 respectively. The Bellman optimality equation is essentially a system of equations, one for each state. When the environment's dynamics ($R^{a}_{ss'}$ and $P^{a}_{ss'}$) are known, dynamic programming (Sutton & Barto 1998) algorithms can be used to find the optimal value function and hence the optimal policy. However, in most real-world applications the environment's dynamics are not known, therefore temporal-difference learning (Sutton & Barto 1998) algorithms are used.

$$V^*(s) = \max_{a}\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^*(s')] \tag{2.9}$$

$$Q^*(s,a) = \sum_{s'}P^{a}_{ss'}[R^{a}_{ss'} + \gamma \max_{a'} Q^*(s',a')] \tag{2.10}$$

2.1.3 Temporal-Difference Learning

The assumption of perfect problem domain knowledge makes dynamic programming algorithms to be of limited practicality. Furthermore, these algorithms are computational intensive. Temporal-difference (TD) learning (Sutton & Barto 1998) is the most fundamental class of methods for solving the reinforcement learning problem. TD methods can learn directly from raw experience when the environment's dynamics $R^a_{ss'}$ and $P^a_{ss'}$ are not known.

TD methods perform bootstrapping, that is, they update estimates based on other learnt estimates. Model-based reinforcement learning refers to the algorithms that attempt to also learn the environmental model apart from the optimal policy. This involves the integration of planning and learning, and it is outside the scope of the thesis; the interested reader is directed towards (Sutton & Barto 1998). The focus of our work is on model-free or direct reinforcement learning. The two most widely used TD techniques are the SARSA and Q-learning algorithms which are described in this section.

In SARSA, the agent simply maintains a table to represent states and actions. This table is called the Q-table, and each entry is called a Q-value. Each Q-value represents the value of a state-action pair, that is $Q(s,a) \in \mathbb{R}$. SARSA uses the rule 2.11 to update the Q-values (Sutton & Barto 1998). Parameter $\alpha \in [0,1]$ represents the step-size or the learning rate of the algorithm. The learning rate α affects how big the change in estimated Q-value is and the discount factor γ affects the agent's preference over immediate rewards and future rewards. SARSA takes its name from the quintuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \tag{2.11}$$

Q-learning is very similar to SARSA. Q-learning uses the rule 2.12 to update the Q-values (Sutton & Barto 1998). It is an off-policy TD algorithm since value function updates are made following an independent policy.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \tag{2.12}$$

SARSA and Q-learning have both strict convergence criteria. The conditions under which Q converges to Q^* with a probability of one are presented below (Sutton & Barto 1998; Mitchell 1997). Even if not all the convergence criteria are met in practice, the algorithms can produce reasonably good results (Bertsekas & Tsitsiklis 1996).

1. The agents visit all state-action pairs infinitely often.
2. The environment's states satisfy the Markov property.
3. The learning rate reduces to zero.
4. Exploration reduces to zero.

5. The reward function produces values which have an upper bound.

2.1.4 Multiagent Reinforcement Learning

A multiagent system deals with the construction of a system involving multiple autonomous and interacting agents, which are situated in a common environment that can perceive through sensors, and act upon it through actuators (Buşoniu et al. 2008). There are many possible benefits for using multiagent systems including among others, better robustness to individual agent failures, better scalability, taking advantage of geographic distribution and sharing experiences for faster and better learning (Stone & Veloso 2000). Agents can be homogeneous or heterogeneous. The latter occurs in cases where each agent has a different set of capabilities or in cases where agents are controlled by a different designer or owner. Agents can also have the same or conflicting interests, thus leading to cooperative or competitive systems (or both).

Multiagent systems often need to be very complex, and multiagent reinforcement learning (MARL) is a promising candidate for dealing with this emerging complexity (Stone & Veloso 2000). The two MARL goals that are typically found in the literature are stability of an agent's learning, and adaptation to the dynamic behaviour of the other agents (Buşoniu et al. 2008). Convergence is often presented in the literature as a requirement for stability, and rationality as a requirement for adaptation. Learning in multiagent systems remains an open research area (Stone 2007). Despite the benefits of multiagent systems, new challenges arise that need to be tackled. The rest of this section discusses some of the challenges encountered in both single-agent and multiagent RL. We also discuss some of the well-known MARL algorithms.

2.1.4.1 Challenges

The curse of dimensionality (Sutton & Barto 1998) refers to the exponential growth of the search space, in terms of the number of states and actions. Table-based methods such as SARSA and Q-learning are most affected by this problem. In a MARL scenario the problem becomes more acute since complexity is now also exponential in the number of agents in the system (Buşoniu et al. 2008).

The problem of partial observability refers to situations where an agent does not perceive all of the information from a state as a result of data unavailability, sensor misinterpretation or hidden state information (Russell & Norvig 2003; Mitchell 1997). The problem becomes even harder in a MARL scenario where each agent cannot observe the entire environment, but has only knowledge about its local environment. Furthermore, in a MARL scenario the effects of an agent's action on the environment, also depend on the other agents' actions. Therefore, the Markov property does not hold if an agent cannot observe the joint action.

Typically in a multiagent system an agent is provided with a reward at the global level. However, recall that the other agents also act in the environment, therefore an agent may be rewarded for taking a bad action, or punished for taking a good action.

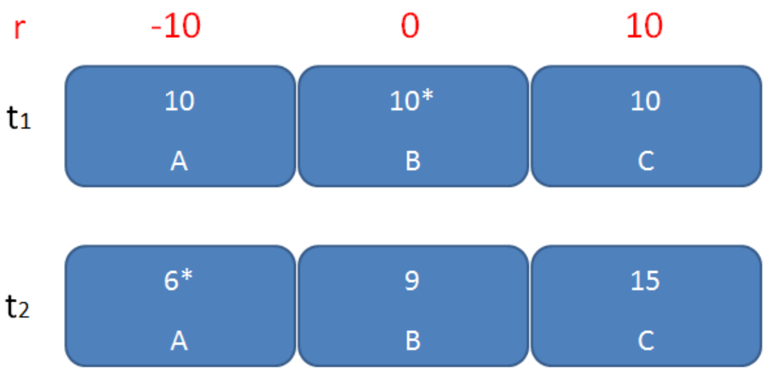


Figure 2.2: Instance of the Multi-armed Bandit Problem

Consider an instance of the multi-armed bandit problem shown in Figure 2.2 with three slot machines A, B and C which provide a reward of -10, 0 and 10 respectively. The goal is to maximise the sum of rewards provided. There are 30 learning agents in the system; the star indicates a particular player P. At time step t_1 (upper part of Figure 2.2) each machine is chosen by 10 agents and player P chooses machine B. The global reward given to each agent is $r_1 = 0$. At time step t_2 (bottom part of Figure 2.2), machines A, B and C are chosen by 6, 9 and 15 agents respectively. The global reward is now $r_2 = 90$. Therefore, although player P has chosen a bad action, he will still receive a reward of 90.

This is known as the multiagent credit assignment problem (Stone & Veloso 2000) and research attempts to provide answers to questions like "which agent should receive the reward?" and "how much reward should each agent receive?".

2.1.4.2 Algorithms

Applications of reinforcement learning to multiagent systems typically take one of two approaches; multiple individual learners or joint action learners. We describe these approaches below and also present some popular alternatives.

Individual Learners (ILs)

ILs (Claus & Boutilier 1998) involves the deployment of multiple agents each using a single-agent reinforcement learning algorithm. Multiple ILs assume any other agents to be a part of the environment and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action a in state s changes over time.

If an IL agent i uses the SARSA or Q-learning algorithm, then it applies the following update rules respectively:

$$Q_i(s,a) \leftarrow Q_i(s,a) + \alpha \left[r + \gamma Q_i(s',a') - Q_i(s,a)\right] \tag{2.13}$$

$$Q_i(s,a) \leftarrow Q_i(s,a) + \alpha \left[r + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)\right] \tag{2.14}$$

where s and s' are the states at times t and $t+1$ respectively, and r is the reward received at time $t+1$. Despite subsequent algorithm development ILs remains a popular solution.

Joint Action Learners (JALs)

JALs (Claus & Boutilier 1998) is a group of multiagent specific algorithms designed to consider the existence of other agents; in this setting an agent observes the actions of the other agents or each agent communicates its action to the others. To overcome the appearance of a dynamic environment, JALs were developed that extend their value function to consider for each state the value of each possible combination of actions by all agents.

For instance if a JAL agent i uses the SARSA algorithm, then it applies the following update rule:

$$Q_i(s,\overrightarrow{a}) \leftarrow Q_i(s,\overrightarrow{a}) + \alpha\left[r + \gamma Q_i(s',\overrightarrow{a}') - Q_i(s,\overrightarrow{a})\right] \tag{2.15}$$

where $\overrightarrow{a}$ and $\overrightarrow{a}'$ are the joint actions at times t and $t+1$ respectively.

For Q-learning it becomes slightly more complicated. A JAL agent i counts the number of times a state-joint action pair of the other agents occur $C(s,\overrightarrow{a_{-i}})$. The agent is then able to calculate its frequency using:

$$F(s,\overrightarrow{a_{-i}}) = \frac{C(s,\overrightarrow{a_{-i}})}{n(s)} \tag{2.16}$$

A JAL agent i that uses the Q-learning algorithm, it then applies the following update rule:

$$Q_i(s,\overrightarrow{a}) \leftarrow Q_i(s,\overrightarrow{a}) + \alpha\left[r + \gamma \max_{a_i'} \sum_{\overrightarrow{a'_{-i}}} F(s',\overrightarrow{a'_{-i}}) Q_i(s', a_i', \overrightarrow{a'_{-i}}) - Q_i(s,\overrightarrow{a})\right] \tag{2.17}$$

The consideration of the joint action causes an exponential increase in the number of values that must be calculated with each additional agent added to the system. For this reason, the method suffers from scalability issues in large and complex domains.

Alternatives

Lauer & Riedmiller (2000) proposed an extension of ILs for deterministic, fully cooperative tasks called distributed Q-learning. The idea behind the approach is that Q-values are only updated if the update leads to an increase in the Q-value:

$$Q_i(s,a) = max\left\{Q(s,a), r + \gamma \max_{a'} Q_i(s',a')\right\} \tag{2.18}$$

If an update leads to a decrease, the agent assumes that this is another agent's fault. Therefore, distributed Q-learning agents never reduce a Q-value. In a deterministic and fully cooperative environment, if all agents use distributed Q-learning they will converge to the optimal policy. For fully cooperative but stochastic environments, Optimal Adaptive Learning has been proposed which is guaranteed to converge to the optimal Nash equilibrium (Wang & Sandholm 2002).

Hu & Wellman (2003) proposed an extension of JALs called Nash Q-learning. This approach assumes that agents will play a Nash equilibrium strategy. It further assumes that an agent not only can observe the other agents' actions, but also their rewards. Each agent keeps a Q-table not only for itself, but for the other agents as well, and uses the following update rule:

$$Q_i(s, \vec{a}) \leftarrow Q_i(s, \vec{a}) + \alpha \left[r + \gamma Nash_i(s', Q_1, ..., Q_n) - Q_i(s, \vec{a}) \right] \tag{2.19}$$

where $Nash_i(s', Q_1, ..., Q_n)$ is the expected payoff for agent i when all agents play a Nash equilibrium in state s'. The approach has been shown to converge to a Nash equilibrium under strict conditions. A recent algorithm called Nash-DE weakens these conditions and has been shown to converge to a Nash equilibrium as well (Akchurina 2009).

Future Coordinating Q-learning (FCQ-learning) combines both ILs and JALs (De Hauwere et al. 2011). An agent starts as an individual learner and at each state it decides using statistical tests whether its state space would also include the joint action. As a consequence, only in some states the value function is extended to consider the joint action.

Another approach which provides a combination of ILs and JALs is Adaptive State Focus Q-learning (Busoniu et al. 2005). An agent starts as an individual learner, but if convergence is not reached, its state space is expanded to include the state of the other agents.

2.1.5 Scaling-Up

Reinforcement learning algorithms experience difficulties when scaling-up to real-world applications (Buşoniu et al. 2008; Sutton & Barto 1998). Real-world applications involve complex domains with large and continuous state and action spaces. Therefore, learning becomes infeasible due to memory constraints and computational time. This section describes a non-exhaustive list of popular approaches to improve scalability and coordination; these are, function approximation, communication, exploration-exploitation strategies, value function initialisation and reward shaping.

2.1.5.1 Function Approximation

It is perhaps the most common technique to deal with scalability issues. The idea of function approximation techniques is to reduce the impact of scalability by reducing the agent's exploration (Sutton & Barto 1998). With large and continuous state and action spaces, a tabular implementation becomes very inefficient. Function approximation techniques approximate the search space.

Tile coding is one of the most common function approximation techniques (Sutton & Barto 1998). Tile coding partitions the state space into tilings, each tiling is further partitioned into tiles where state feature values are grouped into. For example, Figure 2.3 shows a two-dimensional space (i.e. with two state features) partitioned into two tilings, each consisting of 25 tiles. Each point in the two-dimensional state space activates one tile per tiling; this is shown by the point

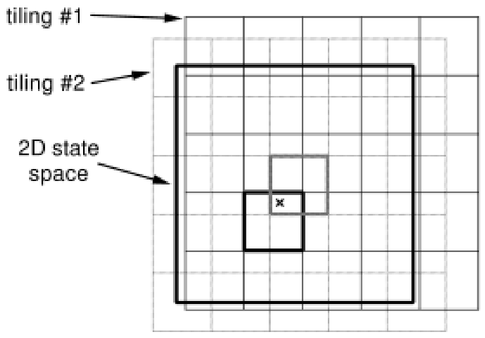


Figure 2.3: Example of Tile Coding with Overlapping Tiles (Sutton & Barto 1998)

“x” in the figure. In other words, activated tiles correspond to multiple states. Expected reward values are now stored per tile rather than state-action pair. Tile coding sensitivity can be adjusted by the addition or removal of tilings and tiles. The shape of the tiles indicates the degree of generalisation, and the number of tilings indicates the degree of resolution.

State feature selection is a manual process performed by the designer. Obviously, domain knowledge would benefit the process as the domain expert will include important features and remove irrelevant ones. With a reduced number of features the state-action space becomes smaller and agents learn faster.

Boyan & Moore (1995) state that the use of function approximation in a continuous search space does not guarantee convergence to the optimal policy as the requirement of visiting all state-action pairs infinitely often cannot be realised. However, tile coding has been successfully applied in many domains, including the large-scale and complex domain of robotic soccer (Stone et al. 2005). According to Sutton & Barto (1998), it has been empirically demonstrated that tile coding works better with the SARSA algorithm, rather than Q-learning. For other methods of function approximation the reader is directed towards (Sutton & Barto 1998).

2.1.5.2 Communication

Communication is considered to be the most common way of interaction between intelligent agents (Matarić 1998). Any observable behaviour can also be interpreted as a form of communication. For this reason we distinguish between the following two forms of communication (Matarić 1998). Direct communication is defined as a purely communicative act in order to transmit information (e.g. a speech act, radio message transmission). Indirect communication is concerned with the observation of other agents’ behaviour and its effects on the environment. Cooperation is a form of interaction, usually based on communication. Cooperative behaviours may depend on direct or indirect communication. Communication is practised by many species in nature, and has also been demonstrated to be beneficial in multiagent systems.

Typically, an agent communicates or shares its local information with other agents in the system. Such information can include state sensations, actions, rewards or a combination of the aforementioned. Communication tackles the partial observability problem, where distributed agents cannot sense all of the relevant information necessary to complete a cooperative task. This however does not come without challenges; communication messages can be costly, noisy, corrupted, dropped or received out of order. For these reasons and also because real-world applications are typically already complex, communication should be minimal and simple.

Tan (1993) uses agent communication in the prey-hunter domain. The domain's task is for one or more hunter RL agents to capture at least one randomly moving prey agent in a 10 by 10 grid world. A prey is captured if it occupies the same cell with a hunter. The author runs several experiments, some of which are discussed here. In the first experiment the domain consists of one hunter agent and one prey agent. The author introduces a randomly moving scout agent which at each time step shares its action and local state sensation with the hunter agent (it is assumed that the hunter agent is aware of the scout's initial relative location). Results show that the hunter with a scout requires fewer steps to capture the prey than the one without. Results improve as the scout's visual depth increase (with the expense of more communication).

In a variation of this experiment, there are two hunter RL agents and one prey agent. Tan (1993) extends the scout concept by having each hunter to act as a scout for the other. Similar results are obtained as the previous experiment. However, when the visual depth was small mutual-scouting hinders learning. As a conclusion, sensory information from other agents could be beneficial, but unnecessary or insufficient information can interfere with learning. Tan (1993) also considers the case of joint tasks, where a prey is now captured if it occupies the same cell with two hunter agents. Hunter agents cooperate by either passively observing each other or actively sharing (mutual-scouting) their location and state sensations. Results reveal that cooperative agents perform the joint task significantly better than the independent agents.

Servin & Kudenko (2008a,b) and Servin (2009) propose a distributed RL approach to semantic-less communication. Agents are organised in a hierarchy and a sender agent (lower hierarchical level) learns semantic-less communication signals which represent the "summary" of its local state observations. The recipient agent (higher hierarchical level) also needs to learn how to interpret these semantic-less signals. This approach improves the scalability of the multiagent system since the recipient's state space is significantly reduced. It also reduces communication costs.

2.1.5.3 Exploration-Exploitation Trade-off

The exploration-exploitation trade-off constitutes a critical issue in the design of an RL agent. Recall that one of the convergence criteria is the need for all state-action pairs to be visited infinitely often, in a goal-directed efficient manner. Random search and other systematic search methods such as breadth-first search may visit all state-action pairs, but not in an efficient manner.

The exploration-exploitation trade-off plays an important role in achieving the goal-directed efficient visit to the state-action pairs. An agent exploits its knowledge when it selects actions that will result to higher immediate rewards, and an agent is said to explore when it explores new states and actions. The exploration-exploitation trade-off aims to offer a balance between the exploitation of the agent's knowledge and the exploration through which the agent's knowledge is enriched. Too much exploitation is not desirable since it is more likely for an agent to stuck in a local maximum. Too much exploration is also not desirable since learning becomes very difficult. In scenarios where multiple agents co-exist, the agents may also need to explore in order to obtain information about the other agents.

We describe two of the most popular action selection mechanisms; these are the greedy and ϵ-greedy (Sutton & Barto 1998) mechanisms. An agent uses the greedy action selection mechanism when it selects actions only by exploiting its knowledge. As previously stated, behaving greedily all the time can lead to a local maximum settlement. A variant of the greedy mechanism is called ϵ-greedy, where the agent behaves greedily most of the time but with a probability ϵ it selects an action randomly. ϵ-greedy methods can theoretically sample over time every action ensuring convergence to Q^*. The value of ϵ plays an important role; a high value e.g. 0.9 allows more exploration while a low value e.g. 0.01 allows more exploitation. To get the best of both, it is advised to reduce ϵ over time (Sutton & Barto 1998). Therefore, exploration is encouraged by using a high value of ϵ, but as the agent becomes more and more experienced exploitation is encouraged by using a low value of ϵ.

We will later discuss about reward shaping, a more advanced way for effective exploration.

2.1.5.4 Value Function Initialisation

A value function is typically initialised pessimistically or optimistically. Pessimistic initialisation sets the Q-values i.e. state-action pairs to the minimum reward value, while optimistic initialisation sets the Q-values to the maximum reward value. For example, if a reward function generates rewards in the range $[0, 10]$, the former will initialise the values to 0 while the latter to 10. Optimistic initialisation allows all actions in all states to be tried out before convergence to the optimal policy occurs. On the contrary, with pessimistic initialisation an agent's behaviour can only converge to the optimal policy if the exploration-exploitation strategy identifies it. A randomly initialised value function is often used to combine the advantages of each method.

2.1.5.5 Reward Shaping

Reward shaping refers to the process of training an agent using an artificial reward rather than the reward naturally received by the environment, in order to favour or bias learning towards specific behaviours (Mataric 1994). Reward shaping constitutes a common way to incorporate domain or expert knowledge.

An early approach for incorporating knowledge and facilitating the learning task is to initialise the Q-values (Wiewiora 2003). If such information is known this could benefit the agent a lot. However, the major difficulty of this approach is to interpret a certain desirable behaviour (the knowledge) into a set of Q-values.

Randlov & Alstrom (1998) consider an agent learning to ride a bicycle from a starting point to a target destination. The authors use reward shaping by providing an additional reward to encourage the RL agent to stay balanced. Results show that shaping accelerates the learning process immensely. However, reward shaping must be used with great care otherwise it can produce undesirable results. In one of the experiments regarding the bicycle domain, the RL agent learnt to ride in circles since it could benefit more for staying balanced rather than riding to the target destination.

To overcome such problems potential-based reward shaping (PBRS) has been proposed (Ng et al. 1999). The additional reward provided is the difference of a potential function Φ defined over a source s and a destination state s' as shown in Equation 2.20 below:

$$PBRS = r + \gamma\Phi(s') - \Phi(s) \tag{2.20}$$

where r is the original environmental reward and γ is the same discount factor as used in Equations 2.11 and 2.12.

Ng et al. (1999) have proved that the use of PBRS does not alter the optimal policy of a single agent, although as a heuristic it can increase or decrease the time of convergence. Wiewiora (2003) has proved that an agent learning with PBRS and no knowledge-based Q-table initialisation, will behave identically to an agent not using PBRS but its Q-table is initialised with the same potential function. Devlin & Kudenko (2011, 2012) and Devlin et al. (2014) extend the concept of PBRS to the multiagent scenario. The authors have proved that PBRS in a multiagent system can alter the joint policy learnt but does not alter the Nash equilibria of the underlying system.

Difference rewards is a form of reward shaping that tackles the multiagent credit assignment problem, specifically designed for fully cooperative multiagent systems (Wolpert & Tumer 2000). The difference rewards mechanism rewards an agent based on its contribution to the system's task by removing a large amount of the noise created by the actions of the other agents in the system. Difference rewards have been successfully applied in many real-world applications such as air traffic management (Tumer & Agogino 2007) and distributed sensor networks (HolmesParker et al. 2013; Colby & Tumer 2013). We will extensively describe difference rewards in chapter 5.

2.2 Network Attacks and Defence

The increasing adoption of technologies, the wide spread application of computers and the exponential growth of networks has made the area of information technology an integral part of our lives. This gave birth to numerous applications such as social networking, online shopping, smart metering and cloud computing where computer and network security plays a vital role. Security is defined as the protection provided to an automated information system in order to preserve the confidentiality, integrity and availability of its resources, commonly known as the CIA triad

(Stallings 2010). Confidentiality relates to two concepts, namely data confidentiality and privacy. The former ensures that confidential data is not exposed to unauthorised individuals. The latter ensures that individuals have the full control of all the information relating to them. Integrity relates to two concepts, namely data integrity and system integrity. Data integrity protects data from undergoing an unauthorised modification or destruction. System integrity deals with the authorised performance of a system's intended functionality. Availability assures that a system's service is provided in a timely and reliable manner to its legitimate users.

2.2.1 Security Attacks

Eavesdropping (Stallings 2010) refers to obtaining information from data being transmitted, for example, over a telephone conversation or an e-mail exchange. This kind of attack does not cause any alteration of the data. It is therefore very difficult to detect, and for this reason the focus is on prevention, typically by means of encryption. However, even if the data is encrypted, the attacker can perform **traffic analysis** and may still be able to extract information from observed patterns, for example, frequency or length of messages exchanged.

A **masquerade** (Stallings 2010) attack occurs when the attacker pretends to be a different entity. **Reconnaissance** (Pfleeger & Pfleeger 2006) refers to the preparation done by the attacker prior to an attack. The attacker attempts to gather as much information as possible in order to identify potential victims that are easy targets, such as weak or vulnerable machines.

A **Trojan** (Hansman & Hunt 2005) is a computer program that serves a malicious purpose (e.g. capturing passwords) when run by a user. A **virus** is a self-replicated computer program that serves a malicious purpose which when run by a user it attaches itself to other programs. Contrary to viruses, a **worm** is a self-replicated piece of code which spreads rapidly through network services or email.

The most rapidly-growing problem however is the **rootkit** (Anderson 2008). Once installed, the rootkit allows the attacker to remotely control the infected machine. Typically, machines infected in this way become part of a botnet (network of bots). Interestingly, rootkits are also used by law enforcement agencies which turn suspects' laptops into listening devices. Modern rootkits have stealth capabilities, that is, they try to hide from the operating system so that they cannot be detected and removed. Some rootkits even install antivirus software to prevent competing botnets from taking over the machine.

Distributed denial of service (DDoS) (Anderson 2008) attacks target the availability of a system. The attacker compromises a large number of host machines (the botnet) which start bombarding the victim with traffic. Such attacks do not allow the authorised or legitimate users to access resources in a timely and efficient manner, or even make the resources completely inaccessible to them. DDoS attacks are extensively discussed in a later section. Similarly to a DDoS attack, **spam** (Anderson 2008) refers to the flood of generally unwanted emails sent out mostly by botnet machines.

Finally, **social engineering** (Anderson 2008) attacks refer to the attacks which exploit psychology rather than technology. In other words, these attacks target people directly rather than their computer or network. The fastest-growing attack of this type is **phishing**, where victims receive an email instructing them to visit a website which appears to be legitimate (such as a bank's website) but in fact it is not. The website is designed to steal sensitive information such as usernames, passwords and credit card details.

2.2.2 Defence Against Network Attacks

Anderson (2008) defines four lines of defence against network attacks. These are management, filtering, intrusion detection and encryption.

Management aims at minimising the occurrence of intrusions. Such mechanisms include applying patches to fix known vulnerabilities, changing default settings (e.g. passwords), removing unnecessary services (e.g. mail servers) and training staff to avoid taking risky actions that may expose a system.

The second line of defence is filtering the harmful network traffic. The firewall is considered to be the most widely sold product. A firewall is a machine that lies between a local system and Internet, examines packets and performs filtering according to specific rules. There are three types of filters, the ones that operate at the IP packet level (network layer), at the TCP session level (transport layer) or the application level (application layer). An example of the latter is the spam filter. An ingress filter is responsible for examining incoming traffic, while an egress filter is responsible for the outgoing traffic.

Intrusion detection is the third line of defence. Preventive mechanisms are important and essential but they are not perfect. In addition, "it's often cheaper to prevent some of the attacks and detect the rest than it is to try to prevent everything" (Anderson 2008). An intrusion detection system (IDS) monitors the logs and network traffic, and triggers an intrusion alarm if a malicious or suspicious behaviour is detected. IDSs are discussed in detail in the next section.

The last line of defence is the use of security protocols using encryption such as TLS and SSH. Encryption can be useful but similarly to the others, it does not entirely solve the problem. Encryption is outside the scope of our work.

2.2.3 Intrusion Detection

Although the focus of this thesis is on intrusion response, we provide the background material and review the research conducted for intrusion detection since the two areas are directly relevant. Furthermore, there is an extensive literature regarding the application of machine learning to intrusion detection, specifically anomaly detection, where an alarm is triggered when an anomalous event is detected.

As already mentioned preventive mechanisms are not perfect. Furthermore, it is very expensive to try and prevent everything (Anderson 2008). Therefore, it is always safe to assume that attacks will happen. The pioneering work of Anderson (1980) laid the foundations for automated intrusion detection. Anderson (1980) claimed that audit trails can reveal important information regarding the identification of misuse behaviour. Nowadays an intrusion detection system is defined as the one which monitors the logs and network traffic, and triggers an alarm if something malicious or suspicious is detected (Anderson 2008). The key aspects of IDSs are presented below.

2.2.3.1 Data Collection

Depending on the data collection points, intrusion detection can be classified as host-based and network-based (Ghorbani et al. 2010). A host-based IDS analyses audit logs and system calls on the protected host. An audit log file contains sets of events created by the operating system such as running processes and consumed memory. System calls are sequences of operations a program requests from the operating system such as open, read, and close. A host-based IDS can also analyse the log files for specific applications and even the program code itself.

Our focus is on network-based IDSs. Data can be collected from network packets and flows (Ghorbani et al. 2010). The most commonly used open-source software to collect and analyse packets is the Tcpdump (TCPDUMP 2011). A flow is defined as

> "A set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties."
> (Sperotto et al. 2010)

Traditionally, these properties are the IP source and destination addresses, the source and destination port numbers, and the transport layer's protocol type. NetFlow (Cisco 2011) is the de facto industry standard protocol for flow capture and monitoring.

Inspecting the payload of packets requires high resource consumption, and due to today's high speed networks it is infeasible to inspect each individual packet. Alternatively, inspecting flow information can be more efficient since they consist of aggregated information and do not carry any payload. However, flow-based inspection does not offer the precision of packet-based inspection, for example, pattern matching in payload content.

Each approach has its advantages and limitations and therefore they should complement each other. Indeed, that is exactly what is happening according to the Arbor Network's worldwide security survey conducted by Anstee et al. (2013) among more than 290 companies and organisations. Specifically, NetFlow and deep packet inspection (DPI) tools (among others) are being used by 83% and 41% of the respondents. 70% of the respondents use the firewall logs for threat detection.

Finally, feature selection constitutes a critical design decision, since it highly affects the detection performance. Typically, these features include packet sizes, IP addresses, ports, timestamps, session size, session duration, session volume etc (Sommer & Paxson 2010). For a comprehensive feature classification scheme the interested reader is directed towards (Onut & Ghorbani 2007), where the authors classify features according to 15 categories.

2.2.3.2 Detection Types

There are typically three intrusion detection types, namely, misuse, anomaly and specification-based detection (Ghorbani et al. 2010).

Misuse or signature-based detection aims at identifying already known attacks by monitoring the traffic for signatures, these are, known characteristics of attacks. The disadvantage of this approach is that it cannot uncover novel intrusions. Pattern matching approaches are commonly used for misuse detection. Snort (Roesch 1999) is a well-known defence system which uses misuse detection.

Anomaly detection aims at uncovering novel attacks by attempting to learn a notion of normal network activity; any activity which deviates from the normality profile is marked as potentially intrusive. The disadvantage of this approach is that it usually suffers from a high rate of false positives and negatives. Anomaly detection typically makes use of advanced statistical techniques or machine learning.

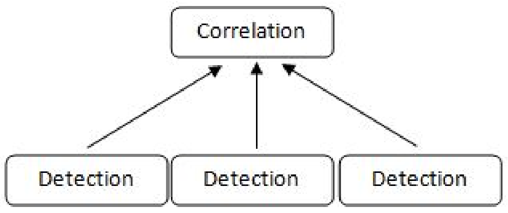
Contrary to learning a normal profile of activity, specification-based detection defines allowed types of behaviours and marks any other behaviour that deviates from the specification as intrusive. The disadvantage of this approach is that it is very difficult to specify and validate the behaviour for systems with a large number of components and applications.

2.2.3.3 Architecture

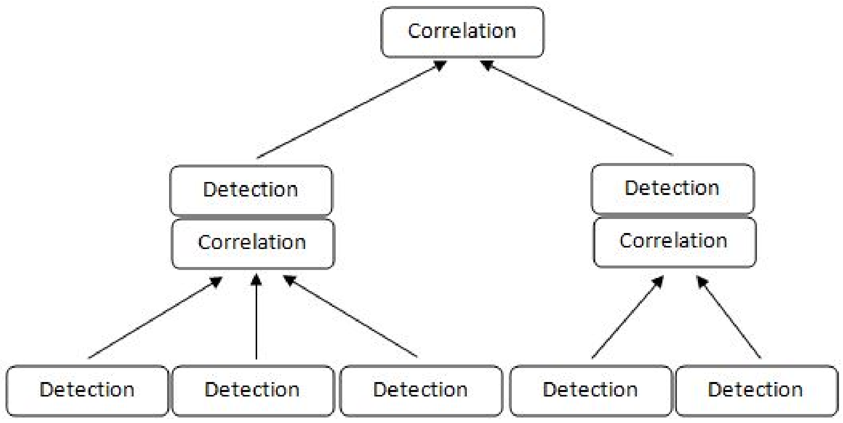
IDSs can be collaborative or non-collaborative. Collaborative IDSs were proposed to detect more accurately coordinated attacks such as DDoS attacks and worm outbreaks (Zhou et al. 2010). A collaborative IDS can detect intrusions that occur at different places in the Internet by correlating decisions between other IDSs. A collaborative IDS has typically two components, a detection unit and a correlation unit. A detection unit consists of multiple sensors, where each sensor monitors a network component or host, and raises low-level intrusion alerts. A correlation unit receives and analyses all low-level alerts and generates a high-level intrusion alarm.

Depending on the place where data is collected and analysed a collaborative IDS can be classified as centralised, hierarchical (or semi-distributed) or fully distributed (Zhou et al. 2010). An example of a centralised IDS architecture is shown in Figure 2.4a. It depicts three detection units i.e. IDSs where low-level alerts are reported to a central correlation unit for further analysis.

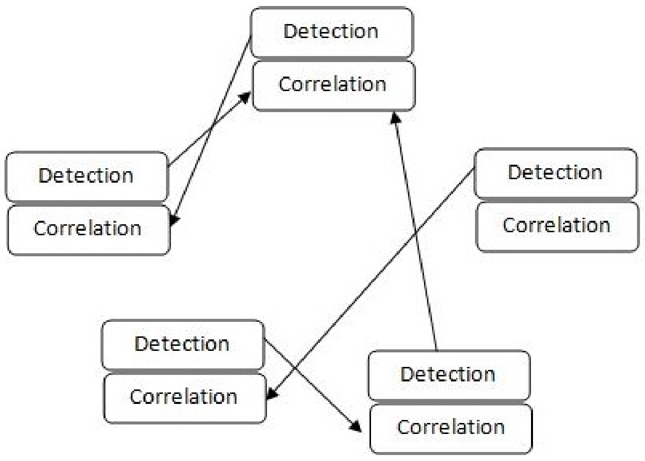
Figure 2.4b shows a three-level hierarchical IDS. First-level detection units report to a second-level correlation/detection unit which analyses information and reports to the third-level unit. The highest-level of the hierarchy further analyses the received information and decides whether or not an intrusion alarm should be triggered. In practice, a hierarchy can be determined by many factors such as the geography, network administration control and collection of similar software platforms.



(a) Centralised IDS



(b) Hierarchical IDS



(c) Distributed IDS

Figure 2.4: IDS Architecture

Finally, Figure 2.4c depicts a fully distributed IDS architecture. Each IDS monitors and analyses information and can trigger an intrusion alarm if necessary. An IDS can also exchange information with other IDSs. Different communication schemes are found in the literature such as peer-to-peer (P2P), gossiping and multicast.

The main advantage of the centralised architecture is that it offers a high detection accuracy. However, it is not efficient for large-scale deployment because the central component often has slow response times. Another drawback is that the central component becomes a single point of failure or vulnerability (i.e. it can become a victim itself).

Distributed architectures eliminate the single point of failure, although failure of higher-level nodes in a hierarchical architecture will still cause a problem. They also scale-up better, with the fully distributed architecture scaling-up better. However, distributed architectures suffer from reduced detection accuracy, especially for the fully distributed architecture where an IDS component has only limited access to global information.

Finally, we note that most IDSs are centralised and little work exists on large-scale distributed IDS architectures (Ghorbani et al. 2010; Zhou et al. 2010).

2.2.4 Anomaly Detection Using Machine Learning

Anomaly detection typically makes use of statistical and machine learning techniques. One of the earliest attempts to use machine learning, and more specifically artificial neural networks for anomaly detection was made by Debar et al. (1992). The authors use a fully connected recurrent network with many layers and the backpropagation learning algorithm. To take advantage of the knowledge in the field, an expert system is also integrated that contains a database of intrusion scenarios. The IDS quickly triggers alarms when it identifies deviations from normal profile based on the neural network model (anomaly detection) or when there is a match with a database entry (misuse detection). The system can have a high development cost. It is difficult to train a large neural network because finding the right parameter values may require a great number of trainings and simulations. Furthermore, it has a prediction error and for this reason the authors suggest that their approach should complement a statistics-based anomaly detection component.

Artificial neural networks have quickly gained wide popularity and the work of Ghosh et al. (1998) is regarded as a typical one in the field. The authors use a multilayer perceptron with the backpropagation learning algorithm to detect anomalous and unknown intrusions. Interestingly, the authors show that training the neural network with randomly generated data can lead to better performance. A drawback of this approach is that any change in the training set would require the re-training of the neural network.

The earliest work to apply RL appears in 2000 by (Cannady 2000). The authors use the Cerebellar Model Articulation Controller (CMAC), which a special type of a three-layer feedforward neural network capable of online learning. The system uses feedback from the protected system to update the CMAC weights. The system learns to detect UDP and ICMP flooding DoS attacks.

Servin & Kudenko (2008b,a) and Servin (2009) propose a distributed RL approach in order to detect flooding DDoS attacks. Agents are organised in a hierarchy and a sender agent (lower hierarchical level) learns semantic-less communication signals which represent the "summary" of its local state observations. The recipient agent (higher hierarchical level) also needs to learn how to interpret these semantic-less signals. Finally, the root of the hierarchy learns whether or not an intrusion alarm should be triggered. The proposed approach is capable of uncovering novel intrusions and performs better than a hand-coded misuse detector. However, it performs worse than a hand-coded hybrid (misuse and anomaly) detector but the authors state that the main advantage of their approach is that no network expert is required to configure the anomaly detector.

Xu et al. (2007) state that information sharing such as combining local information or decisions among agents can improve detection accuracy but it can be costly. The authors use distributed reinforcement learning in order to optimise communication costs of information exchange among agents, by learning when to broadcast information to other agents.

Researchers have also turned into other machine learning techniques such as clustering and nearest neighbour-based techniques. Both techniques are unsupervised i.e. they do not require labelled training examples. Mitrokotsa & Douligeris (2005) use a special type of neural network called self-organising map. It produces clusters of data in a 2D topological map where similar features are grouped together in specific regions of the map.

Ertoz et al. (2003) introduce the MInnesota Intrusion Detection System (MINDS). MINDS uses the Local Outlier Factor (LOF) approach which assigns to each data example a degree of being an outlier with respect to its neighbourhood. The basic assumption behind nearest neighbour-based techniques is that normal instances lie in dense neighbourhoods while anomalous instances lie far from their closest neighbours. Evaluation of MINDS is performed on the DARPA 1998 dataset and on a real network at the University of Minnesota with the help of a security analyst. MINDS detected several novel intrusions, some of which could not be identified by Snort (Roesch 1999), a state-of-the-art signature-based tool. The authors state that MINDS is not intended to replace Snort but to complement it, and the two should work synergistically.

Researchers have also developed an epidemiological model where a self-sustaining virus outbreak needs to pass an *epidemic threshold* at which its rate of replication exceeds the rate of its removal (Kephart & White 1993). The first proposal to apply artificial immune systems came from Forrest et al. (1996). Their pioneering work gave the spark of inspiration and motivated other researchers in the field to use immune inspired anomaly detection systems. Hart & Timmis (2005) state that results so far haven't been very successful, but significant breakthroughs may occur in the future.

2.2.5 Challenges and Guidelines on Using Machine Learning

Sommer & Paxson (2010) have recently criticised the use of machine learning for anomaly detection. They state that although machine learning has been successfully applied in domains like spam detection and product recommendation systems, the vast majority of IDSs found in operational deployment are misuse detectors. Furthermore, the anomaly detectors found in operational deployment employ mostly statistical techniques. The authors identify the following challenges on using machine learning for anomaly detection. We also present a set of guidelines for strengthening future research. We emphasise that the authors do not consider machine learning to be an inappropriate tool for anomaly detection. On the contrary, the authors believe the use of machine learning is reasonable and possible but great care is required.

2.2.5.1 Challenges

High Cost of Errors

Contrary to other domains, in anomaly detection the cost of errors is really high. On the one hand a false positive is very expensive since it requires analyst time thoroughly examining the reported incident. On the other hand a false negative can cause serious damage to an organisation's IT infrastructure. A related challenge is that there "too few attacks". Anderson (2008) gives the following example: "If there are ten real attacks per million sessions - which is certainly an overestimation - then even if the system has a false alarm rate as low as 0.1%, the ratio of false to real alarms will be 100". When one considers product recommendation systems, although a relevant recommendation can potentially increase sales, an irrelevant recommendation can only lead to a customer just continue shopping. According to Sommer & Paxson (2010), the high cost of errors is the primary reason for the lack of machine learning-based anomaly detectors in operational deployment. The high rate of false alarms occurs for the following two reasons.

Firstly, Sommer & Paxson (2010) claim that machine learning works better at identifying similarities rather than anomalies. A fundamental rule of a machine learning algorithm is that its training requires a large and representative set of instances of both positive (normal specimens) and negative (anomalous specimens) classes. Consider for example spam detection, a machine learning algorithm is provided with large amounts of spam and ham, and after the training period it is able to reliable identify unsolicited email.

However, there exist some problems with the anomaly detection domain. Anomaly detection is used to reveal novel attacks whose symptoms deviate from normal behaviour. Therefore one cannot train a machine learning algorithm with anomalous specimens. Furthermore, there is not a perfect model of normality. It is very difficult to define a "normal" region which contains every possible normal activity. Moreover, normal activity evolves and may not be representative in the near future (Chandola et al. 2009). Sommer & Paxson (2010) state that if the machine learning algorithm is trained using specimens of known attacks and specimens of normal activity, then machine learning is better suited for detecting mutated (variations of existing) intrusions rather than novel ones.

Secondly, network traffic usually exhibits great diversity (Sommer & Paxson 2010). Traffic characteristics such as bandwidth, duration of connections and application mix can exhibit great variability resulting to an unpredictable behaviour. The diversity and variability occur regularly and often falsely considered as anomalous by an IDS. For this reason it is difficult to define what is actually “normal”. Furthermore, the Internet is an environment full of noisy data such as packets created from software bugs and out-of-date or corrupt DNS data (Anderson 2008). Similarly, such data can be falsely considered anomalous by an IDS.

Difficulties with Evaluation

Two difficulties are mainly encountered regarding the evaluation of anomaly detectors (Sommer & Paxson 2010). The first problem is the difficulty of obtaining training data. The reason behind this is privacy concerns due to the sensitivity of such data (e.g. confidential communications and business secrets). Also, there are not any standardised datasets available. According to Sommer & Paxson (2010) the two publicly available datasets, namely DARPA/Lincoln Labs and KDD Cup datasets, should not be used for any current study since they are now more than a decade old. Again, when compared to other domains e.g. spam detection, large datasets of spam do exist that are free of privacy concerns.

The second problem is about *adversarial drift*, that is, when attackers modify their behaviour to evade detection. This constitutes the old classic arms-race between attackers and defenders. The authors do admit however that exploiting a machine learning technique requires considerable effort, time and expertise from the attacker’s behalf. In addition, since most attackers target weak and vulnerable systems instead of handpicking victims, the possibility of a sophisticated attack exploiting a machine learning technique is low (Sommer & Paxson 2010). Having said that, sophisticated attacks do occur as people are sometimes willing to spend very large amounts of money to achieve their goals (as in the Stuxnet worm (Zetter 2014) case). Therefore, if adversarial drift is not taken into consideration or is de-prioritised, the risk associated with a sophisticated attack exploiting a machine learning technique is very high.

Semantic Gap

Anomaly detectors typically generate an output label, that is, an instance is labelled as either normal or anomalous. Some detectors improve on that by generating an anomaly score, indicating the degree to which an instance is considered anomalous (Chandola et al. 2009). Sommer & Paxson (2010) state that this is not enough. Consider for example the output “HTTP traffic of host did not match the normal profile”. Even if we assume that the system correctly detected a web server exploit, a network operator must still spend considerable effort in order to figure out what had happened. In other words, anomaly detectors ideally must transfer their results into actionable reports. Sommer & Paxson (2010) term this the *semantic gap*.

Also, someone is at serious risk of discriminating if he uses a machine learning-based IDS. For example, how would someone defend himself in a court if he cannot explain the underlying rules of a neural network? Anderson (2008) calls this *redlining*.

2.2.5.2 Guidelines

Cost Reduction

A way to reduce the false alarm rate and therefore the cost is to keep the scope narrow, that is, to limit the types of intrusions an anomaly detector is trained to identify (Sommer & Paxson 2010). Kantchelian et al. (2013) propose the use of an ensemble of classifiers one for each family of malicious behaviour. Each individual classifier generates a decision and the network operator combines each of them to generate a final decision, which is likely to be more accurate since each classifier specialises into detecting a specific family. Each classifier can use the same machine learning technique or different classifiers for each family can be used. Google uses an ensemble of classifiers for detecting malicious advertisements (Sculley et al. 2011). The drawback of this approach is that although there has been some work on automated classification of instances to families, deployed systems do so manually.

Another way to reduce the cost is to use aggregated features such as "number of connections per source" or to average features over a period of time such as "volume per hour" in order to deal with the diversity, variability and noise of network traffic. One type of machine learning-based anomaly detection system that is indeed found in operational deployment is the one that takes into consideration highly aggregated information (Sommer & Paxson 2010). Finally, a well-designed machine learning algorithm could potentially reduce the false alarm rate. For example, a careful inspection of the feature space will most likely reveal that some features are irrelevant, and some are more relevant than others.

Evaluation

As already discussed the two publicly available datasets should not be used by current studies since they are now more than a decade old. Researchers have turned into alternative approaches (Sommer & Paxson 2010). One alternative is to use simulation. The advantage is that it is free of privacy concerns. However, it is very difficult to realistically simulate Internet traffic. Another alternative is to anonymise or remove sensitive information from captured data. However, anomaly detectors often rely on information that had previously been removed during the anonymisation process. Besides, there is always the fear that sensitive information can still be leaked.

Another option is to capture data from a small-scale lab. However, the traffic obtained is different from the aggregated traffic seen upstream (where intrusion detectors are typically deployed). Sommer & Paxson (2010) state that the "gold standard" is to obtain data from large-scale environments. Even this choice suffers from some limitations. Specifically, data obtained from large-scale environments lack information that had been filtered out or unintentionally lost. Furthermore, such a dataset will contain a lot of noisy information. The important point here is to realise that no evaluation method is perfect, and researchers always need to acknowledge the shortcomings of the approach they use.

Regarding the adversarial drift, the use of an ensemble of classifiers as discussed earlier can help to tackle this issue (Kantchelian et al. 2013). An adversarial drift could theoretically be arbitrarily radical, however in practice it is limited by the adversary's resources. Typically during a campaign, an adversary recycles techniques from previous ones and therefore a campaign evolves slowly over time. For example, two different spam emails may attempt to sell the same product under a different (misspelled) name. Due to their evolutionary nature campaigns can be grouped into families where adversarial drift can be captured and organised into distinct trends. It is often the case that attacks within the same family have a similar detection strategy.

Mind the Gap

The general advice here is to gain insight of the anomaly detector capabilities. Not only do researchers need to determine why false positive alarms are generated (or not generated in case of false negatives), but they also need to understand why it produces correct results. Consider the following case from the 1980s (Sommer & Paxson 2010). A Pentagon's project involved the use of a neural network to detect pictures of tanks. The classifier was indeed identifying correctly the pictures of tanks. However, it turned out the classifier was recognising the colour of the sky, as all the pictures of tanks were taken on a cloudy day.

Furthermore, the use of an ensemble of classifiers as discussed earlier provides *isolation* which closes the semantic gap (Kantchelian et al. 2013). Based on the assumption that malware campaigns are most cost-effective using a single family, the use of multiple classifiers can isolate malicious campaigns from one another and provide better understanding and accuracy to the human expert. Multi-family attack is of course possible but the adversary is required to spend more resources.

Finally, as discussed earlier redlining should be taken seriously in commercial systems as such cases can contravene the European data protection law (Anderson 2008).

2.2.6 Next Generation Intrusion Defence

Cohen proved that the detection of viruses is as hard as the halting problem (Anderson 2008). Therefore a perfect solution to the intrusion detection problem is not possible. Anderson (2008) classifies security failures into two categories, the ones that cause errors and those which do not. The first category of failures can be detectable (if we forget about Cohen's proof). Failures from the second category though, cannot be detected. Similarly, Ghorbani et al. (2010) state that intrusion detection relies on the assumption that intrusive behaviours are distinguishable from normal ones and can therefore be detected. Anderson (2008) states the following:

> "In general, you must expect that an opponent will always get past the threshold if he's patient enough and either does the attack very slowly, or does a large number of small attacks."

We have already discussed about the challenges of anomaly detection. Anderson (2008) admits that there is room for much improvement, and predicts that future research will focus on a fully coordinated monitoring between all network's levels (backbone, local network, host) and all levels of the protocol stack (network, transport, application). This prediction also applies to intrusion response e.g. for firewalls. We believe that machine learning could contribute to improve network defence by coordinating all these sensors and performing event correlation.

In addition, IDSs have been criticised for being passive, that is, no response action is executed beyond triggering an intrusion alarm when a suspicious event is detected. The alarm draws the attention of authority in order to decide accordingly for an appropriate response action. We further believe that machine learning could play a key role towards automated intrusion response, where we envision that novel response approaches could be learned. Such an autonomous defence system could also learn to choose which response approach among many alternatives is more suitable in different situations. Autonomous intrusion response gives rise to many challenges. The next section focuses on distributed denial of service (DDoS) attacks where we examine and discuss about all these new issues and challenges.

2.3 Distributed Denial of Service Attacks

DDoS attacks constitute a rapidly evolving and one of the most serious threats in the current Internet. Numerous incidents have been reported over the years; a few noteworthy are mentioned here (Greene 2011). In 2000 DDoS attacks brought high-profile websites like Yahoo, eBay, Amazon and CNN to their knees. Interestingly, the attacks were launched from compromised machines in governments, businesses, universities and organisations. In 2007, many Estonian websites for government and banks were hit by DDoS attacks. Twitter and Facebook were affected by DDoS attacks in 2009. Amazon servers were again hit by DDoS attacks in 2009 just a couple of days before Christmas. Anonymous (a hacktivist group) targeted Visa, MasterCard and PayPal in 2010, as a way to protest against the decision of the three to cut off WikiLeaks.

The average cost for an enterprise to defend against a DDoS attack is \$2.5 million (Kerner 2013). Beyond the financial loss caused by DDoS attacks, victims also suffer loss to their reputation which results in customer dissatisfaction and loss of trust. Common motivations are political and ideological hacktivism, vandalism, rivalries and extortion (Anstee et al. 2013).

The Arbor Network's worldwide security survey conducted by Anstee et al. (2013) among more than 290 companies and organisations reveals the following interesting findings:

1. 50% of the participants see 1-10 DDoS attacks per month, while 12% experience more than 100.

2. The average size[1] of a DDoS attack is 1.77 Gbps, while larger DDoS incidents are observed

[1]Size is calculated as the traffic rate (prior mitigation) that is directed towards the victim as typically measured in the upstream (e.g. Internet Service Provider (ISP) network).

in the range 2 to 10 Gbps (Kerner 2013). The largest DDoS attack reported in 2009, 2010 and 2013 was 49 Gbps, 100 Gbps and 309 Gbps respectively.

3. 62% of the attacks are currently less than 1 Gbps (Kerner 2013) and 88% of the attacks last for less than an hour.

4. 62% of the participants have seen increased demand for DDoS protection from their customers.

So what do these figures mean? As a result of the Internet's popularity and usefulness, there are many "interesting" targets and malicious (or simply ignorant) users that the DDoS threat is not going to disappear on its own (Keromytis et al. 2002). DDoS attacks grow in sophistication, frequency and size, and this demonstrates the need to adopt novel approaches to rapidly detect and respond to DDoS attacks.

2.3.1 Architecture

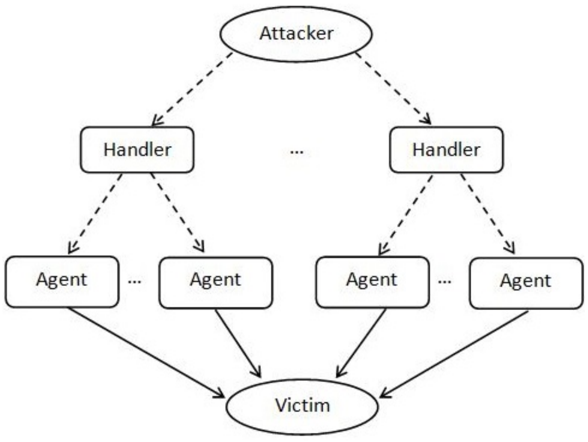
A DDoS attack is a highly coordinated attack; the strategy behind it is described by the agent-handler model (Douligeris & Mitrokotsa 2004; Specht & Lee 2004) as shown in Figure 2.5a. The model consists of four elements, the attacker, handlers, agents and victim. A handler (or master) and the agent (or slave or zombie or deamon) are hosts compromised by the attacker, which constitute the botnet. Specifically, the attacker installs a malicious software on vulnerable hosts to compromise them, thus being able to communicate with and control them. The attacker communicates with the handlers, which in turn control the agents in order to launch a DDoS attack. As of 2007, there are botnets with over half a million machines (Anderson 2008).

The basic agent-handler model can be extended by removing the handlers layer and allowing communication between the attacker and agents via Internet Relay Chat (IRC) channels (Douligeris & Mitrokotsa 2004; Specht & Lee 2004) as shown in Figure 2.5b. A more recent extension occurs at the architectural level, where the centralised control is replaced by a peer-to-peer architecture. These extensions make the DDoS problem orders of magnitude harder to tackle because they offer limited botnet exposure, a high degree of anonymity and they provide robust connectivity.

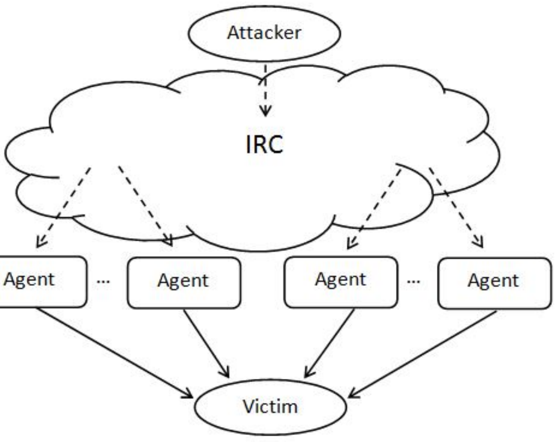
According to Anderson (2008), the botnet's compromised machines typically have the following life cycle. They are firstly used for targeted attacks. Then they are used for spamming. Once spam filters are able to stop spam they are used for applications such as phishing. Finally, they are used for DDoS attacks.

2.3.2 Taxonomy

Here we discuss material from the DDoS taxonomy proposed by Mirkovic & Reiher (2004). It is the most popular and to our opinion the most complete taxonomy. Other popular taxonomies include the ones by Douligeris & Mitrokotsa (2004) and Specht & Lee (2004).



(a) Agent-Handler Model



(b) IRC-based Model

Figure 2.5: DDoS Architecture

2.3.2.1 Source Address Validity

An attacker can either use the real IP source address of the compromised agent machines or use IP spoofing, that is, hiding their true identities by using a fake IP source address. According to Anderson (2008), most attackers do not use IP spoofing anymore. It is now much more difficult to send a packet with spoofed IP address using the Windows operating system. Furthermore, IP spoofed packets can be stopped using filtering techniques which are described at a later section. The most important reason however has to do with economics. Nobody came after the actual attackers because no telecommunications company would give network access to a competitor. Despite these reasons, DDoS attacks using spoofed addresses still occur. As already mentioned, the largest DDoS attack reported in 2013 (309 Gbps) involved IP spoofing. Lastly, there exist tools that forge all packet header contents (Hussain et al. 2003).

2.3.2.2 Exploited Weakness

Depending on the exploited weakness DDoS attacks can be classified as semantic (or vulnerability-based or low-rate) and brute-force (or flooding). Semantic attacks exploit a specific feature of a protocol or its implementation by sending malformed packets. Brute-force attacks send a high volume of seemingly legitimate requests to the victim which renders it incapable of providing normal service to its legitimate users. It is important to note that there is a fine line between the two since if a semantic attack is executed in volume, it can also be classified as brute-force (although in this case its volume is typically lower compared to a solely brute-force attack).

A semantic attack can be stopped by modifying the exploited protocol or by deploying a local filter (e.g. firewall). However, the firewall will not be able to stop brute-force attacks. Firstly because it may become a target itself due to the large volume of traffic and secondly because it will cause collateral damage, that is, when legitimate traffic is punished along the DDoS traffic. As we will discuss later the victim must request help from upstream filters. The focus of the thesis is on brute-force attacks.

A special type of brute-force attacks is the amplification (or reflector) attack. In an amplification attack, the attacker exploits the broadcasting feature of an amplification (or reflector) device such as a router or server. Specifically, the attacker sends to a number of amplifiers spoofed IP packets with the source address set to the victim's. These amplifiers will reply to the victim, thus causing a DDoS flood. It is important to note that while IP spoofing is optional for the rest of the DDoS attacks, it is a necessary requirement for amplification attacks.

According to Anderson (2008), "... the bad guys have turned increasingly to brute force ... Instead of using a handful of compromised machines to send out clever attacks via amplifiers using spoofed source addresses, the bad guys simply burn thousands of end-of-life botnet machines to send the bad packets directly. The rapier has been replaced with the Kalashnikov." However, DDoS amplification attacks still occur. As already mentioned, the largest DDoS attack reported in 2013 (309 Gbps) was an amplification attack.

Examples of Semantic Attacks

A common attack is the **TCP SYN flood** which exploits the three-way handshake in order to establish a TCP connection. The three-way handshake works as follows. A client sends a SYN (synchronise/start) request to a server. The server sends back a SYN/ACK (synchronise/acknowledge). The client then sends an ACK (acknowledge) in order to establish the TCP connection. In a TCP SYN attack, the attacker initiates a large number of SYN requests that never acknowledges. This leaves the server with too many half-open TCP connections and since it has a limited buffer for connections, the server remains unresponsive to new requests from legitimate users.

Slowloris (ArborNetworks 2010) is another common attack that opens HTTP connections with a web server but then goes idle. It only transmits a small number of bytes to keep the connection open, but actually never completes the request (an example is the HTTP GET request). The web server eventually will not be able to accept new requests from legitimate users.

An example of a malformed packet attack is when an attacker sends IP packets which contain the same address for both the source and destination. The victim's operating system cannot handle this confusion and the victim's machine crashes.

Examples of Brute-force Attacks

UDP flood is an example of a brute-force DDoS attack. The attacker sends a large number of UDP packets to random or specific ports of the victim. When a UDP packet is received the victim follows the next three steps. It first checks to see which application is listening to that port. The victim observes that no application is listening to that port. Finally, the victim replies back with an ICMP Destination Unreachable packet. If the victim receives a large number of UDP packets, it will need to generate a large number of ICMP packets thus causing its machine to crash. If IP spoofing is used the ICMP packets do not return back to the zombie machines.

Another example is the **ICMP Ping flood** attack. The attacker sends a large number of ICMP Echo Request (ping) packets to the victim to check whether it is alive. The victim normally replies back with ICMP Echo Reply (pong) packets. If the victim receives a large number of ping packets, it will need to generate a large number of pong packets thus causing its machine to crash.

Another common attack is the **HTTP GET flood** (ArborNetworks 2010). A GET request is initiated to make a legitimate request for web server or application resources. The attacker initiates a large number of HTTP GET requests that consumes a significant amount of the victim's resources, therefore the victim cannot respond to legitimate requests.

The first DDoS amplification attack observed is the **Smurf** attack. The zombies send ICMP Echo Request (ping) packets to a number of amplification devices (e.g. routers) with the return address spoofed to the victim's IP address. The amplification devices send back ICMP Echo Reply (pong) thus causing a flood towards the victim. We note that the amplifiers may also suffer from the high load of requests.

2.3.2.3 Attack Rate Dynamics

The majority of DDoS attacks use a constant-rate mechanism where agent machines generate packets at a steady rate. The DDoS impact is rapid but the large and continuous flood can be easily detected. It is the most cost-effective method for the attacker since it requires a minimal number of agent machines.

The attacker can also deploy a variable rate mechanism to delay or avoid detection and response. An increasing-rate mechanism gradually increases the rate of the attack which causes a slow exhaustion of the victim's resources. Attacks can also use a fluctuating rate mechanism for example a pulse attack. In a pulse attack the agent machines periodically start and resume the attack thus causing a periodic service disruption to the victim.

2.3.2.4 Persistence Agent Set

Typically the attacker deploys a constant agent set, that is, the attacker uses all the agent machines in a similar manner. However, the attacker can split the available agent machines into a number of groups. Groups can perform the same attack (e.g. constant-rate attack) or perform a different attack (e.g. constant-rate and pulse attacks). Furthermore, groups can either all be simultaneously active or just a subset of them.

2.3.2.5 Victim Type

Depending on the victim type DDoS attacks can target an application, host or network. A DDoS attack can target a specific application on the victim host thus becoming inaccessible to legitimate users. The detection of application-layer DDoS attacks is challenging for three reasons. Firstly, they typically appear legitimate at the transport layer. Secondly because other applications can still run normally (assuming that CPU is fairly shared among them) and lastly because a defence system must monitor each individual application. Application-layer DDoS attacks are mostly semantic and have usually a small volume. There has been an increase lately of this type of attacks due to the adoption of cloud computing (ArborNetworks 2010; Armbrust et al. 2010).

DDoS attacks can also target the victim host itself or its network such as a router or bottleneck link. Also, DDoS attacks can target the infrastructure of e.g. an ISP network or even of the Internet such as DNS servers and core routers. These attacks are usually brute-force i.e. they rely on volume and not on packet contents. Therefore they are easier to detect but as we will discuss later the victim must request help from upstream.

2.3.2.6 Victim Impact

Depending on the impact on the victim, DDoS attacks can be classified as disruptive or degrading. Disruptive attacks attempt to bring down the victim and completely deny service to its legitimate users. According to Mirkovic & Reiher (2004), all reported attacks belong to the first category. Depending on the possibility of recovery during or after an attack, a disruptive DDoS attack can be further classified as human-recoverable, self-recoverable or non-recoverable. Human-recoverable attacks require the intervention of the human operator. An example is when a DDoS attack causes the victim machine to crash or freeze and the human operator needs to reboot or reconfigure it. In the case of self-recoverable attacks the victim is equipped with mechanisms to recover without the intervention of the human operator. Non-recoverable attacks cause a permanent damage to the victim's hardware that requires replacement.

Degrading attacks attempt to degrade the service provided by the victim to its legitimate users, rather than completely denying the service. For example a degrading attack would cause the victim to provide a slower average service to its legitimate users. This could lead to customer dissatisfaction and therefore some of the customers could change a service provider. Moreover, the victim could spend money to upgrade its servers in order to meet the false demand. Degrading attacks are not popular since they cause an indirect damage to the victim. However, it is much more difficult to detect them and according to Mirkovic & Reiher (2004) the majority of defence techniques would fail to address this type of attack.

2.3.3 Defence Characteristics

2.3.3.1 Challenges

The DDoS threat is challenging for many reasons, including the following (Mirkovic et al. 2003). Firstly, the traffic flows originate from agent machines spread all over the Internet, which they all aggregate at the victim. Furthermore, the volume of the aggregated traffic is really large which is unlikely to be stopped by a single defence point near the victim. Also, the number of compromised agent machines is large, thus making an automated response a necessary requirement. Moreover, DDoS packets appear to be similar to legitimate ones, since the victim damage is caused by the total volume and not packet contents. A defence system cannot make an accurate decision based on a packet-by-packet basis. It requires the defender to keep some statistical data in order to correlate packets and detect anomalies, for example, "all traffic directed towards a specific destination address". Lastly, it is very difficult to traceback, that is, to discover even the agent machines, let alone the actual attackers, firstly because of the DDoS architecture model, and secondly because of IP spoofing.

It is evident that, to combat the distributed nature of these attacks, a distributed and coordinated defence mechanism is necessary where many defensive nodes across different locations cooperate in order to stop or reduce the flood.

2.3.3.2 Deployment Location

Figure 2.6 shows an abstract view of the Internet. It depicts the Internet backbone or core, four ISP networks ISP1-ISP4, six ISP customer networks N1-N6 and seven host machines H1-H7. Routers are denoted by circles. Hosts H2-H7 are compromised zombie machines which perform a DDoS attack at the victim H1. Defence can be deployed at the victim, intermediate or source networks.

Victim

Traditionally defence is deployed at the victim network. In Figure 2.6 the victim network is shown by N1. This is the place which suffers most from the DDoS impact and of course the victim is the most motivated to deploy and bear the cost of the defence mechanism. However, victim-end defence cannot stop a brute-force DDoS attack because it can become a target itself and can also inflict collateral damage because of the heavily aggregated traffic. Examples of victim-end defence include the IDS and the firewall (Ferguson 2000).

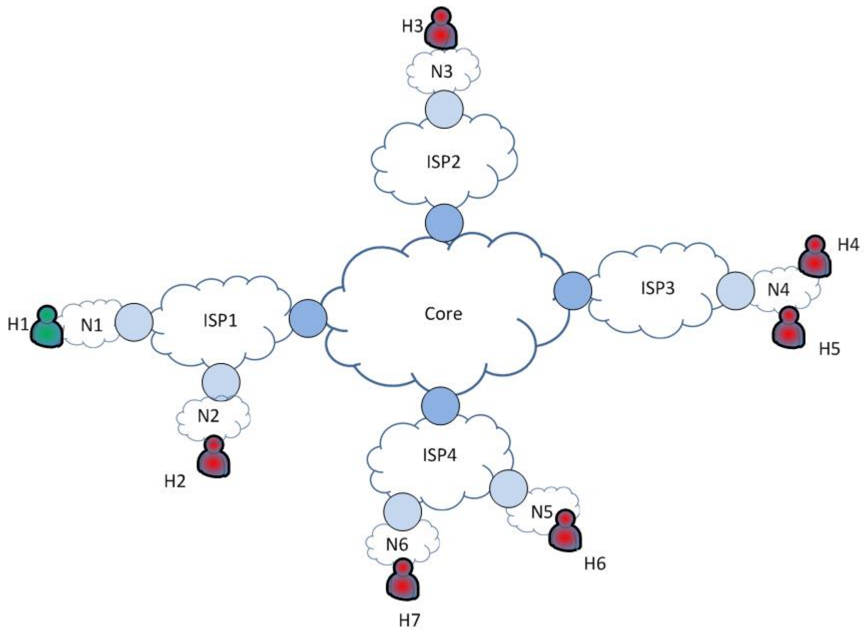


Figure 2.6: Abstract View of the Internet

Intermediate

In this case the victim requests (typically by providing a payment) from upstream networks to respond to the DDoS attack. In Figure 2.6 the intermediate network is ISP1. Depending on the collaboration degree it can also include the areas of Core and ISP2-ISP4, although it's questionable whether such collaboration can be achieved due to security and policy issues (Keromytis et al. 2002; Ioannidis & Bellovin 2002). Nevertheless, currently it seems that intermediate-end defence is a promising and plausible solution. An example of intermediate-end defence is route-based distributed packet filtering (Park & Lee 2001).

Source

In this case the source networks are requested to respond to the DDoS attack. In Figure 2.6 the source networks are N2-N6. This definitely constitutes the ideal solution to the DDoS problem since it is tackled directly at its origins (Mirkovic & Reiher 2004). However, motivation for source-end defence is very low; someone may be willing to spend money to protect himself against DDoS attacks, but he is much less prepared to pay to stop Amazon or Microsoft from being attacked (Yan et al. 2000). Nevertheless, we believe that if source-end DDoS defence is not enforced by legislation, such techniques seem rather unrealistic. An example of source-end defence is D-WARD (Mirkovic et al. 2002).

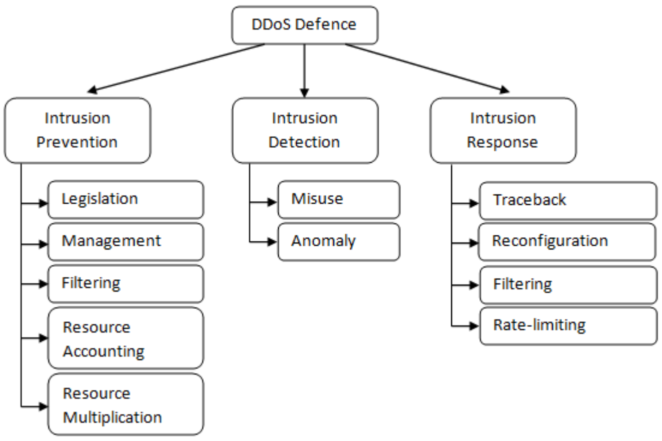


Figure 2.7: DDoS Defence Taxonomy

Multiple
Defence can be deployed in multiple places in order to take advantage of the benefits each location offers. Examples include Pushback (victim-end, intermediate-end) (Mahajan et al. 2002), Router Throttling (victim-end, intermediate-end) (Yau et al. 2005) and DefCOM (victim-end, intermediate-end, source-end) (Mirkovic et al. 2005).

2.3.4 Defence Techniques

In figure 2.7 we present a taxonomy of DDoS defence approaches based on the one proposed by Mirkovic & Reiher (2004). For each approach we describe representative mechanisms. Defence mechanisms are classified into three high-level categories, namely, intrusion prevention, intrusion detection and intrusion response.

2.3.4.1 Intrusion Prevention

Preventive mechanisms attempt to eliminate the possibility of an attack happening, or help the victim tolerate the attack without affecting its legitimate users.

Legislation
In early 2000s, extortion was a popular reason for the bad guys to perform DDoS attacks against bookmakers (Anderson 2008). The problem of extortion was fixed because of two non-technical reasons. Firstly the bookmakers got together and decided never again to pay any ransom and secondly three men were sent to prison for eight years.

Legislation is a complex matter. For example how do you handle online activism? If thousands of people send email to the government to protest against a decision, is this a DDoS attack (Anderson 2008)? Or how do you enforce source-end defence? It has been proposed that the owners of compromised machines should pay for the damage they cause (Yan et al. 2000). How-

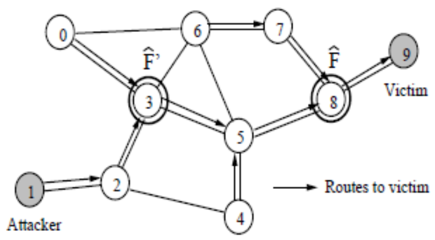


Figure 2.8: Distributed Route-based Filtering (Park & Lee 2001)

ever, achieving this remains an open question and may require fresh legislation that may vary from one country to another.

Management

Management mechanisms are about keeping your system's state in such a way that the possibility of being compromised (and thus taking part in an attack) or becoming a victim is minimised. Such mechanisms include keeping your system up-to-date by applying security patches, disabling unused services and disabling amplifiers (Douligeris & Mitrokotsa 2004).

Filtering

Filtering mechanisms drop network packets according to specific rules or criteria. A popular mechanism is the ingress filtering (Ferguson 2000) which monitors and blocks harmful inbound traffic to enter a network, for example dropping a packet whose IP address does not belong to a known domain prefix. Similarly egress filtering (Brenton 2007) monitors and blocks harmful outbound traffic, for example blocking a spoofed IP packet from leaving the network.

Route-based distributed packet filtering (Park & Lee 2001) is a technique aiming to stop spoofed IP packets based on route information. The technique is best described through an example. Figure 2.8 shows nine nodes which represent different autonomous systems ASs (an AS is a region under common administrative control). It also depicts with arrows all the possible routes to node 9 assuming a single path from any node to another. For simplicity we consider a single attacker in node 1 performing a DoS attack to the victim in node 9. Therefore, the actual attacker's route to the victim is 1, 2, 3, 5, 8 and 9. Without route-based packet filtering the attacker can disguise himself with IP addresses from nodes S = {0,1,2,3,4,5,6,7,8}.

When route-based packet filtering is deployed at node 8, the possible spoofable address space is now S={0,1,2,3,4,5}. Consider the case where the attacker disguises himself with an IP address from node 6. The filter deployed at node 8 - if aware of the topology - expects the packet to arrive from node 7. However the packet reaches node 8 from node 5 and hence it is discarded. With distributed filtering at node 3 the spoofable address space becomes S={1,2}. Consider the case where the attacker disguises himself with an IP address from node 4. The filter at node 3 does not expect any packets from node 4 therefore it is discarded.

Interestingly the approach is very effective in partial deployment. Deployment at only 18% of autonomous systems would have a major impact on DDoS traffic. The drawback of this approach is that filters require to know the network topology, and assuming they do, they also need to update it since the topology changes over time (Douligeris & Mitrokotsa 2004).

Resource Accounting

Resource accounting (Mirkovic & Reiher 2004) mechanisms regulate a user's access to resources according to his privileges or behaviour. A popular mechanism is the Secure Overlay Services (SOS) architecture (Keromytis et al. 2002); an overlay network is a set of nodes which communicate with one another atop of the underlying network. SOS allows communication with a server only with *confirmed* users. A user's packets need to be authenticated and authorised by SOS, before they are allowed to flow through the overlay to the server. Effectively, the overlay hides the server's location and drops illegitimate traffic. A drawback of this approach is that SOS routes packets through a series of overlay nodes which introduces a latency which is far from minimal.

Another limitation of SOS is that only pre-authorised users can access the server. In general, providing access only to pre-authorised users has the following limitations (Iyengar et al. 2010). Firstly, it may deter clients from using the service. Secondly, it may not be feasible to create a list of all users authorised to use a service. For example, this is the case for open e-commerce websites like eBay or Amazon. Lastly, it is difficult to ensure that authorised users will indeed behave benignly.

Some resource accounting mechanisms include challenge-based or "proof-of-work" approaches such as cryptographic puzzles. In (Feng & Kaiser 2012), a website's URL (HTML) tag is updated to include a cryptographic puzzle. More malicious clients are presented with more difficult puzzles based on historical data and the current server load. When a client's browser finds such a protected link it runs a server provided script (using JavaScript), to provide a solution to the puzzle. Depending on the solution, the client is assigned a priority level and service is provided accordingly. A limitation of this approach is that it cannot handle brute-force DDoS attacks.

Resource accounting mechanisms also include quality of service (QoS) regulation. In (Iyengar et al. 2010), a client first contacts a challenge server to obtain a puzzle. Upon successful solution it receives an initial trust token. Trust tokens encode the QoS level the client is eligible to receive from the protected server. A client's token is included in all the future requests to the server. A client that presents a valid token will be served at the priority level encoded in the token. The server updates the client's priority level based on its recent requests and the amount of server resources they have consumed. A limitation of this approach is that it cannot handle brute-force DDoS attacks.

Resource Multiplication
Resource multiplication (Mirkovic & Reiher 2004) mechanisms provide a very large amount of resources to enable the victim to tolerate the attack. The typical approach is load balancing (Douligeris & Mitrokotsa 2004), where network providers increase the bandwidth on critical infrastructure connections to prevent them from going down.

These mechanisms raise the bar on how many machines need to be compromised for an effective DDoS attack, but they are very expensive (Mirkovic & Reiher 2004). So far these mechanisms have been proved sufficient for those who can afford the cost (Mirkovic & Reiher 2004). However, there are continuing worries that gigantic DDoS attacks originating from monster botnets will occur in the future (Anderson 2008).

2.3.4.2 Intrusion Detection

Preventive mechanisms are important and essential, but they are not perfect. Furthermore, "it's often cheaper to prevent some of the attacks and detect the rest than it is to try to prevent everything" (Anderson 2008). Intrusion detection monitors the logs and network traffic, and triggers an intrusion alarm if a malicious or suspicious behaviour is detected.

Misuse Detection
Misuse or signature-based detection aims at identifying already known attacks by monitoring the traffic for signatures i.e. known characteristics of attacks. The disadvantage of this approach is that it cannot uncover novel intrusions. Pattern matching approaches are commonly used for misuse detection. Snort (Roesch 1999) is a well-known defence system which uses misuse detection.

Anomaly Detection
Anomaly detection aims at uncovering novel attacks by attempting to define or learn a notion of normal network activity; any activity which deviates from the normality profile is marked as intrusive. The disadvantage of this approach is that it usually suffers from a high rate of false positives and negatives. Anomaly detection typically makes use of advanced statistical techniques or machine learning. Machine learning anomaly detection techniques have been described and discussed in a previous section.

Hussain et al. (2003) state that the IP packets that have appeared after the DDoS impact are more likely to be illegitimate. Recall that the attacker activates the handlers, which in turn activate the zombie machines. When the traffic is observed near the victim, the activation processes will result in a ramp-up of the attack intensity. This is because of the variation in path latency, along with weak synchronisation of zombie machines' local clocks. However, this method is not robust since an intelligent attacker can create an artificial ramp-up.

D-WARD (Mirkovic et al. 2002) is an example of a system which uses statistical anomaly detection at the source network. It monitors two-way traffic statistics and compares them against

Algorithm 1 Basic Packet Marking (BPM)

```
// Packet marking procedure
for each router $R$ do
   for each packet $w$ do
      append $R$ to $w$
   end for
end for

// Path reconstruction procedure at victim
for any packet $w$ from attacker do
   extract path $(R_i..R_j)$ from $w$
end for
```

a normal traffic model consisting of a set of thresholds, for example the maximum allowed sending rate per flow is UDP_{rate}. D-WARD can suffer from a high rate of false alarms since the distinction between legitimate and DDoS traffic at the source network is challenging. Also, the heavy statistics gathering and per-packet processing is expensive and can introduce overhead and delays, although at the source network D-WARD will most likely experience moderate traffic volumes.

2.3.4.3 Intrusion Response

Response mechanisms aim at mitigating the DDoS impact on the victim, while keeping collateral damage levels to a minimum.

Traceback

Traceback mechanisms aim at identifying the agent machines responsible for the attack. The most popular traceback technique is packet marking where upstream routers mark IP packets so the attack route can be reconstructed (Savage et al. 2000). The Basic Packet Marking (BPM) appends each router's address to each packet as it traverses from the attacker to the victim. As a result, each packet arriving at the victim contains a complete list of the routers it traversed i.e. an attack path. The BPM algorithm is shown in Algorithm 1.

BPM has the following limitations; it infers router overhead and has a high demand for per-packet space requirement. To deal with these issues, Probabilistic Packet Marking (PPM) was introduced where routers probabilistically mark IP packets. After a sufficient number of trials the victim is able to reconstruct the attack route using the partial path information.

Packet marking attempts to automate the tedious manual process of traceback, where a network administrator used to contact his ISP to provide him with all the necessary information to identify the sources of an attack. Traceback can be very expensive though and it is virtually impossible to trace due to the large number of attack paths (Papadopoulos et al. 2003). It is also questionable whether it is useful to spend large amounts of resources to traceback and identify individual zombies when the actual attackers continue to operate unnoticed and uninhibited (Papadopoulos et al. 2003). This is not to say that traceback is not useful. On the contrary, it is necessary for identifying compromised hosts but it may not be suitable as the first line of defence.

Reconfiguration

Reconfiguration mechanisms alter the topology of the network in order to add more resources or isolate the attack traffic. *XenoService* (Yan et al. 2000) is a replication mechanism and works as follows. A number of ISPs install Xenoservers that offer web hosting services. A website is monitored and if reduction in the quality of service is experienced due to a surge in demand, the website is replicated to other Xenoservers. The attacker needs to attack all replication points in order to deny access to the legitimate users.

Replication techniques have the following limitations (Keromytis et al. 2002). They are not suitable in cases where information needs to be frequently updated (especially during an attack) or it is dynamic by nature (e.g. live audio or video stream). If sensitive information security is a major concern, engineering a solution that replicates sensitive information without any "leaks" is very challenging.

Filtering

Filtering mechanisms use information provided by the intrusion detectors to filter out i.e. to completely drop the attack traffic. Such an example is the following. Work conducted by Jung et al. (2002) reveals that during a DDoS attack to a website, most sent requests were generated by IP addresses that did not appear before. For example for the CodeRed worm only 0.6-14% of the IP addresses appeared before[2].

Taking into consideration these findings Peng et al. (2003) proposed history-based filtering. An IP address database is maintained storing the addresses of previous successful connections. When the victim network or website experiences a high level of congestion, incoming packets whose address is not in the database are filtered out.

The major drawback of this approach is that if an attacker becomes aware of this scheme, he could mislead the system to include IP address of the zombie machines in the database. And of course, there is always the possibility that a previously legitimate user has become one of the bad guys. Furthermore, collateral damage is unavoidable as new legitimate users or legitimate users that haven't been seen for a while will be punished as well.

Some commercial products also belong to this category like the Arbor PeakFlow (ArborNetworks 2014). Mirkovic & Reiher (2004) state that unless the traffic characterisation from the detection module is very accurate, the attackers might leverage such mechanisms as denial of

[2]The CodeRed worm appeared in June 2001. The first phase of its operation was to infect machines and turn them into zombies. More than 359000 machines were infected in just fourteen hours (Moore et al. 2002). The second phase was to launch a DDoS attack against the White House's website.

service tools.

Rate-limiting

Rate-limiting mechanisms drop some fraction of the network traffic as opposed to completely filtering it out. These mechanisms are typically used when the detection mechanism cannot precisely characterise the attack traffic i.e. when attack signatures cannot be derived. We are very interested in distributed rate-limiting approaches which are discussed in the next section.

2.3.5 Distributed Rate-limiting

This section describes three popular distributed and cooperative rate-limiting mechanisms, namely, Pushback, Router Throttling and DefCOM.

2.3.5.1 Pushback

The first and most influential work in the field is the Pushback mechanism proposed by Mahajan et al. (2002). Firstly, the authors define the *aggregate* as the traffic that is directed towards a specific destination address i.e. the victim (note that source addresses cannot be trusted because hosts can spoof traffic, disobey congestion signals etc). The authors view a DDoS attack as a router congestion problem. A local Aggregate-based Congestion Control (ACC) agent is installed on the victim's router which monitors the drop history. If the drop history deviates from the normal[3], the local ACC reduces the throughput of the aggregate by calculating and setting a rate-limit.

Completely shutting off the aggregate traffic is not allowed by the authors. The reason being that the aggregate traffic likely contains some legitimate traffic as well, and therefore dropping all the aggregate traffic facilitates the task of the attacker, which is to deny legitimate users access to the victim's service.

Pushback is a cooperative mechanism consisting of many ACC agents. The local ACC can optionally request from adjacent upstream ACC routers to rate-limit the aggregate according to a max-min fashion, a form of equal-share fairness, where bandwidth allocation is equally divided among all adjacent upstream routers. The max-min fairness algorithm is shown in Algorithm 2 (McKeown 2008).

Consider for example three contributing links with arrival rates of 2, 5 and 12 Mbps, and that the desired arrival rate from these links is 10 Mbps. The limits sent to each of the contributing links are 2, 4 and 4 Mbps respectively[4]. The general idea is that if a link carries more aggregate traffic is more likely to contain attack traffic. Rate-limiting is revised periodically. Of course,

[3]The authors distinguish between typical congestion levels (e.g. observed during peak times) and unusual or serious congestion levels caused by DDoS attacks (although serious congestion can occur due to other reasons as well e.g. a fiber cut).

[4]A popular alternative is the proportional-share fairness where bandwidth is allocated according to the needs of the contributing links. In the same example the limits sent to each of the contributing links would be 1.05, 2.63 and 6.32 Mbps.

Algorithm 2 Max-min Fairness

```
$N$ flows share a link with capacity $C$.
Flow $f$ wishes to send at a rate of $W(f)$ and is allocated a rate of $R(f)$.
repeat
  Pick the flow $f$ with the smallest requested rate
  if $W(f) \leq C/N$ then
    $R(f) = W(f)$
  else
    $R(f) = C/N$
  end if
  $N = N - 1$
  $C = C - R(f)$
until $N = 0$
```

max-min fairness is a generic notion and in practice rate-limits will be heavily guided from ISP policies such as tariff payments (Mahajan et al. 2001; Yau et al. 2001).

Rate limiting of the aggregate recursively propagates upstream towards its sources in a hierarchical fashion (incidentally providing a form of traceback). This is shown in Figure 2.9 where link $L0$ is highly congested due to a high-bandwidth aggregate. Local ACC $R0$ can protect the traffic from $L1 - L3$ that does not belong to the aggregate (i.e. traffic directed towards destinations other than the victim). However it cannot protect the (likely) legitimate traffic within the aggregate from $L1$. Therefore, Pushback will propagate from $R0$ to $R2$ and $R3$ and then to $R4$ and $R7$ and as a result aggregate traffic from $L1$, $L5$ and $L6$ will be protected. However, legitimate traffic arriving at router $R4$ will be punished along the attack traffic. An implementation of the Pushback mechanism is presented in (Ioannidis & Bellovin 2002). Pushback deployed by an ISP would be more effective if agreements with its peering ISPs are made on how to honour Pushback requests.

The major limitation of Pushback is that it still causes collateral damage, that is, when legitimate traffic is rate-limited along with the attack traffic. This is because the resource sharing starts at the congested point (i.e. $R0$ in Figure 2.9), where the traffic is highly aggregated and contains a lot of legitimate traffic within it. Moreover, Pushback is not effective in cases where attackers are uniformly distributed across the inbound links e.g. amplification attacks. In these cases the mechanism cannot concentrate rate-limiting on the malicious traffic.

Another limitation is that Pushback introduces heavy responsibilities to routers. A router is responsible for calculating rate-limits and sending *request* messages to its upstream peers; the upstream router applies the rate-limit upon receipt of the message. An upstream router sends a *status* message to its downstream router to report the arrival rate of the aggregate. Downstream routers send *refresh* messages so that upstream routers keep rate-limiting the aggregate.

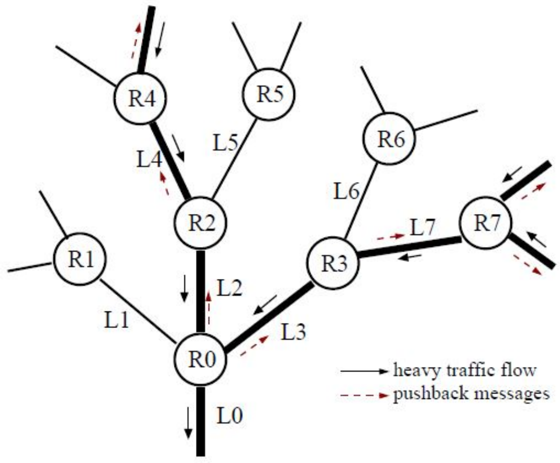


Figure 2.9: Illustration of Pushback (Mahajan et al. 2002)

2.3.5.2 Router Throttling

Another popular work is the Router Throttling mechanism by Yau et al. (2005). The authors view the DDoS attacks as a resource management problem, and they adopt a victim-initiated approach which according to Douligeris & Mitrokotsa (2004), similar techniques to throttling are used by network operators.

Recall that the aggregate is defined as the traffic that is directed towards a specific destination address i.e. the victim (note that source addresses cannot be trusted because hosts can spoof traffic, disobey congestion signals etc). The mechanism is described as follows. When a server operates below an upper boundary U_s, it needs no protection (this includes cases of weak or ineffective DDoS attacks). When the server experiences heavy load, it requests from upstream routers to install a throttle on the aggregate. In case the server load is still over the upper boundary U_s, the server asks from upstream routers to increase the throttle. If the server load drops below a lower boundary L_s, the server asks the upstream routers to relax the throttle. The goal is to keep the server load within the boundaries $[L_s, U_s]$ during a DDoS attack. Lastly, if the server load is below L_s and the next throttle relaxation raises it by an insignificant amount (i.e. less than ϵ) then the throttle is removed.

The set of all routers is given by R. Defence routers are determined by a positive parameter integer k, and are given by $R(k) \subseteq R$, which is defined as the set of routers that are either k hops away from the server, or less than k hops away but are directly attached to a host. The effectiveness of throttling increases with an increasing value of k, provided that routers in $R(k)$ belong to the same administrative domain (e.g. ISP) or collaborative ones. Consider for example the network topology shown in figure 2.10. It depicts a set of host machines represented by squares and a set of routers represented by circles. The host machines are traffic sources towards

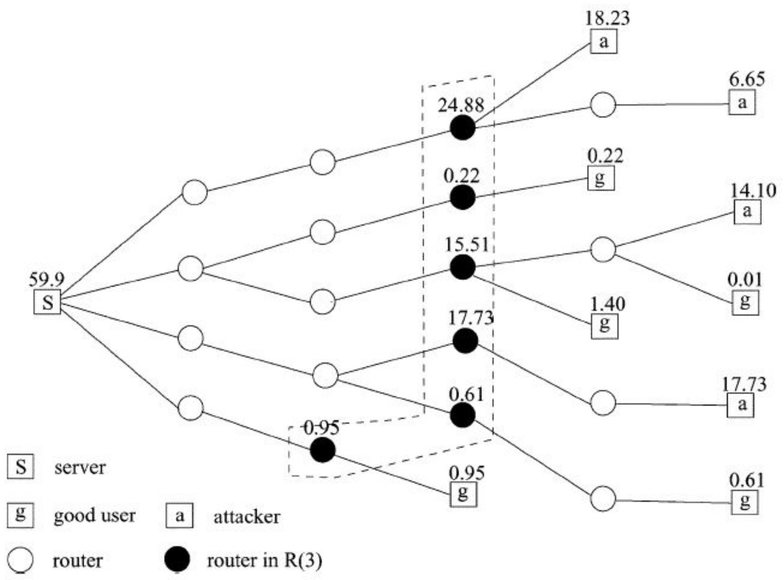


Figure 2.10: Illustration of Router Throttling (Yau et al. 2005)

the victim server denoted by S. The set $R(3)$ represents the deployment locations which are depicted by the shaded circles. The bottom router is included in the set $R(3)$, although it is only 2 hops away, because it is directly attached to a host.

Yau et al. (2005) present the Baseline throttling algorithm in which all upstream routers in $R(k)$ throttle traffic towards the server, by forwarding only a fraction of it. The algorithm is shown in Algorithm 3 and we will describe it through an example. In the topology shown in Figure 2.10 the number above each host denotes the current rate at which it sends to server S. Let $L_s = 18$, $U_s = 22$, $\alpha = 1/2$ and $\beta = 0.05$. Table 2.1 shows the trace of the throttle fraction for the Baseline algorithm. Initially the server experiences a total load of 59.9 which is way above U_s. The algorithm is invoked and causes each upstream router to drop a fraction of α i.e. half of their aggregate traffic. In the next round, a further reduction causes the server load to drop to 14.975 which is below L_s. The throttle is then relaxed by increasing it by β. The throttle is once more relaxed to reach 20.965 i.e. within $[L_s, U_s]$. At this point the forwarding rates at each upstream router are 8.708, 0.077, 5.4285, 6.2055, 0.2135 and 0.3325 respectively. Notice that the Baseline algorithm penalises all upstream routers equally, irrespective of whether they are well behaving or not.

The authors then propose the AIMD (additive-increase/multiplicative-decrease) throttling algorithm, which installs a uniform leaky bucket rate at each upstream router in $R(k)$ that achieves level-k max-min fairness. The algorithm is shown in Algorithm 4. We will use the same example topology shown in Figure 2.10 to describe the algorithm. Table 2.2 shows the trace of the throttle rate for the AIMD algorithm. The throttle is initialised to $r_s = (L_s + U_s)/4 = 10$ and we use an

Algorithm 3 Baseline Router Throttling

```
$\rho_{last} = -\infty$
while $(1)$ do
  monitor traffic arrival rate $\rho$ for time window $w$
  if $(\rho > U_s)$ then
    // throttle not enough; further restrict throttle rate
    send reduction signal to $R(k)$
  else if $(\rho < L_s)$ then
    // throttle too strong
    if $(\rho - \rho_{last} < \epsilon)$ then
      remove rate throttle from $R(k)$
      break
    else
      // relax throttle
      $\rho_{last} = \rho$
      send increase signal to $R(k)$
    end if
  else
    break
  end if
end while
```

Round	Throttle Fraction f	Server load
		59.900
1	0.5	29.950
2	0.25	14.975
3	0.30	17.970
4	0.35	20.965

Table 2.1: Throttle Fraction f for the Baseline Algorithm

additive step $\delta = 1$. Initially, when the algorithm is invoked it sends a throttle of 10 and brings the server load to 31.78. Since it is still higher than U_s the throttle is halved and drops below L_s to 16.78. The throttle is then increased by δ and the server load becomes 19.78 which is within the desired limits $[L_s, Us]$. At this point the forwarding rates at each upstream router are 6, 0.22, 6, 6, 0.61 and 0.95 respectively. This is a max-min fairness allocation at level-k. Notice that legitimate users are less affected compared to the Baseline algorithm. Notice that a careful parameter configuration is required; for example, had the additive step parameter been different, the trace of the throttle rate would also be different.

The AIMD throttling approach is effective in experiments involving both UDP and TCP traffic. TCP is interesting because the achieved throughput by a host depends on the rate at which ACKs are returned from the victim server; experiments show that the rate of successfully processed host requests during a DDoS attack is close to the original host request rate without an

Algorithm 4 AIMD Router Throttling

```
$\rho_{last} = -\infty$
$r_s = (L_s + U_s)/f(k)$
**while** $(1)$ **do**
    monitor traffic arrival rate $\rho$ for time window $w$
    send $r_s$ throttle to $R(k)$
    **if** $(\rho > U_s)$ **then**
        // throttle not enough; further restrict throttle rate
        $r_s = r_s/2$
    **else if** $(\rho < L_s)$ **then**
        // throttle too strong
        **if** $(\rho - \rho_{last} < \epsilon)$ **then**
            remove rate throttle from $R(k)$
            **break**
        **else**
            // relax throttle
            $\rho_{last} = \rho$
            $r_s = r_s + \delta$
        **end if**
    **else**
        **break**
    **end if**
**end while**
```

Round	Throttle Rate r_s	Server load
		59.900
1	10	31.78
2	5	16.78
3	6	19.78

Table 2.2: Throttle Rate r_s for the AIMD Algorithm

attack.

AIMD Router Throttling has two advantages over Pushback. It is more of an end-to-end approach initiated by the victim while Pushback is more of a hop-by-hop approach and therefore collateral damage is significantly reduced. Furthermore, routers have more simplified responsibilities compared to Pushback.

The approach has the following limitations though. Router Throttling (like Pushback) is not effective in the case of non-aggressive or "meek" attackers i.e. where an attacker's sending rate is similar to the rate of a legitimate user. The authors admit that "... our solution is then mainly useful in ensuring that a server under attack can remain functional within the engineered load limits" (Yau et al. 2005). In this case throttling fails to differentiate between legitimate and attack traffic and severe collateral damage is caused. However, this requires that an attacker compromises and recruits a considerably higher number of host machines (zombies) in order to launch an attack of the same effect.

Furthermore, the approach can suffer from stability problems because of system oscillations in order to settle the aggregate load to a desired level within the lower L_s and upper U_s boundaries. Even if the system does not suffer from stability problems, it still requires a number of oscillations which can cause an increase in the time required for the aggregate load to settle within the desired range. Performing throttling becomes challenging as the range $[L_s, U_s]$ becomes smaller.

Lastly, router Throttling is victim-initiated, that is, the victim controls and sends the throttle signals to the upstream routers. However, it is based on the assumption that either the victim remains operational during a DDoS attack or that a helper machine is introduced to deal with the throttle signalling. The first assumption can be violated in a real-world scenario. As far as the second assumption is concerned, the helper machine can also become a target of the attack. In essence, the problem may arise because the existing throttling approach is victim-initiated i.e. it has a single point of control. In other words, although it offers a distributed response, it is still a centralised approach.

2.3.5.3 DefCOM

According to Mirkovic et al. (2003, 2005) and Oikonomou et al. (2006) an effective DDoS defence must have three characteristics; these are accurate detection, effective response and traffic profiling. Accurate detection refers to the ability of spotting any signs of victim's service degradation. Effective response refers to the ability of reducing the DDoS flood to manageable levels. Traffic profiling refers to the ability of differentiating between legitimate and attack traffic. This is typically achieved by the incorporation of an anomaly detection/traffic differentiation component. Note that Pushback and Router Throttling lack the traffic profiling functionality.

Furthermore the authors propose that attack detection should occur at the vicinity of the victim since it will most accurately spot any service degradation. Attack response should not occur at the victim network since it can be overwhelmed by large floods. Both intermediate-end and source-end response are good choices but the authors propose the former as it typically requires less deployment points. Traffic profiling is expensive since it relies on heavy statistics-gathering and per-packet processing. The authors propose traffic profiling to be held in source networks since they experience moderate traffic volumes.

Mirkovic et al. (2003, 2005) and Oikonomou et al. (2006) propose DefCOM which combines D-WARD (Mirkovic et al. 2002) classifiers (described earlier in section 2.3.4.2) for traffic differentiation. We illustrate its operation through the example shown in Figure 2.11. It depicts the victim node V, three legitimate nodes A, B and D, two attack nodes E and F, two D-WARD classifiers $C1$ and $C2$, a rate-limiter RL and an alert generator AG.

When the victim experiences service disruption it sends an alarm message to all nodes participating in the DefCOM overlay. In our example the alert generator AG sends an alarm message to RL, $C1$ and $C2$. These nodes deploy secure packet stamping to form parent-child relation-

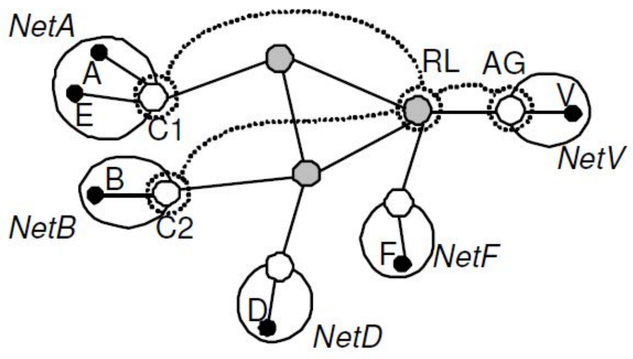


Figure 2.11: Illustration of DefCOM (Mirkovic et al. 2005)

ships thus constructing the traffic tree with the victim being the root. The traffic tree with root AG is shown by the dotted lines. Note that a parent and child may not be adjacent i.e. their distance may be more than a hop away.

The rate-limits are propagated from the victim to the rate-limiters (intermediate nodes), and then from the rate-limiters to the D-WARD classifiers (source nodes). This is similar to the Pushback mechanism. In our example, rate-limits are propagated from AG to RL, and from RL to $C1$ and $C2$.

Each D-WARD node stamps a packet as *legitimate* or *monitored*. Since a classifier has to obey the rate-limit imposed by its parent node, it dedicates the bandwidth to packets marked as legitimate and if there is some bandwidth left it will allow some monitored packets to leave the source network (so a classifier has the rate-limiting functionality as well). In our example, classifier $C1$ will stamp packets from node A as legitimate and packets from node E as monitored. Since it must obey the rate-limit imposed by RL, it will dedicate its bandwidth to node A and the rest, if available, to node E.

The intermediate nodes (rate-limiters) also must obey the rate-limit imposed by their parent nodes. Each rate-limiter dedicates bandwidth to packets marked as legitimate, then to packets marked as monitored (since they have already been policed/rate-limited at D-WARD nodes), and then to unstamped packets. In our example, node RL must obey the rate-limit imposed by AG. It will dedicate its bandwidth to packets from A and B (marked as legitimate), then to packets from E (marked as monitored and already policed/rate-limited at D-WARD node $C1$), and then to D and F. Unstamped legitimate traffic from D suffers, but this gives a motive to its network administrator to amend the situation by deploying a D-WARD classifier.

DefCOM outperforms the Pushback mechanism because it incorporates an anomaly detection/traffic differentiation component (D-WARD) which minimises collateral damage. However it has the following limitations. It is mostly effective in full deployment, but this is unlikely to happen (unless enforced by legislation) since it requires the participation of source networks.

Furthermore, it is extremely difficult for the traffic profilers i.e. D-WARD classifiers to differentiate between legitimate and attack traffic firstly because they reside at the source networks, and secondly because there is no coordination between the traffic profilers since D-WARD classifiers are independent of each other. Lastly, there are security concerns regarding malicious classifiers.

2.4 Summary

DDoS attacks present an incredibly complex problem. As discussed, preventive mechanisms (such as the firewall) are essential but imperfect, and therefore not adequate for solving the problem (Anderson 2008). Many response mechanisms have been proposed but each only partially tackles the problem. For instance, replication techniques are not suitable under conditions of dynamic and/or sensitive information, while traceback techniques have been criticised for not being suitable as the first line of defence (Papadopoulos et al. 2003).

Distributed rate-limiting has been shown to be a promising approach to respond to brute-force DDoS attacks. However, as discussed, existing approaches (such as Router Throttling (Yau et al. 2005)) come with limitations. Multiagent reinforcement learning is a serious candidate for a novel distributed rate-limiting approach that tackles these limitations and offers new benefits.

Despite the elegant theory behind temporal difference learning and the different approaches to tackle scalability (described in Section 2.1.5), MARL still experiences difficulties when scaling-up to real-world applications. There have only been a limited number of successful large-scale applications such as playing backgammon (Tesauro 1994) and robotic soccer (Stone et al. 2005), and more recently air traffic management (Tumer & Agogino 2007) and distributed sensor networks (HolmesParker et al. 2013; Colby & Tumer 2013).

Scalability is one of the most important aspects of a RL-based mechanism, since a non-scalable system will never be considered, let alone adopted, for wide deployment by a company or organisation. Furthermore, it should be apparent by now that coordination among the learning agents is a recurring issue. As discussed, communication between learning agents is not always possible or desirable. Therefore, decentralised coordination is a major challenge in multiagent learning.

The goal of our work is to design a decentralised coordinated agent-based defensive system which is highly scalable and effective to realistic DDoS attack scenarios.

CHAPTER 3

Multiagent Router Throttling

This chapter introduces Multiagent Router Throttling, a novel agent-based distributed approach to router throttling for responding to DDoS attacks. Multiagent Router Throttling consists of multiple independent reinforcement learning installed on a set of upstream routers, and each agent learns to rate-limit or throttle traffic towards a victim server.

The chapter starts by describing the network model used throughout this thesis and states any assumptions made. It then presents the architecture and provides the design details of our proposed approach. One of the novel characteristics of our approach is that it has a decentralised architecture and provides a decentralised coordinated response to the DDoS problem. The single point of control is eliminated thus leading to high resilience against DDoS attack.

Furthermore, we examine the behaviour of our approach in a series of increasingly sophisticated attack rate dynamics, and it is demonstrated that Multiagent Router Throttling outperforms a baseline approach, and either outperforms or achieves the same performance as a popular state-of-the-art throttling approach from the network security literature. Lastly, the behaviour of our approach is captured both within an advanced network simulator and a network emulator.

3.1 Network Model and Assumptions

We adopt the network model used by Yau et al. (2005). A network is a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. All leaf nodes are hosts and denoted by H. Hosts can be traffic sources and are not trusted because they can spoof traffic, disobey congestion signals etc. An internal node represents a router, which forwards or drops traffic received from its connected hosts or peer routers. The set of routers are denoted by R, and

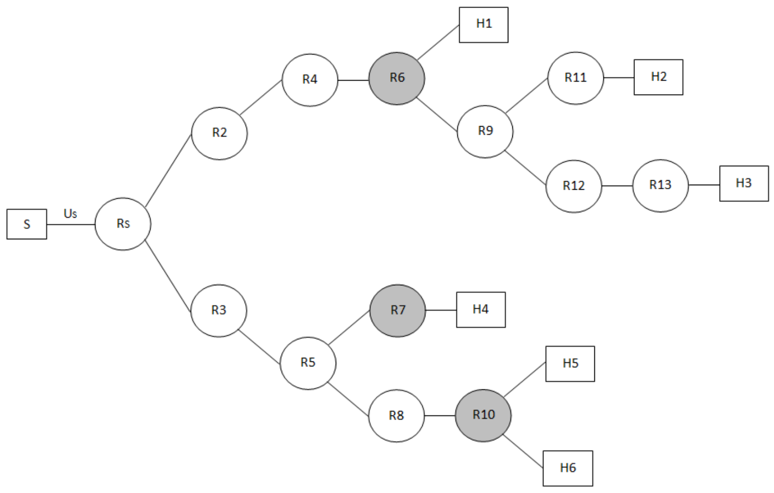


Figure 3.1: Network Topology Showing Defensive Routers

they are assumed to be trusted, i.e. not to be compromised. This assumption is realistic since it is much more difficult to compromise a router than an end host or server, because routers have a limited number of potentially exploitable services (Keromytis et al. 2002). The set of hosts $H = V - R$ is partitioned into the set of legitimate users and the set of attackers. A leaf node denoted by S represents the victim server. Consider for example the network topology shown in Figure 3.1. It consists of 20 nodes, these are, the victim server denoted by S, 13 routers denoted by R_s and $R2 - R13$ and six end hosts denoted by $H1 - H6$, which are traffic sources towards the server.

A legitimate user sends packets towards the victim server S at a rate r_l, and an attacker at a rate r_a. We assume that the attacker's rate is significantly higher than that of a legitimate user, that is, $r_a >> r_l$ (recall that dropping traffic based on source addresses can be harmful because, as mentioned, hosts cannot be trusted). This assumption is based on the rationale that if an attacker sends at a similar rate to a legitimate user, then the attacker must recruit a considerably larger number of agent hosts (zombies) in order to launch an attack with a similar effect (Yau et al. 2005). A server S is assumed to be working normally if its load r_s is below a specified upper boundary U_s, that is, $r_s \leq U_s$ (this includes cases of weak or ineffective DDoS attacks). The rate r_l of a legitimate user is significantly lower than the upper boundary i.e. $r_l << U_s$ where U_s can be determined by observing how users normally access the server.

3.2 Multiagent Router Throttling

In this chapter we propose Multiagent Router Throttling, a novel throttling approach where multiple reinforcement learning agents learn to rate-limit or throttle traffic towards a victim server. One of the novel characteristics of Multiagent Router Throttling is its decentralised architecture and response to the DDoS threat.

The original throttling approach (Baseline and AIMD techniques) by Yau et al. (2005) (described in Section 2.3.5.2) is victim-initiated, that is, the victim controls and sends the throttle signals to the upstream routers. However, it is based on the assumption that either the victim remains operational during a DDoS attack or that a helper machine is introduced to deal with the throttle signalling (Yau et al. 2001). The first assumption can be violated in a real-world scenario. As far as the second assumption is concerned, the helper machine can also become a target of the attack.

In essence, the problem may arise because the existing throttling approach is victim-initiated i.e. it has a single point of control. In other words, although it offers a distributed response, it is still a centralised approach. Our proposed approach consists of multiple independent reinforcement learning agents that learn to rate-limit or throttle traffic towards the victim server, thus providing a decentralised response to DDoS attacks. The architecture and design of our proposed approach is described below.

3.2.1 Agent Selection

We adopt the selection method used by Yau et al. (2005). Reinforcement learning agents are installed on locations that are determined by a positive integer k, and are given by $R(k) \subseteq R$. $R(k)$ is defined as the set of routers that are either k hops away from the server, or less than k hops away but are directly attached to a host. The effectiveness of throttling increases with an increasing value of k, provided that routers in $R(k)$ belong to the same administrative domain e.g. an Internet Service Provider (ISP) or collaborative domains. Consider for example the network topology shown in Figure 3.1. Learning agents are installed on the set $R(5)$, which consists of routers $R6, R7$ and $R10$. Router $R6$ is included in the set $R(5)$, although it is only four hops away from the server, because it is directly attached to the host $H1$.

3.2.2 State Space

The *aggregate* is defined as the traffic that is directed towards a specific destination address i.e. the victim (Mahajan et al. 2002). Each agent's state space consists of a single state feature, which is its aggregate *load*. The aggregate load is defined as the aggregate traffic that has arrived at the router over the last T seconds, which is called the monitoring window. The monitoring window should be set to be about the maximum round trip time between the server S and a router in $R(k)$ (Yau et al. 2005). The time step of the learning algorithm is set to be the same as the monitoring window size.

3.2.3 Action Space

Each router applies throttling via probabilistic packet dropping. For example, action 0.4 means that the router will drop (approximately) 40% of its aggregate traffic towards the victim server, thus setting a throttle or allowing only 60% of it to reach the server. Each action is applied

Algorithm 5 Global (G) Reward Function

```
if $loadRouter_{server} > U_s$ then
    // Punishment
    $r = -1$
else
    // Reward in $[0, 1]$
    $r = legitimateLoad_{server} / legitimateLoad_{total}$
end if
```

throughout the time step. Completely filtering or shutting off the aggregate traffic destined to the server is prohibited, that is, the action 1.0 (which corresponds to 100% drop probability) is not included in the action space of any of the routers. The reason is that the incoming traffic likely contains some legitimate traffic as well, and therefore filtering the incoming traffic facilitates the task of the attacker, which is to deny all legitimate users access to the server (Mahajan et al. 2002).

3.2.4 Global Reward

Each agent has the same reward function and therefore receives the same reward or punishment. The system has two important goals, which are directly encoded in the reward function. The first goal is to keep the server operational, that is, to keep its load below the upper boundary U_s. When this is not the case, the system receives a punishment of -1.

The second goal of the system is to allow as much legitimate traffic as possible to reach the server during a period of congestion. In this case, the system receives a reward of $L \in [0, 1]$, where L denotes the proportion of the legitimate traffic that reached the server during a time step. We consider that legitimate users are all of equal importance, therefore there is no prioritisation between them. The global (G) reward function is shown in Algorithm 5.

At this point we discuss the availability of L. In the case of offline learning, which is the focus of this chapter, the victim can keep track of and identify the legitimate traffic. This is because the defensive system can be trained in simulation, or in any other controlled environment (e.g. wide-area testbed, small-scale lab) where legitimate traffic is known a priori, and then deployed in a realistic network, where such knowledge is not available. In the case of online learning i.e. when the system is trained directly in a realistic network, L can only be estimated. Online learning is the focus of Chapter 5.

Lastly, since this chapter (and the next one) deals with offline learning, a separate training process is needed for each potential victim (which has made an agreement with its ISP to protect it in case of a DDoS attack). This is relaxed in Chapter 5 which deals with online learning.

3.2.5 Agent Learning

Due to our agent, state/action space and reward function selection, the network/throttling domain only needs to utilise immediate rewards. Furthermore, due to the nature of the network domain, a current network state has not necessarily been entirely affected by the actions taken by the agents at the previous time step. This is because the domain is highly probabilistic and exhibits unpredictable behaviour for example, at any time a DDoS attack can be initiated or stopped, more attackers can join or withdraw during an attack, attackers can alter their strategy which may be known or unknown to the network operator, legitimate users can start or quit using the victim's service, legitimate users can also alter their behaviour, routers can fail, network paths can change etc.

For these reasons we are only interested in immediate rewards, therefore we have set the discount factor to $\gamma = 0$. Note that for a complex delayed-reward function a different value for γ may be required. We use the popular SARSA (Sutton & Barto 1998) reinforcement learning algorithm and each agent uses the following update formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r - Q(s,a) \right] \tag{3.1}$$

3.3 Experimental Setup

Our experimental setup is based on work done by Yau et al. (2005). As a convention, bandwidth and traffic rates are measured in $Mbit/s$. The bottleneck link $S - R_s$ has a limited bandwidth of U_s, which constitutes the upper boundary for the server load. The rest of the links have an infinite bandwidth. Legitimate users and attackers are evenly distributed, specifically each host is independently chosen to be a legitimate user with probability p and an attacker with probability $q = 1 - p$. Parameters p and q are set to be 0.6 and 0.4 respectively. Legitimate users and attackers are chosen to send fixed size (1000 $bytes$) UDP traffic at constant rates, randomly and uniformly drawn from the range $[0, 1]$ and $[2.5, 6]$ $Mbit/s$ respectively. We refer to an *episode*, as an instance of the network model just described.

Our goal in this chapter is to investigate the design of our novel approach and evaluate its performance and potential deployability. For this reason, not all of the model parameters have been optimised. These model parameters, despite not being optimised, are set to reasonable values based on the reinforcement learning literature (Sutton & Barto 1998). Parameter tuning is performed in a later chapter.

Reinforcement learning agents use a linearly decreasing ϵ-greedy exploration strategy with an initial $\epsilon = 0.4$ and the learning rate is set to $\alpha = 0.1$. We discretise the continuous action space into ten actions: $0.0, 0.1, ..., 0.9$ which correspond to $0\%, 10\%..., 90\%$ traffic drop probabilities (recall from Section 3.2.3 that action 1.0 is prohibited). We use function approximation, specifically Tile Coding (Sutton & Barto 1998), for the representation of the continuous state space. The state feature (router load) of a router that sees traffic from one, two and three hosts is split into 6, 12 and 18 tiles per tiling respectively. In all three cases, the number of tilings is 8. Q-values are initialised to zero.

We evaluate our proposed approach against the Baseline and the popular AIMD throttling techniques from the network security literature (described in Section 2.3.5.2). Their control parameters are configured based on values or range of values recommended by their authors (Yau et al. 2005).

3.4 Simulation Experiments

We have extensively discussed the evaluation issues of network defensive systems in Section 2.2.5. Taking everything into consideration we have decided to conduct experiments in simulation environments.

Experiments performed in this section were conducted using the popular ns-2 network simulator. ns-2 is an advanced, open-source and free network simulator that offers abstraction, scenario generation and extensibility (Breslau et al. 2000). Abstraction refers to the simulation at different levels of granularity (network, transport, application). Scenario generation refers to the creation of complex traffic patterns, topologies and dynamic events. Extensibility allows users to add new functionality to the simulator. The last point is crucial since it enables us to incorporate our RL functionality. Furthermore, there is an active research community[1].

In this chapter, we are interested in offline learning. Notice the two different phases of each experiment, namely, offline training and evaluation. During the training of our system the victim keeps track of and distinguish between legitimate and attack traffic (requirement for the reward function). However, we particularly emphasise that this is not the case during the evaluation of our system. The rationale behind this is that the defensive system can be trained in simulation, or in any other controlled environment (e.g. wide-area testbed, small-scale lab), where legitimate and attack traffic is known a priori, and then deployed in a realistic network, where such knowledge is not available. Offline learning experiments aim at learning a universal policy, that is, the "best" response for all the model instances the network might be found in. For evaluation, each agent's policy is initialised to its universal policy learnt during offline training.

Offline Learning and Universal Policy

Before moving to the description of our experiments we will present a simple example of what a universal policy might mean. A universal policy represents the "best" response for all possible situations the network might be found in. Consider the network topology in Figure 3.1 with $U_s = 8$, and the two situations or network instances A and B shown in Tables 3.1a and 3.1b respectively. For example, in situation A hosts H1, H2 and H3 send at a rate of 1, 1 and 5 respectively and router R6 observes an aggregate load of 7.

The attackers in situation A are the hosts H3 and H6, while in situation B is the host H3. If we take situation A individually, the best actions (i.e. dropping probabilities) for the reinforcement

[1]To the best of our knowledge, at the time of writing there is a book (Issariyakul & Hossain 2011) on ns-2, several online tutorials, a forum and two Facebook groups.

H1	H2	H3	H4	H5	H6
1	1	5	1	1	5

R6	R7	R10
7	1	6

(a) Instance A

H1	H2	H3	H4	H5	H6
1	1	5	1	1	1

R6	R7	R10
7	1	2

(b) Instance B

Table 3.1: Examples of Network Instances

learning agents installed on R6, R7, and R10 are 0.1, 0.0 and 0.9 respectively. This brings the victim load to 7.9 (below U_s) and allows 2.9 (out of 4) of legitimate traffic to reach the server. If situation B is considered individually, the best actions are 0.3, 0.0 and 0.0 respectively. This brings the victim load to 7.9 (below U_s) and allows 4.4 (out of 5) of legitimate traffic to reach the server. The first thing to notice is that even if the best actions are executed different rewards can be yielded for example $4.4/5$ is higher than $2.9/4$.

An example of a universal policy for each agent that does well in both instances is when agents perform the actions 0.3, 0.0 and 0.7 respectively. In network instance A, this will bring the victim load to 7.7 (below U_s) and allow 2.7 (out of 4) of legitimate traffic to reach the server. In instance B, this will bring the victim load to 6.5 (below U_s) and allow 3 (out of 5) of legitimate traffic. The second thing to notice is that a universal policy ("best" for all) is likely to be worse that an individual best policy i.e. $2.7/4 < 2.9/4$ (for instance A) and $3/5 < 4.4/5$ (for instance B).

3.4.1 Attack Rate Dynamics

The majority of DDoS attacks use a constant-rate mechanism where agent machines generate traffic at a steady rate (Mirkovic & Reiher 2004). The DDoS impact is rapid but the large and continuous flood can be easily detected. It is also the most cost-effective method for the attacker. The attacker can also deploy a variable rate mechanism to delay or avoid detection and response.

These series of experiments aim at investigating how Multiagent Router Throttling behaves in scenarios with different patterns of attack rate dynamics with different sophistication levels. We emphasise that these patterns have not been previously experienced by our system during the offline training period. The patterns of attack rate dynamics used for our experiments are

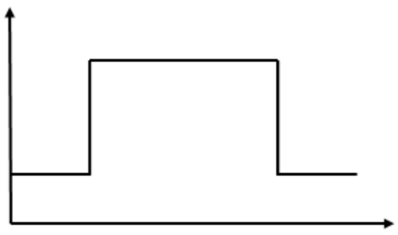
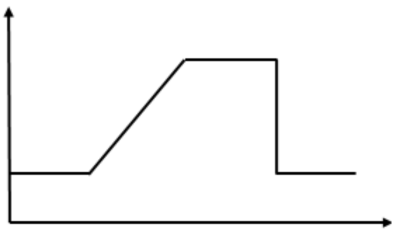


Figure 3.2: Constant-rate Attack


(a) Increasing-rate Attack

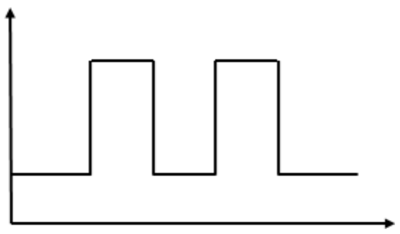

(b) Pulse Attack

Figure 3.3: Variable-rate Attack Dynamics

described below (Mirkovic & Reiher 2004):

- **Constant-rate attack:** The maximum rate is achieved immediately when the attack is started. This is illustrated in Figure 3.2.
- **Increasing-rate attack:** The maximum rate is achieved gradually over the attack period. We choose the attack rate to gradually increase and reach its maximum halfway through the attack period. This is illustrated in Figure 3.3a.
- **Pulse attack:** The attack rate oscillates between the maximum rate and zero. The duration of the active and inactive period is the same and represented by T. We create two different attacks namely the high and low pulse attacks which have a period of $T = 5$ and $T = 2$ time steps respectively. This is illustrated in Figure 3.3b.
- **Group attack:** Attackers are split into two groups and each group performs simultaneously a different attack pattern. We choose the first group to perform a constant-rate attack and the second to perform a low pulse attack.

3.4.1.1 Topology with 8 Nodes and 2 RLs

The network topology used for this experiment is shown in Figure 3.4. The bottleneck link $S - R_S$ has a limited bandwidth of $U_s = 6$ and all links have a delay of $10ms$. Defensive agents are installed on the set $R(2)$, i.e. routers $R2$ and $R3$.

The system is trained for 62500 episodes. At the start of each episode a new network instance is generated i.e. we re-choose the legitimate users, attackers and their rates according to the

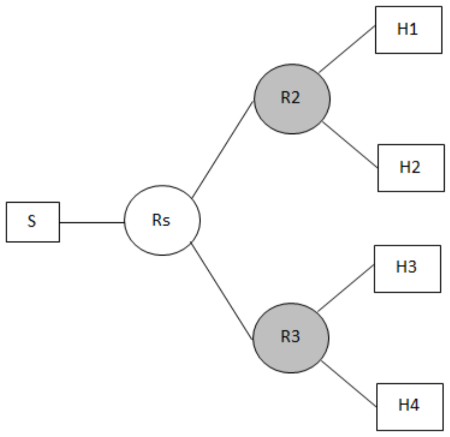


Figure 3.4: Network Topology with 8 Nodes and 2 RLs

network model. Attackers use the constant-rate attack pattern. Exploration is stopped after the 50000th episode. Each episode runs for $60s$ and both legitimate and DDoS traffic starts at 0. The monitoring window or time step of an agent is set to $2s$. As discussed earlier, the system training attempts to learn a universal policy for all episodes i.e. network instances; we emphasise that the learning environment is highly dynamic.

We plot the global reward at the last time step of each episode, averaged over 20 repetitions (i.e. over 20 universal policies). Training results are presented in Figure 3.5 and it is clear that the system learns and improves over time until it converges. Because of the probabilistic nature of the environment, different rewards will be yielded in each different episode (as demonstrated by the small example in Section 3.4) and hence the shape of the graph.

For evaluation we randomly sample 50 episodes each of a duration of $120s$. Legitimate traffic is started at $t = 0s$ and stopped at $t = 120s$. Attack traffic lasts for $100s$; it is started at $t = 5s$ and stopped at $t = 110s$. Each reinforcement learning agent uses its policy learnt during offline training. We evaluate our approach against the Baseline and the popular AIMD throttling techniques (Yau et al. 2005) which were described in Section 2.3.5.2. These approaches use a lower boundary of $L_s = 4$ and the same upper boundary of $U_s = 6$ as our approach.

Evaluation is performed using the different patterns of attack rate dynamics described earlier. Notice that these patterns have not been previously seen by our system during the training period. Performance is measured as the percentage of legitimate traffic that reached the server; the higher the value on the graph the better. Figures 3.6a - 3.6d show the average performance over the 50 episodes for the 20 policies learnt during the system training, for each of the attack rate dynamics respectively; error bars show the standard error around the mean. We refer to our proposed approach as MARL. Experimental results reveal the following:

1. In all scenarios, MARL significantly outperforms the Baseline throttling approach. Simil-

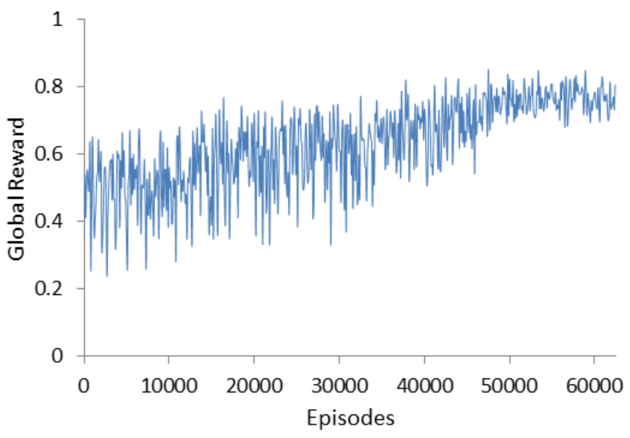
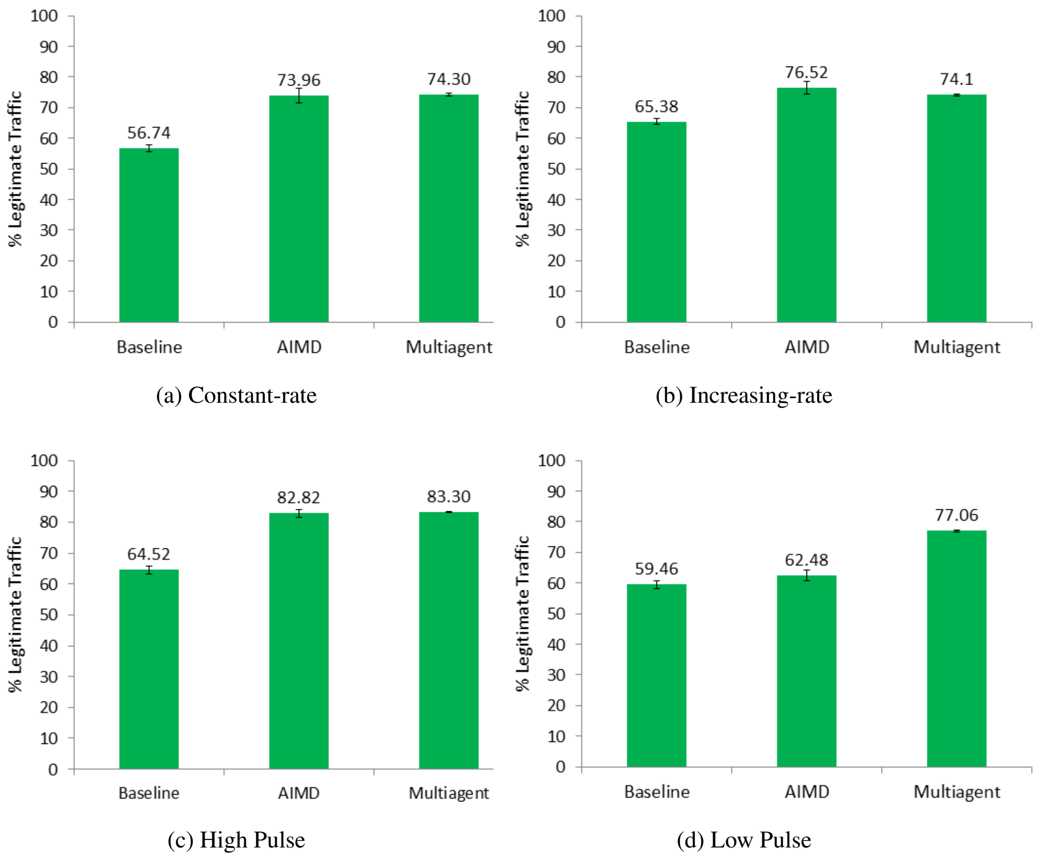


Figure 3.5: Training Results for 2 RLs



(a) Constant-rate

(b) Increasing-rate

(c) High Pulse

(d) Low Pulse

Figure 3.6: Evaluation Results for 2 RLs

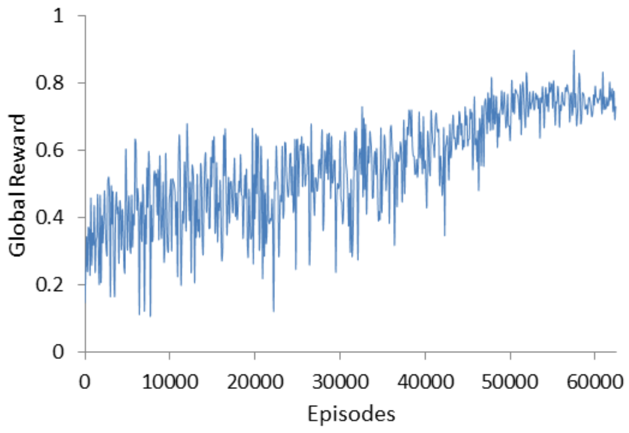


Figure 3.7: Training Results for 3 RLs

arly, AIMD outperforms the Baseline approach in all scenarios.

2. The MARL approach has the same performance as the AIMD approach in the scenarios involving the constant-rate and high pulse attacks. The AIMD approach performs slightly better in the scenario with the increasing-rate attack.

3. MARL significantly outperforms the AIMD approach in the scenario involving the low pulse attack.

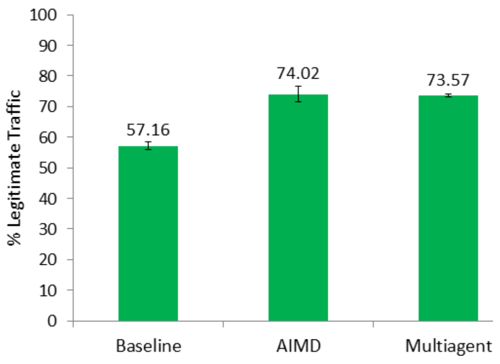
3.4.1.2 Topology with 20 Nodes and 3 RLs

To further investigate how our proposed approach deals with the different attack rate dynamics we repeat the same experiment using a larger and more realistic network topology. The network topology used for this experiment is shown in Figure 3.1. The bottleneck link $S - R_S$ has a limited bandwidth of $U_s = 8$ and all links have a delay of $10ms$. Defensive agents are installed on the set $R(5)$, i.e. routers $R6, R7$ and $R10$.

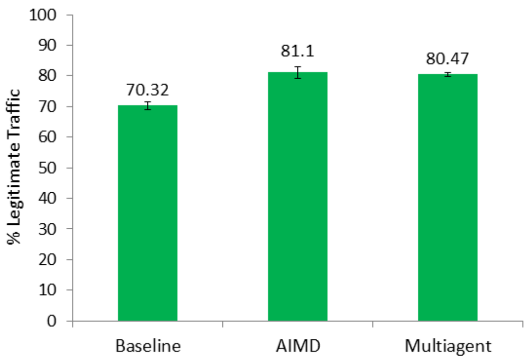
As in the previous experiment, we plot the global reward averaged over 20 repetitions (i.e. over 20 universal policies). Training results are presented in Figure 3.7 and it is clear that the system learns and improves over time until it converges.

As previously, for evaluation we randomly sample 50 episodes and we compare our proposed approach against the Baseline and the popular AIMD throttling (Yau et al. 2005) approaches. These approaches use a lower boundary $L_s = 6$ and the same upper boundary $U_s = 8$ as our approach.

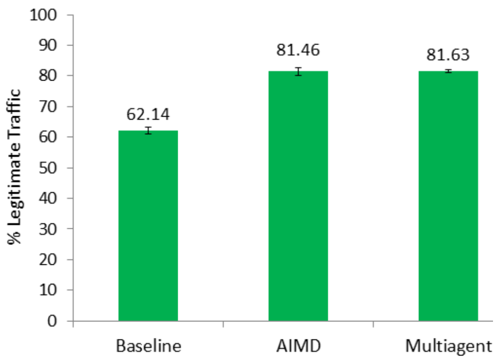
Performance is measured as the percentage of legitimate traffic that reached the server. Figures 3.8a - 3.8e show the average performance over the 50 episodes for the 20 policies learnt during the system training, for each of the attack rate dynamics respectively; error bars show the



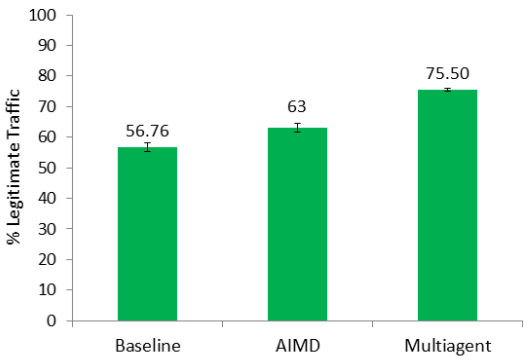
(a) Constant-rate



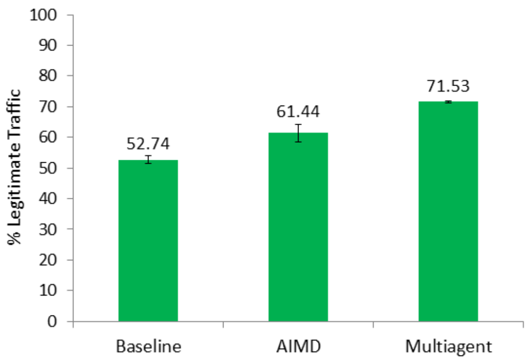
(b) Increasing-rate



(c) High Pulse



(d) Low Pulse



(e) Group

Figure 3.8: Evaluation Results for 3 RLs

standard error around the mean. Notice that for the group attack we retook a random sample of 50 episodes such that in each episode there are at least two attackers, one for each group. Experimental results reveal the following:

1. In all scenarios, MARL significantly outperforms the Baseline throttling approach. Similarly, AIMD outperforms the Baseline approach in all scenarios.
2. The MARL approach has the same performance as the AIMD approach in the scenarios involving the constant rate, increasing rate and high pulse attacks.
3. MARL significantly outperforms the AIMD approach in the scenarios involving the low pulse and group attacks.

Therefore, results show that our proposed approach significantly outperforms the Baseline, and either outperforms (in highly dynamic scenarios) or has the same performance as the AIMD approach. Furthermore, results show that Multiagent Router Throttling can cope with different patterns of attack rate dynamics that were not previously experienced in training.

3.4.2 Legitimate Traffic Distribution

This series of experiments aims at investigating how Multiagent Router Throttling behaves when legitimate traffic is not sent at constant rates. It has been shown that heavy-tail (or power-law) distributions can model local-area and wide-area network traffic (Park et al. 1996). A distribution is heavy-tail if:

$$P[X = x] \sim x^{-\alpha} \text{ as } x \to \infty \tag{3.2}$$

where $0 < \alpha < 2$. One of the most commonly used heavy-tail distributions is the Pareto distribution, where its probability density function is given by:

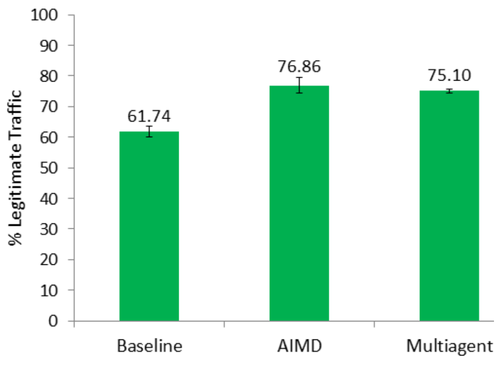
$$p(x) = \alpha \kappa^{\alpha} x^{-\alpha-1} \tag{3.3}$$

where $\alpha, \kappa > 0, x \geq \kappa$.

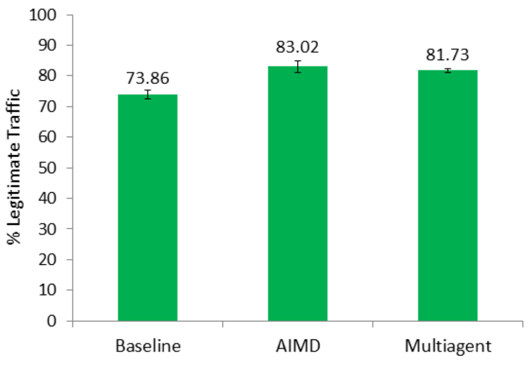
Specifically, the superposition of many Pareto traffic sources can be used to describe network traffic. The Pareto ON/OFF model (Park et al. 1996) has been proposed, where constant size packets are sent at fixed rates during ON periods, no packets are sent during OFF periods and the duration of both ON and OFF periods follows a Pareto distribution.

This series of experiments is based on the previous one i.e. the one discussed in Section 3.4.1.2. We use the topology with the three learning agents (Figure 3.1) and the agents use their universal policies learnt during offline training (Figure 3.7).

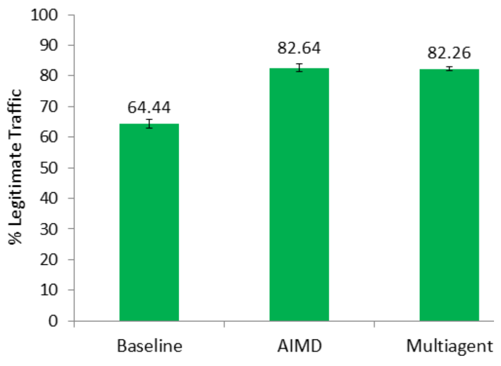
During evaluation the inter-arrival packet times of the legitimate users are described by the Pareto ON/OFF model described earlier. Notice that this was not the case during offline training. The average duration time for both ON and OFF periods are set to $500ms$ respectively. The



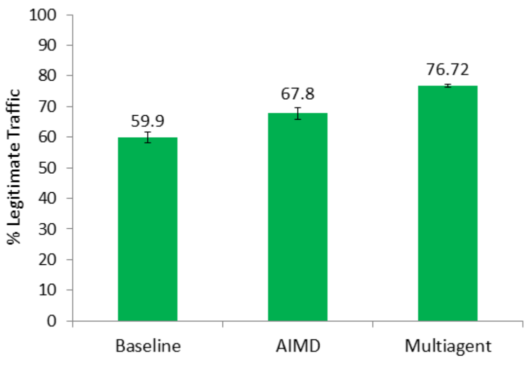
(a) Constant-rate



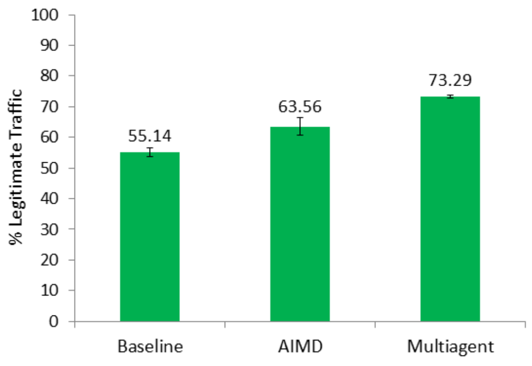
(b) Increasing-rate



(c) High Pulse



(d) Low Pulse



(e) Group

Figure 3.9: Evaluation Results Using the Pareto ON/OFF Model

shape parameter of the Pareto distributions is set to $\alpha = 1.5$. These are the ns-2 default values. Recall from Section 3.3 that the fixed rate of a legitimate user is randomly and uniformly drawn from the range $[0, 1] Mbit/s$, and also that the packet size for all legitimate users is $1000\ bytes$.

Figures 3.9a - 3.9e depict the average performance over the sample of 50 episodes for the 20 policies learnt during offline training, for the constant-rate, increasing-rate, high pulse, low pulse and group attacks respectively. Results show that Multiagent Router Throttling can cope with realistic legitimate rate distributions. We obtain similar results to experiments from Section 3.4.1.2 and reach to the same conclusion i.e. the MARL approach significantly outperforms the Baseline, and either outperforms or has the same performance as the AIMD approach.

3.5 Emulation Experiments

The major limitation of ns-2 is that is is very slow and therefore fails to scale-up. Another limitation is that modelling is complex and time consuming. As we move on, experiments will grow both in topology and size and number of learning agents.

For these reasons we have implemented a network emulator in C++ which serves as a testbed for demonstrating the effectiveness of our proposed approach. The emulator has been introduced in order to allow us to study our learning-based approach in large-scale scenarios. This is achieved by moving to a higher level of abstraction and omitting details that do not compromise a realistic evaluation. Specifically, the emulator treats the internal model of a network as a black box and mimics the observable behaviour of the network by only considering inputs and outputs of the model. It's important to note that this is adequate to demonstrate the functionality of throttling approaches.

3.5.1 Emulator Versus Simulator

In this series of experiments we repeat the experiments from section 3.4.1.2 using the abstract network emulator in order to examine whether previously obtained results still hold.

We plot the global reward at the last time step of each episode, averaged over 20 repetitions (i.e. over 20 universal policies). Training results are presented in Figure 3.10 and it is clear that the system learns and improves over time until it converges.

Figures 3.11a - 3.11e depict the average performance over the sample of 50 episodes for the 20 policies learnt during offline training, for the constant-rate, increasing-rate, high pulse, low pulse and group attacks respectively. We obtain similar results to experiments from Section 3.4.1.2 and reach to the same conclusion i.e. the MARL approach significantly outperforms the Baseline, and either outperforms or has the same performance as the AIMD approach.

3.6 Summary

The original throttling approach (Baseline and AIMD) by Yau et al. (2005) is victim-initiated, that is, the victim controls and sends the throttle signals to the upstream routers. However, it

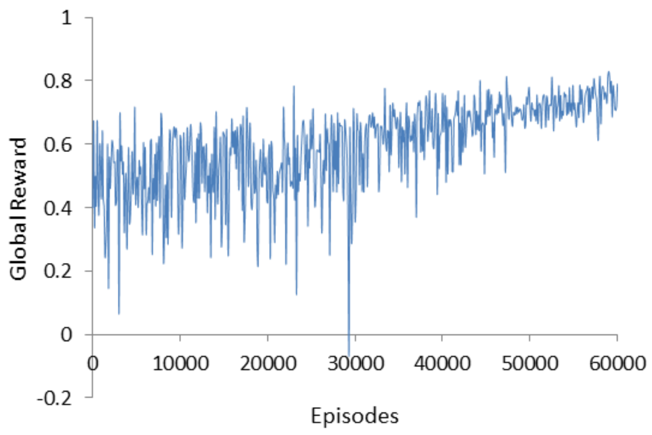


Figure 3.10: Training Results Using Emulation

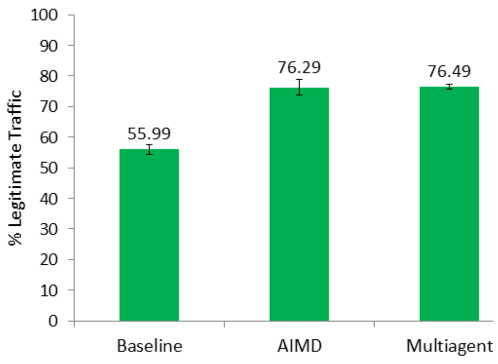
is based on the assumption that either the victim remains operational during a DDoS attack or that a helper machine is introduced to deal with the throttle signalling (Yau et al. 2001). The first assumption can be violated in a real-world scenario. As far as the second assumption is concerned, the helper machine can also become a target of the attack.

In essence, the problem may arise because the existing throttling approach is victim-initiated i.e. it has a single point of control. In other words, although it offers a distributed response, it is still a centralised approach. Our proposed approach has a decentralised architecture as it consists of multiple reinforcement learning agents that learn to rate-limit or throttle traffic towards the victim server. It provides a decentralised coordinated response to the DDoS problem, thus being resilient to the attacks themselves.

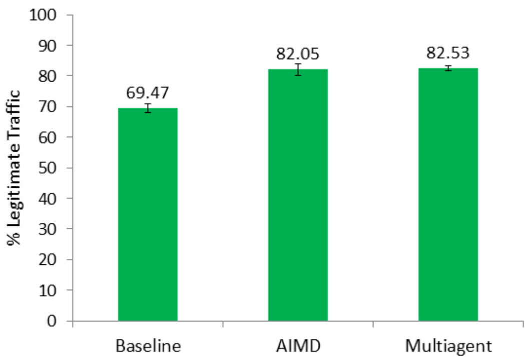
We have demonstrated that Multiagent Router Throttling significantly outperforms the Baseline, and either outperforms or performs the same as the AIMD approach in a series of increasingly sophisticated attack rate dynamics involving constant-rate, increasing-rate, pulse attacks and a combination of the aforementioned. The next chapter investigates these findings further and explains the observed behaviour.

Experiments conducted in this chapter involve small-scale topologies. Recall from the literature review that the basic form of multiagent reinforcement learning fails to scale-up to large and complex real-world domains. Further investigation is required to examine how the advantages of Multiagent Router Throttling can apply to realistic large-scale topologies. Scalability is the primary focus of the next chapter.

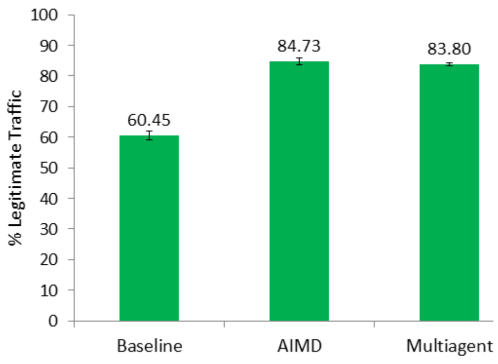
Finally, training our system in an offline manner requires to have a reasonable knowledge of the network model and topology. The same applies to the non-learning techniques (Baseline and AIMD) which require parameter tuning based on this knowledge. However, if these are inaccurate (i.e. they do not reflect the actual ones) or change in due course our approach would



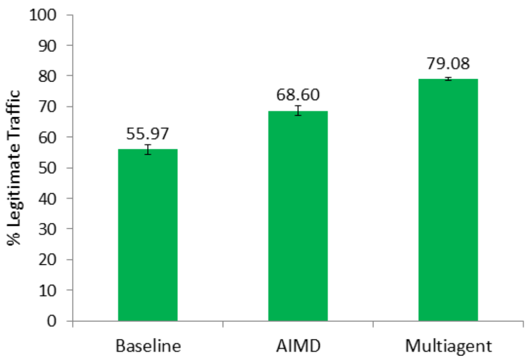
(a) Constant-rate



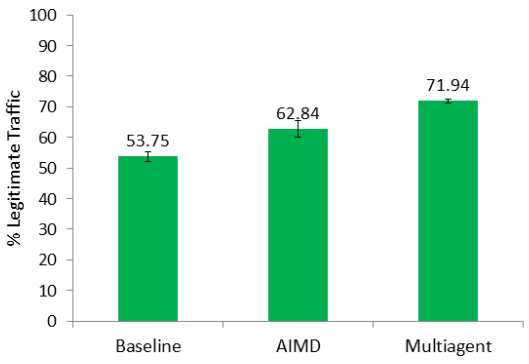
(b) Increasing-rate



(c) High Pulse



(d) Low Pulse



(e) Group

Figure 3.11: Evaluation Results Using Emulation

require re-training, and the non-learning approaches would require parameter re-tuning. Online learning is the focus of Chapter 5.

CHAPTER 4

Coordinated Team Learning for Scalability

In Chapter 3 we have introduced Multiagent Router Throttling, a novel approach to defend against DDoS attacks, where multiple reinforcement learning agents are installed on a set of routers and learn to rate-limit or throttle traffic towards a victim server. Multiagent Router Throttling provides a decentralised coordinated response to the DDoS threat. It has been demonstrated to perform well against DDoS attacks in small-scale network topologies but suffers from the "curse of dimensionality" when scaling-up to large topologies. The focus of this chapter is to tackle the scalability challenge.

Scalability is one of the most important aspects of a defence system since a non-scalable system will never be adopted for wide deployment by a company or organisation. In this chapter we propose Coordinated Team Learning (CTL) that resolves this challenge. CTL is a novel design to the original Multiagent Router Throttling approach based on the divide-and-conquer paradigm. CTL incorporates several mechanisms, namely, hierarchical team-based communication, task decomposition and team rewards and its scalability is successfully demonstrated in large scenarios.

The proposed CTL approach provides an automated and effective response against the highly complex and multi-dimensional DDoS threat. We evaluate CTL in a series of scenarios with increasingly sophisticated attack rate dynamics and show that it outperforms both a baseline and a popular state-of-the-art throttling technique. Furthermore, the network environment is highly dynamic and our approach is highly responsive to the attackers' dynamics thus providing flexible behaviours over frequent environmental changes.

4.1 Hierarchical Communication (Comm)

This chapter heavily extends the original design of Multiagent Router Throttling that was described in full detail in Section 3.2, and from now on we will refer to it as MARL. The focus of this chapter is also on offline learning.

As it is later demonstrated the basic MARL approach suffers from the "curse of dimensionality" and fails to scale-up in large scenarios. To scale-up we propose a number of mechanisms based on the divide-and-conquer paradigm. Generally, the divide-and-conquer paradigm breaks down a large problem into a number of sub-problems which are easier to be solved. The individual solutions to the sub-problems are then combined to provide a solution to the original problem.

The first step towards scalability is to form teams of agents as shown in Figure 4.1. Dashed lines do not necessarily represent nodes with a direct connection. The structure of a team is shown in Figure 4.2. Each team consists of its leader, an inner layer of intermediate routers, and the throttling routers which are k hops away from the server. Notice that the proposed defensive architecture constitutes an overlay network i.e. atop of the underlying network topology. Recall that the participating routers belong to the same administrative domain e.g. an ISP. In case of collaborative domains, each team can belong to a different administrative domain. Note that only the throttling routers are reinforcement learning agents. The rest of the routers are non-learning agents and we will explain their role shortly. The number of teams and their structure depend on the underlying topology and network model.

The second step towards scalability involves communication. Direct communication is defined as a purely communicative act in order to transmit information (i.e. a speech act) (Matarić 1998). Indirect communication is concerned with the observation of other agents' behaviour and its effects on the environment. Specifically, communication tackles the partial observability problem, where distributed agents cannot sense all of the relevant information necessary to complete a cooperative task.

The domain does not permit reliance on a complex communication scheme. Firstly, as the communicated information increases, the defensive system can introduce a higher delay in responding to the DDoS problem. Secondly, a designer would aim for minimal communication as different components of the defensive system may belong to different network administrative domains. Lastly, mechanisms to ensure a reliable and secure communication need to be introduced and these add an extra layer of complexity and can further increase the communication latency.

We propose a hierarchical uni-directional communication scheme. The victim's router signals its local load reading to the team leaders. The team leaders signal both their local load reading and the received reading from the victim's router to their intermediate routers. Similarly, the intermediate routers signal their local load reading and the two received readings to their throttling routers. This is depicted in Figures 4.1 and 4.2 by the uni-directional arrows.

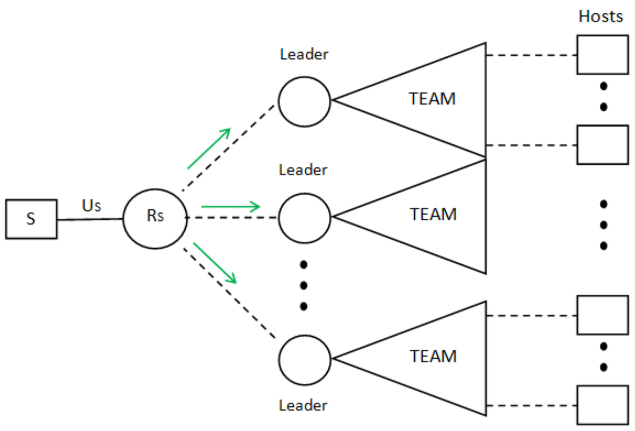


Figure 4.1: Team Formation

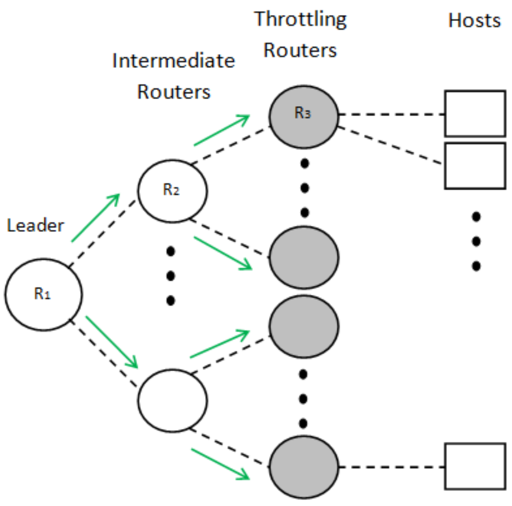


Figure 4.2: Team Structure

We should note that this constitutes an indirect communication scheme because the local load readings of the signallers, are essentially the effects of the throttling agents' actions. The state space of each reinforcement learning agent now consists of four features. Consider for example the router R_s and the routers R_1, R_2, R_3 in Figure 4.1 and 4.2 respectively. Assuming their local instantaneous traffic rates are r_s, r_1, r_2 and r_3 respectively, the state features of router R_3 are $< r_s, r_1, r_2, r_3 >$. The hierarchical communication method uses the same global reward function as the basic approach which is described in section 3.2.

4.2 Independent and Coordinated Team Learning (ITL & CTL)

The final step towards scalability is the use of task decomposition and team rewards. For task decomposition, it is now assumed that instead of having a big DDoS problem at the victim, there

Algorithm 6 Independent Team (IT) Reward Function

```
if $(loadRouter_{leader} > (U_s/\#teams))$ then
    // Punishment
    $r = -1$
else
    // Reward in $[0, 1]$
    $r = legitimateLoad_{leader}/legitimateLoad_{team}$
end if
```

Algorithm 7 Coordinated Team (CT) Reward Function

```
if $(loadRouter_{leader} > (U_s/\#teams))$ AND $(loadRouter_{server} > U_s)$ then
    // Punishment
    $r = -1$
else
    // Reward in $[0, 1]$
    $r = legitimateLoad_{leader}/legitimateLoad_{team}$
end if
```

are several smaller DDoS problems where the hypothetical victims are the team leaders[1]. The hypothetical upper boundary of each leader depends on its traffic sources (i.e. the amount of host machines). Assuming a defence system of homogeneous teams, with respect to their sources, the hypothetical upper boundary for each team leader is given by $U_s/\#teams$.

Moreover, agents are now provided with rewards at the team level rather than the global level, that is, agents belonging to the same team receive the same reward or punishment. In this section we propose the independent team learning and coordinated team learning approaches. Their reward functions are as follows:

Independent Team Reward: An agent within a team receives a punishment of -1 if the team's load exceeds its hypothetical upper boundary. It receives a reward of $L \in [0, 1]$, where L denotes the proportion of the legitimate traffic that reached the team leader (with respect to the total legitimate of the team). The independent team reward function of each reinforcement learning agent is shown in Algorithm 6.

Coordinated Team Reward: This approach involves coordination between the teams of agents by allowing a team's load to exceed its hypothetical upper boundary as long as the victim's router load remains below the global upper boundary. The coordinated team reward function of each reinforcement learning agent is shown in Algorithm 7.

Lastly, as it is later demonstrated the hierarchical communication functionality is beneficial

[1]An ISP backbone or core router, like a team leader or an intermediate router, is able to handle large amounts of traffic therefore it is unlikely to become a victim itself.

to the system, therefore both of the approaches include it.

4.3 Experimental Setup

Experiments are conducted using the network emulator we have developed. Experiments are conducted using the network emulator we have developed. Our experimental setup is based on work done by Yau et al. (2005). As a convention, bandwidth and traffic rates are measured in $Mbit/s$. The bottleneck link $S - R_s$ has a limited bandwidth of U_s, which constitutes the upper boundary for the server load. The rest of the links have an infinite bandwidth. Legitimate users and attackers are evenly distributed, specifically each host is independently chosen to be a legitimate user with probability p and an attacker with probability $q = 1 - p$. Parameters p and q are set to be 0.6 and 0.4 respectively. Legitimate users and attackers are chosen to send UDP traffic at constant rates, randomly and uniformly drawn from the range $[0, 1]$ and $[2.5, 6]$ $Mbit/s$ respectively. We refer to an *episode*, as an instance of the network model just described.

Reinforcement learning agents are installed on throttling routers. We discretise the continuous action space into ten actions: $0.0, 0.1, ..., 0.9$ which correspond to $0\%, 10\%..., 90\%$ traffic drop probabilities (recall from Section 3.2.3 that action 1.0 is prohibited). We use function approximation, specifically Tile Coding (Sutton & Barto 1998), for the representation of the continuous state space. Recall from Section 4.1 that the state space of each agent consists of four features; its local router load and three communicated router loads from downstream routers. The local router load of a router that sees a low, medium and high amount of traffic is split into 6, 12 and 18 tiles respectively; as a note, in practise, the network administrator is aware of the nodes that typically experience a substantial amount of traffic. In all cases, each of the three communicated router loads is split into 6 tiles. The number of tilings is 8 in all cases.

For the purposes of our experiments we use tree network topologies consisting of homogeneous teams of agents. Notice that our proposed approach is not restricted to tree network topologies; the proposed defensive architecture constitutes an overlay network i.e. atop of the underlying network topology. Therefore, parent-child relationships can still be obtained even if the underlying topology is not tree-structured.

Each team of agents contains two intermediate routers and six throttling routers (i.e. six reinforcement learning agents, three for each intermediate router). There are also 12 host machines corresponding to each team. A throttling router can have behind it a different number of host machines; we have varied this to be from 1 to 3. The upper boundary depends on the topology size and is set to be equal to $U_s = \#Hosts + 2$. For example, for the network topology consisting of 2 teams the upper boundary is given by $U_s = 24 + 2 = 26$.

Finally, the control parameters for the Baseline and AIMD throttling techniques are configured based on values or range of values recommended by their authors (Yau et al. 2005).

4.4 Offline Learning Experiments

Notice the two different phases for offline learning, namely, training and evaluation. During the offline training of our system we can keep track of and distinguish the legitimate traffic. However, we particularly emphasise that this is not the case during the evaluation of our system. Recall that the rationale behind this is that the defensive system can be trained in simulation, or in any other controlled environment (e.g. wide-area testbed, small-scale lab), where legitimate traffic is known a priori, and then deployed in a realistic network.

4.4.1 Performance

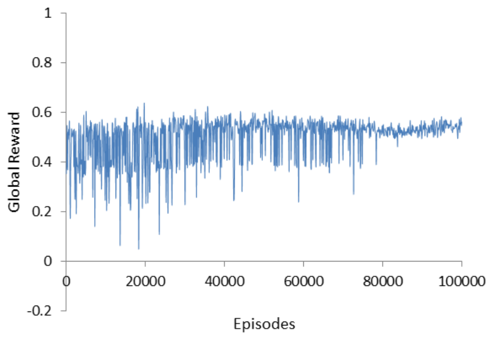
The first experiment of this section aims at learning a universal policy, that is, the "best" response for all possible instances of the network model using the topology consisting of 30 learning agents (5 teams). The system is trained for 100000 episodes; since this chapter investigates scalability and the number of agents will be increased, we leave the system to train for more episodes compared to the number of training episodes in the previous chapter. At the start of each episode a new network instance is generated i.e. we re-choose the legitimate users, attackers and their rates according to the model (described earlier in section 4.3). Attackers use the constant-rate attack pattern.

Our approach uses a linearly decreasing ϵ-greedy exploration strategy with an initial $\epsilon = 0.3$, and the learning rate is set to $\alpha = 0.05$; since the number of training episodes was increased to 100000, we use slightly lower values for both ϵ and α than those in the previous chapter. These parameters, despite not being optimised, are set in accordance to the reinforcement learning literature (Sutton & Barto 1998). As in the previous chapter, our goal is to investigate the new proposed CTL design and evaluate its performance and potential deployability. Parameter tuning is performed in Chapter 5 which deals with online learning.

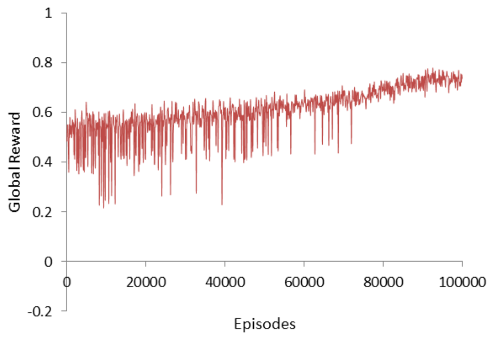
Exploration is stopped after the 80000th episode. Each episode runs for 1000 time steps and both legitimate and DDoS traffic starts at 0. The system training attempts to learn a universal policy for all network instances. MARL, Comm, ITL+Comm and CTL+Comm refer to the basic approach (described in section 3.2), hierarchical communication (described in section 4.1), independent and coordinated team learning (described in section 4.2) approaches respectively.

We plot the global reward at the last time step of each episode, averaged over ten repetitions (i.e. over ten universal policies). Training results for the four reinforcement learning-based approaches are presented in Figures 4.3a - 4.3d. It is clear that the system learns and improves over time until it finally converges. Because of the probabilistic nature of the environment, different rewards will be yielded in each different episode and hence the shape of the graphs.

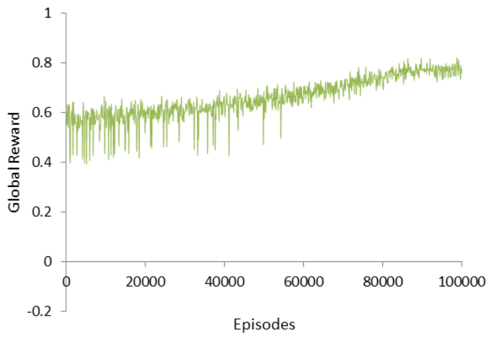
We evaluate our approach against the Baseline and the popular AIMD router throttling (Yau et al. 2005) approaches. Each reinforcement learning agent uses its policy learnt during the system training. For evaluation we randomly sample 100 episodes each of a duration of 60 time



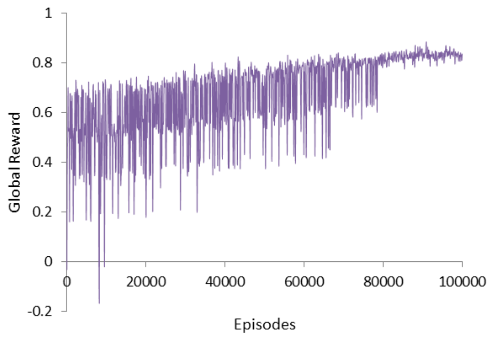
(a) MARL



(b) Comm



(c) ITL+Comm



(d) CTL+Comm

Figure 4.3: Offline Learning for 30 RLs

steps. Legitimate traffic is started at $t = 0$ and stopped at $t = 60$. Attack traffic lasts for 50 time steps; it is started at $t = 5$ and stopped at $t = 55$. All approaches use an upper boundary for the victim of $U_s = 62$ and the Baseline and AIMD also use a lower boundary of $L_s = 56$.

Evaluation is performed using the following five different patterns of attack dynamics of different sophistication levels. We emphasise that these patterns have not been previously seen by our system during the training period. The attack dynamics are described below (Mirkovic & Reiher 2004):

- **Constant-rate attack:** The maximum rate is achieved immediately when the attack is started.
- **Increasing-rate attack:** The maximum rate is achieved gradually over 25 time steps.
- **Pulse attack:** The attack rate oscillates between the maximum rate and zero. The duration of the active and inactive period is the same and represented by D. We create two different attacks namely the high and low pulse attacks which have a period of $D = 5$ and $D = 2$ time steps respectively.

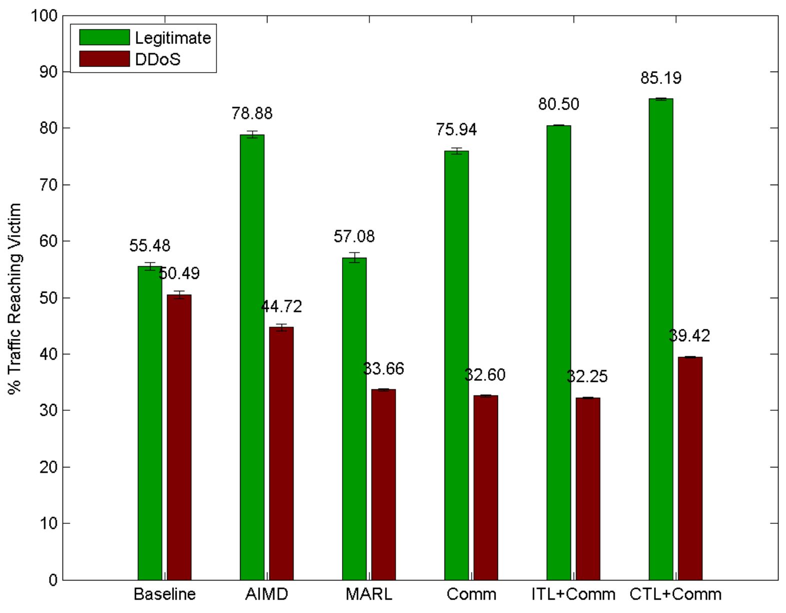


Figure 4.4: Performance for Constant-rate Attack

- **Group attack:** Attackers are split into two groups and each group performs simultaneously a different attack pattern. We choose the first group to perform a constant-rate and the second to perform a low pulse attack.

Figures 4.4 - 4.8 show the average performance over the 100 episodes for the 10 policies learnt during the system training, for the five types of attack rate dynamics respectively; error bars show the standard error around the mean. Performance is measured as the percentage of the legitimate traffic that reached the victim throughout an episode; the higher the value on the graph the better. For completeness, the figures also show the percentage of the DDoS traffic (note that the problem is caused by the attack volume and not the traffic content).

As expected, the AIMD approach outperforms the Baseline approach in all five scenarios. Furthermore, the basic MARL approach fails to perform well in this large-scale domain. The Comm approach offers great benefit to the system's performance over the basic MARL approach. Recall that the goal of an agent in offline learning is to obtain a universal policy, that is, the "best" response for all possible situations the network might be found in. Hierarchical communication helps each agent to distinguish between the different situations and therefore to learn a better policy for similar ones.

ITL+Comm and CTL+Comm further improve the system performance as they use task decomposition and team rewards. As expected, CTL+Comm performs better since it allows a team

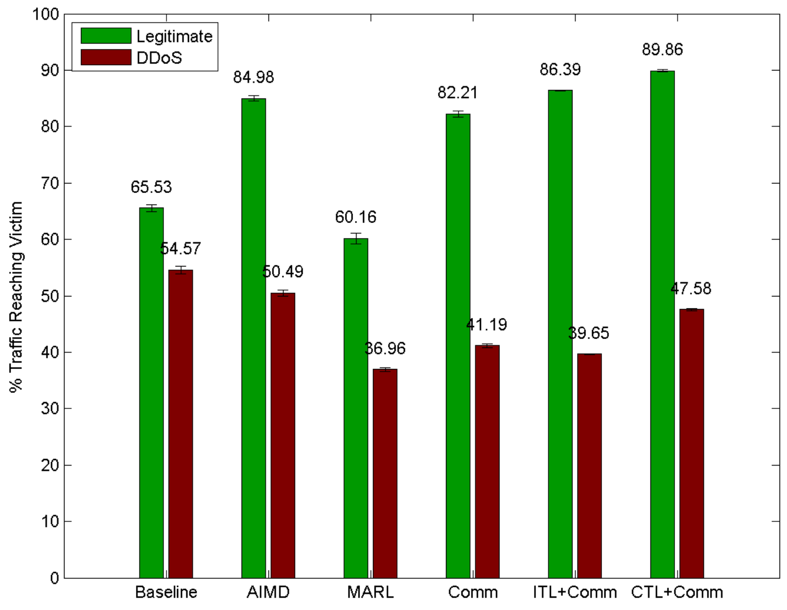


Figure 4.5: Performance for Increasing-rate Attack

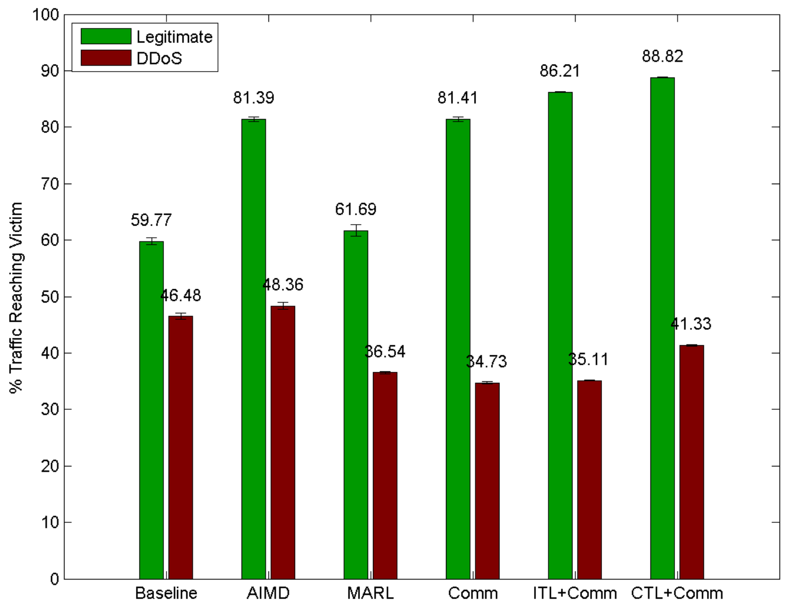


Figure 4.6: Performance for High Pulse Attack

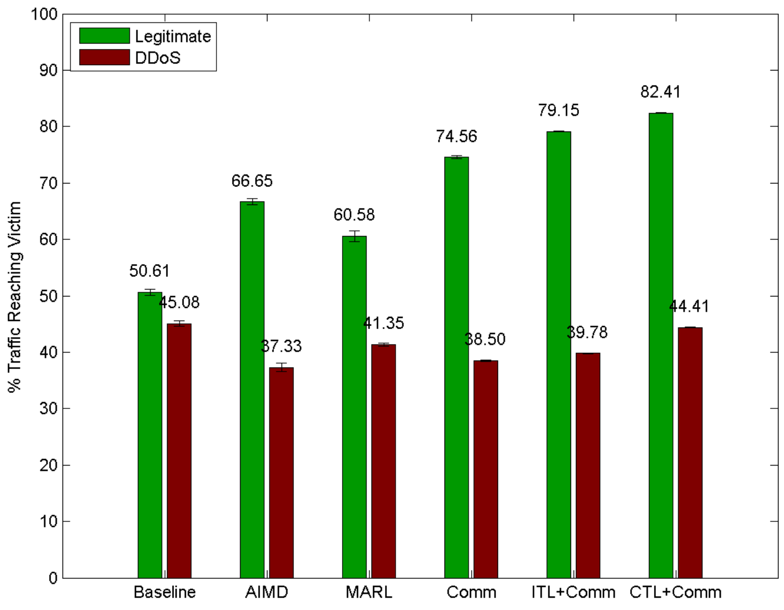


Figure 4.7: Performance for Low Pulse attack

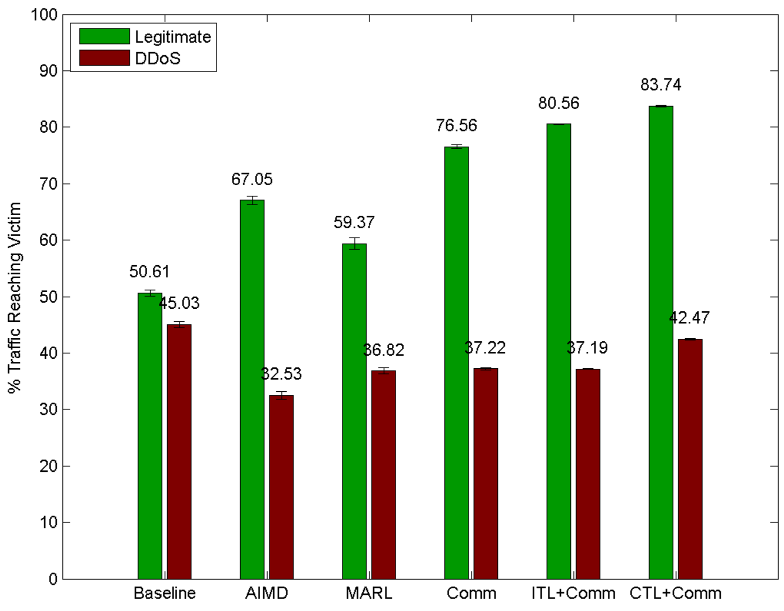


Figure 4.8: Performance for Group Attack

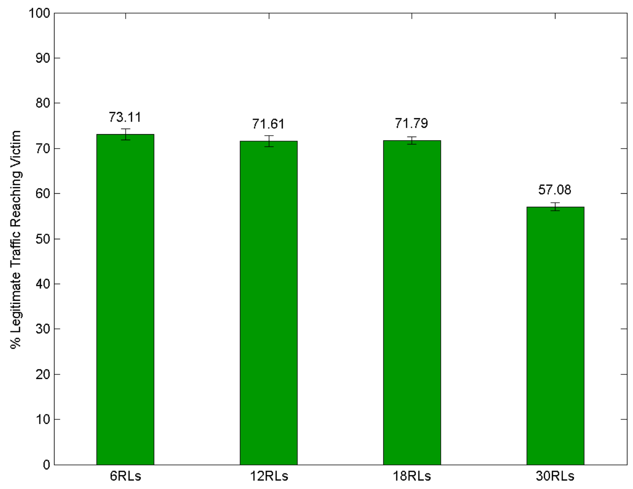


Figure 4.9: Scalability of MARL

leader's load to exceed its hypothetical boundary as long as the victim router's load is below its limit.

Most importantly, CTL+Comm outperforms the Baseline, AIMD and all other learning-based approaches in all scenarios. Specifically, it outperforms the Baseline and AIMD approaches about 24-33% and 5-17% respectively. The AIMD approach suffers the most in the highly dynamic scenarios of high pulse, low pulse and group attacks.

CTL+Comm outperforms AIMD for the following two reasons. Firstly because CTL+Comm learns a better behaviour during the training phase, and secondly because our proposed approach is more adaptable and responsive to the attackers' dynamics. These are examined in section 4.4.3.

4.4.2 Scalability

This series of experiments aims at examining the scalability of each individual learning-based approach. Experiments are conducted for the constant-rate attack scenario for up to 30 reinforcement learning agents. As previously explained, each agent is first trained offline to obtain a universal policy that it will later use for evaluation.

Figures 4.9 - 4.12 show the scalability results for the MARL, Comm, ITL+Comm and CTL+Comm approaches respectively. It is shown that the performance of MARL remains unaffected for up to 18 agents but severely declines in the case of 30 learning agents. The performance of Comm improves from 6 to 12 agents but then it starts declining when moving to the cases of 18 and 30 agents, although the performance drop is not as severe as in the case of MARL.

Lastly, it is demonstrated that the performance of both ITL+Comm and CTL+Comm remains unaffected by the addition of new teams of learning agents. This is a strong result suggesting that

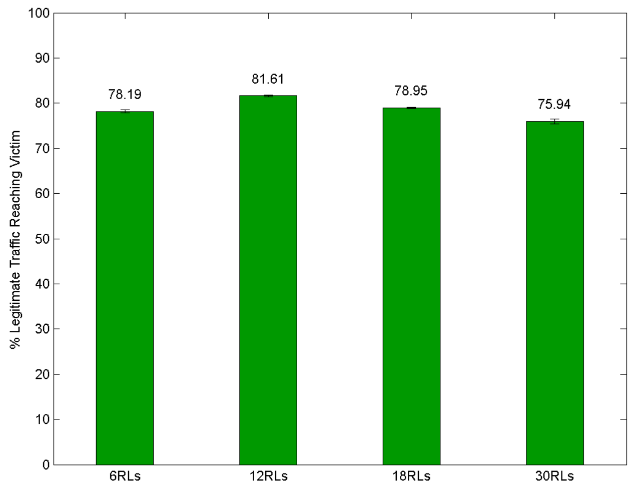


Figure 4.10: Scalability of Comm

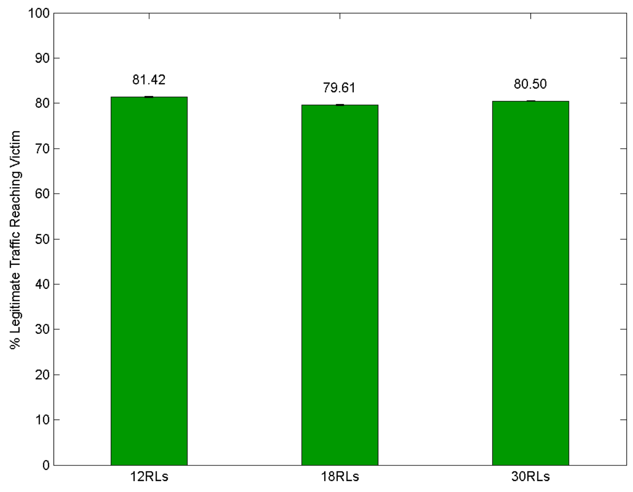


Figure 4.11: Scalability of ITL+Comm

the two approaches are capable of scaling to large network topologies.

4.4.3 Aggregate Load Convergence

This series of experiments aims at shedding light on the previous experimental results from Section 4.4.1 by investigating how the aggregate load behaves during a DDoS attack. Recall from Section 2.3.5.2 that the original Router Throttling approach (Baseline, AIMD (Yau et al. 2005)) requires the aggregate load to converge within the lower and upper boundaries i.e. $r_s \in [L_s, U_s]$.

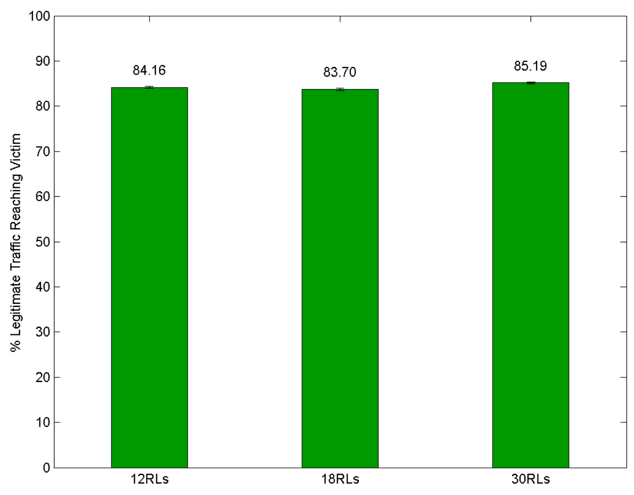


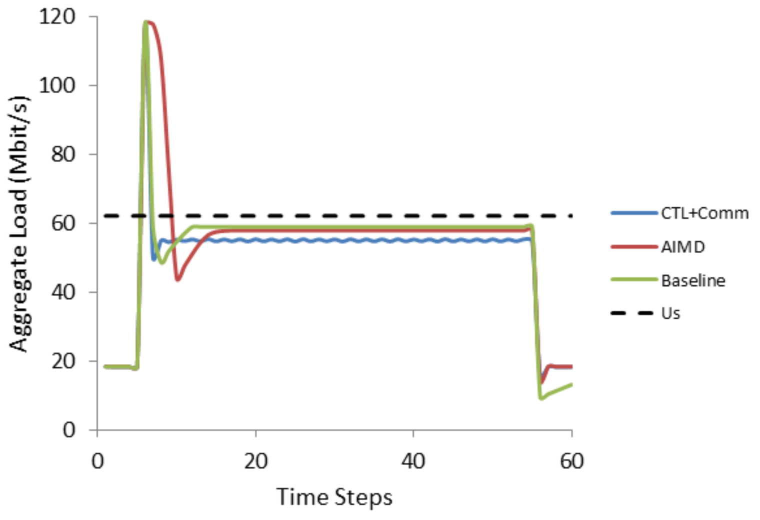
Figure 4.12: Scalability of CTL+Comm

Also, recall from Section 3.1 that Multiagent Router Throttling is required to bring the aggregate load below the upper boundary i.e. $r_s \leq U_s$. To better examine this we consider the scenarios of constant-rate, increasing-rate and high pulse attacks. At this point we are only interested in the CTL+Comm approach since it can better scale-up (Section 4.4.2) and significantly outperforms the other reinforcement learning-based approaches (Section 4.4.1).

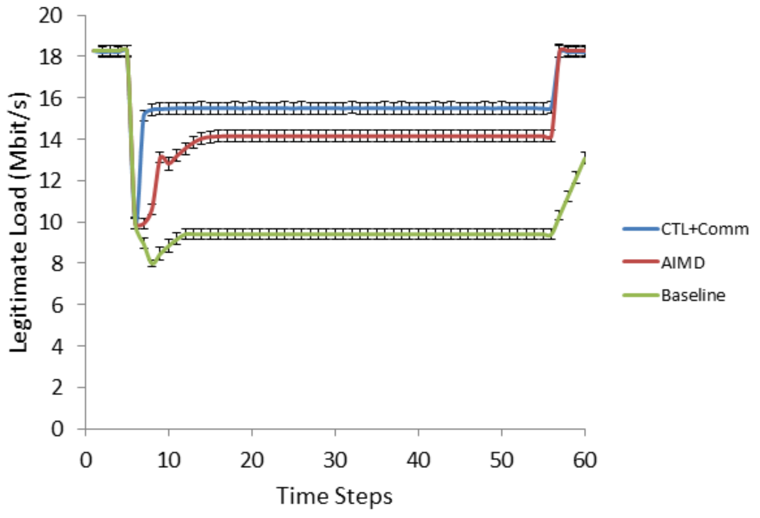
Figure 4.13a shows how the aggregate load varies for the constant-rate attack scenario when the Baseline, AIMD and CTL+Comm approaches are used. There exist 30 reinforcement learning agents that use their universal policies learnt during offline training in section 4.4.1, and values are averaged over the same 100 episodes used earlier for evaluation purposes in Section 4.4.1. The upper boundary for the victim is $U_s = 62$; the Baseline and AIMD approaches also use a lower boundary of $L_s = 56$. All three approaches do what they are intended to do, that is, to bring down the aggregate load to acceptable levels i.e. $r_s \leq 62$ and $r_s \in [56, 62]$ for the CTL+Comm and the non-learning approaches respectively.

Figure 4.13b shows how the legitimate load varies for the constant-rate attack scenario. This is averaged over the 100 episodes and error bars are plotted which show the standard error around the mean; the higher the value on the graph the better. The plots verify the previous results from Section 4.4.1 i.e. that CTL+Comm outperforms AIMD.

These results shed light on why this occurs. Our proposed approach outperforms the existing throttling approach for two reasons. Firstly, the system behaviour learnt during offline training is better than the hard-wired AIMD algorithm. Secondly, the existing non-learning approaches require a number of oscillations to bring the victim load to desirable levels; this will become more apparent as we continue to more dynamic attack scenarios. In contrast, our pro-



(a) Aggregate Load

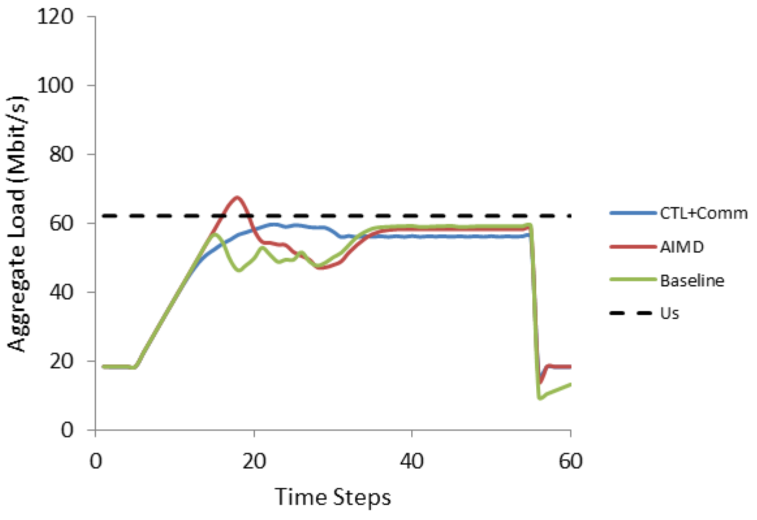


(b) Legitimate Load

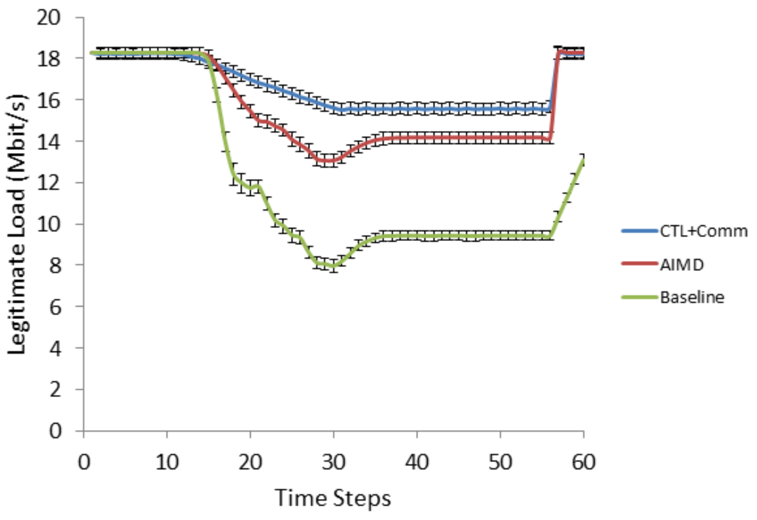
Figure 4.13: Load for Constant-rate Attack

posed CTL+Comm is highly responsive to the attackers' dynamics since the system learns the router throttles during offline training and as a result it does not require any system oscillations.

We repeat the same experiments with the increasing-rate DDoS attack scenario where similar results are obtained. Figures 4.14a and 4.14b show how the aggregate and legitimate load varies respectively for the increasing-rate attack scenario when the Baseline, AIMD and CTL+Comm approaches are used, averaged over the 100 episodes. Results show that all three approaches



(a) Aggregate Load

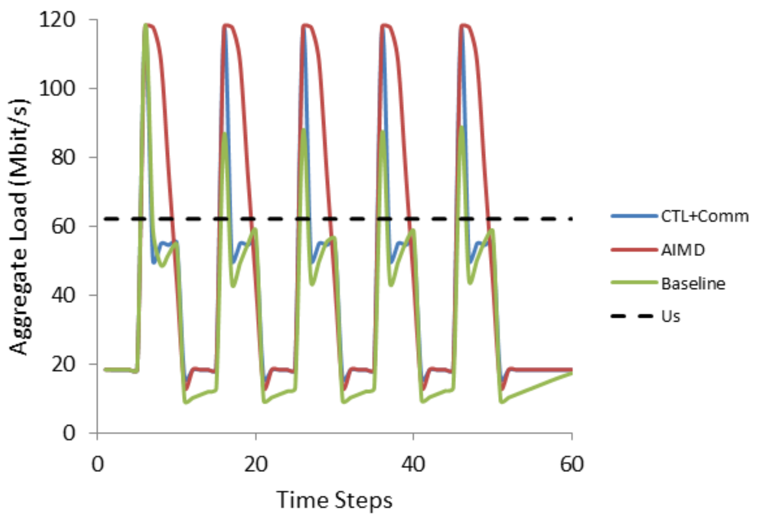


(b) Legitimate Load

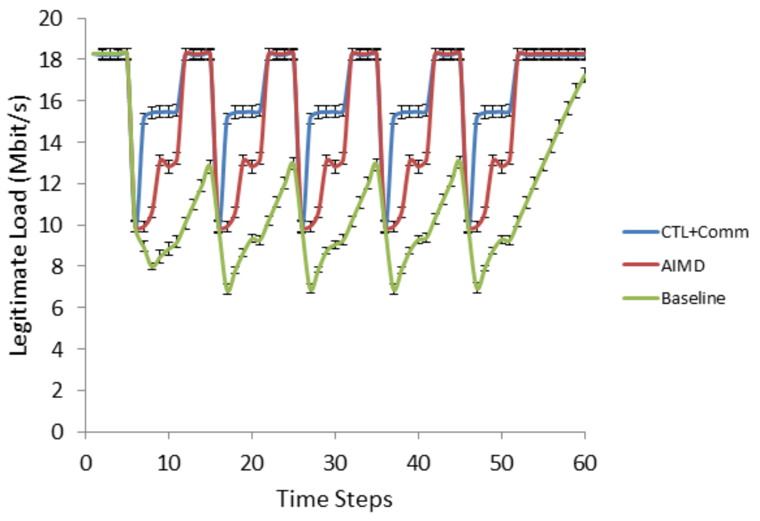
Figure 4.14: Load for Increasing-rate Attack

bring down the victim router's load to acceptable levels but our proposed approach outperforms the existing throttling approaches. As previously, this occurs because CTL+Comm has learnt a better behaviour during offline learning and also because it is highly responsive to environmental changes.

Lastly, we repeat the experiments with the highly dynamic scenario of high pulse DDoS attack. Figures 4.15a and 4.15b show how the aggregate and legitimate load respectively varies for



(a) Aggregate Load



(b) Legitimate Load

Figure 4.15: Load for High Pulse Attack

the high pulse attack scenario when the Baseline, AIMD and CTL+Comm approaches are used, averaged over the 100 episodes. It is evident that the AIMD approach requires considerably more time to bring the aggregate load within the desired boundaries, while the other two approaches do so much quicker. Also, our reinforcement learning-based approach allows more legitimate traffic than the existing approaches.

4.5 Summary

We have shown that our proposed CTL+Comm approach can significantly scale-up to large network topologies. Specifically, it has been demonstrated that its performance remains unaffected by the addition of new teams of learning agents. This is demonstrated for offline learning in topologies involving up to 30 reinforcement agents (Section 4.4.2). Note that the proposed approach can be useful in other related multiagent domains for example congestion problems such as air and road traffic management.

The CTL+Comm approach significantly outperforms both the Baseline and AIMD throttling approaches in a series of increasingly sophisticated attack dynamics involving constant-rate, increasing-rate, pulse attacks and a combination of the aforementioned as demonstrated in Section 4.4.1. The learnt decentralised behaviour performs better than the AIMD's centralised and hard-wired behaviour allowing more legitimate traffic to reach the victim server during a DDoS attack.

The original throttling approach (Baseline, AIMD) can suffer from stability problems because of system oscillations in order to settle the aggregate load to a desired level within the lower L_s and upper U_s boundaries. Performing throttling becomes challenging as the range $[L_s, U_s]$ becomes smaller. Even if the system does not suffer from stability problems, it still requires a number of oscillations which can cause an increase in the time required for the aggregate load to settle within the desired range. In contrast, our proposed CTL+Comm is highly responsive to the attackers' dynamics since the system learns the router throttles during offline training and as a result it does not require any system oscillations as demonstrated in Section 4.4.3.

Finally, training our system in an offline manner requires us to have a reasonable knowledge of the network model and topology. The same applies to the non-learning approaches (Baseline and AIMD) approaches which require parameter tuning based on this knowledge. However, if these are inaccurate (i.e. they do not reflect the actual ones) or change in due course our approach would require re-training, and the non-learning approaches would require parameter re-tuning. Online learning is the main focus of the next chapter.

CHAPTER 5

Incorporating Difference Rewards for Online Learning

In Chapter 4 we have proposed the Coordinated Team Learning (CTL) approach and demonstrated its scalability in offline learning experiments. The focus of this chapter is on online learning. Unlike non-learning approaches, one of the core advantages of reinforcement learning is its capability for online learning. Online learning relaxes the assumption of prior knowledge availability required for offline learning, and also allows a system to adapt to new situations.

The contribution of this chapter is the incorporation of a form of reward shaping called difference rewards to enable online learning. We show that the learning speed of our system is significantly improved and we further demonstrate its scalability in experiments involving up to 1000 reinforcement learning agents. The significant improvements on learning speed and scalability lay the foundations for a potential real-world deployment.

Furthermore, we show that the incorporation of difference rewards not only improves the learning speed and scalability, but the system performance is also significantly improved. We compare our approach against a baseline and a popular state-of-the-art router throttling techniques from the network security literature, and we show that our proposed approach outperforms them.

Moreover, we demonstrate that the proposed system meets the real-world requirements of robustness to agent failures and measurement noise.

Lastly, to bridge the gap between offline and online learning we demonstrate how behaviours learnt during offline learning can be ”injected” into our system in order to facilitate the online learning process.

5.1 Motivation for Online Learning

Let us first discuss the importance of online learning. Firstly, training our system in an offline manner requires the defender to have a reasonable knowledge of the network model and topology. The same applies to the non-learning approaches (e.g. Baseline and AIMD) approaches which require parameter tuning based on this knowledge. However, if these are inaccurate (i.e. they do not reflect the actual ones) or change in due course our approach would require re-training, and the non-learning approaches would require parameter re-tuning. Therefore, online learning relaxes the assumption of prior knowledge availability required for offline learning.

Secondly, learning a universal policy i.e. a policy that does well in every possible situation that the network might be found in, is time consuming. For example, in the experiments conducted in Chapter 4 which involved 30 reinforcement learning agents, the system was trained offline for 100000 episodes in order for each agent to be able to learn a universal policy.

Thirdly, learning a universal policy hinders scalability. Responding to DDoS attacks involves the coordination of throttling agents. Achieving such a coordination for every possible situation that the network might be found is already difficult even in small-scale scenarios. The goal of online learning is to learn the best response for only the particular situation the network is currently found in. For this reason, it is expected that scalability will be further improved.

Lastly, online learning can create an adaptive response to the DDoS problem as not only do attackers change their strategy, but legitimate users can also change their behaviour. Furthermore, an adaptive approach can create a robust throttling mechanism that enable agents in the system to recover from unpredictable situations such as router failures.

5.2 Difference Rewards

Difference rewards (Wolpert & Tumer 2000) were introduced to tackle the multiagent credit assignment problem encountered in reinforcement learning. Typically, in a multiagent system an agent is provided with a reward at the global level. This reward signal is noisy due to the other agents acting in the environment. This is because an agent may be rewarded for taking a bad action, or punished for taking a good action. The reward signal should reward an agent depending on its individual contribution to the system objective. The difference rewards method was previously known under the term "collective intelligence" or COIN.

Difference rewards[1] D_i is a shaped reward signal that helps an agent i learn the consequences of its actions on the system objective by removing a large amount of the noise created by the actions of other agents active in the system (Wolpert & Tumer 2000). It is defined as:

$$D_i(z) = R(z) - R(z_{-i}) \tag{5.1}$$

[1]The method "difference rewards" is also known under different names, for instance, when used with evolutionary computation approaches it is referred to as "difference evaluation functions".

where z is a general term representative of either states or state-action pairs depending on the application, $R(z)$ is the reward function used, and $R(z_{-i})$ is $R(z)$ for a theoretical system without the contribution of agent i.

As mentioned, in a multiagent system the reward is typically provided at the global level. Therefore, in this case the difference rewards signal D_i is given by:

$$D_i(z) = G(z) - G(z_{-i}) \tag{5.2}$$

where $G(z)$ is the global system performance, and $G(z_{-i})$ is $G(z)$ for a theoretical system without the contribution of agent i.

Difference rewards exhibit the following two properties. Firstly, any action taken that increases D_i simultaneously increases G. Secondly, since difference rewards only depend on the actions of agent i, D_i provides a cleaner signal with reduced noise that is created by the other agents acting in the system. These properties allow for the difference rewards to significantly boost learning speed and performance in a multiagent system. For these reasons the method has been successfully demonstrated (Tumer & Agogino 2007; Colby & Tumer 2013) to be capable of online learning.

The challenge for deriving the difference rewards signal is obviously how to calculate the second term of the equation $G(z_{-i})$ which is called the *counterfactual*[2]. From the literature, there are typically three ways to calculate the counterfactual.

In particular domains, the counterfactual $G(z_{-i})$ is possible to be directly calculated. In other cases, difference rewards can be calculated by introducing the term c_i, which represents a constant action of agent i, which is (theoretically) executed in order to eliminate the contribution of agent i and evaluate the system performance without it. The difference rewards D_i signal is then given by:

$$D_i(z) = G(z) - G(z_{-i} + c_i) \tag{5.3}$$

Alternatively, in cases where this is not possible, it has been demonstrated that difference rewards can be estimated. One way to do that is by using the Expected Difference Rewards (Colby & Tumer 2013):

$$ED_i(z) = G(z) - E_i(a)[G_z] \tag{5.4}$$

where $E_i(a)[G_z]$ is the expected value of the global reward over all actions that agent i may take. For a discrete action space it is given by:

$$ED_i(z) = G(z) - \sum_{a \in A} P_i(a) G_i(z_a) \tag{5.5}$$

where $P_i(a)$ is the probability of executing action a by agent i, and $G_i(z_a)$ is the global reward

[2]Oxford Dictionaries define the counterfactual as “relating to or expressing what has not happened or is not the case”.

Algorithm 8 Global (G) Reward Function

```
if $loadRouter_{server} > U_s$ then
  // Punishment in $[-1, -0.5)$
  if $loadRouter_{server} < 2U_s$ then
    $r = -(loadRouter_{server}/2U_s)$
  else
    $r = -1$
  end if
else
  // Reward in $[0, 1]$
  $r = legitimateLoad_{server}/legitimateLoad_{total}$
end if
```

when action a is executed by agent i.

Difference rewards have been applied in different domains such as air traffic management (Tumer & Agogino 2007) and distributed sensor networks (HolmesParker et al. 2013; Colby & Tumer 2013).

5.3 Incorporating Difference Rewards

5.3.1 Basic Design (D_MARL)

Recall from Section 3.2 that one of the goals of our system, which is encoded in the global reward function, is to keep the victim server operational during a DDoS attack. Specifically, when the server load is above the upper boundary the agents were receiving a punishment of $r = -1$. Difference rewards however have been empirically demonstrated to work better if the original reward function has a gradient throughout (Agogino & Tumer 2004). For this reason, in this chapter, punishment is provided "smoothly" within the range $[-1, 0)$. The new Global (G) reward function is given in Algorithm 8. We will later show how the approach behaves when it uses the reward function with and without the gradient throughout.

Let us now discuss the availability of L in the online learning setting i.e. when the system is trained directly in a realistic network. If the detection mechanism is accurate enough to derive attack signatures then the problem can be simply solved by filtering the attack traffic. However, we are interested in cases where the detection mechanism cannot precisely characterise the attack traffic i.e. when attack signatures cannot be derived. Inevitably, in such cases L can only be estimated.

There are different ways to measure legitimate traffic, for example by observing the behaviour or habits of customers and regular users or visitors of the victim's services and detect deviations. Another example is by observing the IP addresses that have been seen before; work conducted by Jung et al. (2002) reveals that during a DDoS attack to a website most sent requests were generated by IP addresses that did not appear before. For example for the CodeRed worm (Moore et al. 2002) only 0.6-14% of the IP addresses appeared before. Another way is by observing whether IP packets have appeared before or after the DDoS impact, as the latter suggests that they are likely illegitimate (Hussain et al. 2003).

The difference rewards signal D_i for any agent i requires the calculation of the global G reward without the contribution of agent i. To calculate the difference rewards we need to introduce the term c_i, which represents a constant action of agent i which is taken to eliminate its contribution to the system. The constant action c_i is later determined from the experimental results. Difference rewards are then calculated using the equation 5.3. We will refer to the combination of MARL with difference rewards as D_MARL.

5.3.2 Coordinated Team Learning (D_CTL)

At this point recall that the focus of previous chapters was on offline learning. The goal of an agent in offline learning is to learn a universal policy, that is, the "best" response for all possible situations the network might be found in. For this reason we have proposed in Section 4.1 the Hierarchical Communication (Comm) approach in order to help each agent to distinguish between the different situations and therefore to learn a better policy for similar ones.

However, the focus of this chapter is on online learning where the goal of an agent is to learn the best response for the particular situation the network is currently found in. Further considering that the goal of incorporating difference rewards is to provide a cleaner reward signal to the learning agents, it is expected that the benefit of hierarchical communication will not be as great as it was in the offline learning setting. Indeed, this is later verified by the experimental results.

Therefore, in this chapter we remove the hierarchical communication functionality and the new simplified architecture of our defensive system is described as follows. Teams of agents are formed as shown in Figure 5.1. Dashed lines do not necessarily represent nodes with a direct connection. In other words, the proposed defensive architecture constitutes an overlay hierarchy i.e. atop of the underlying network topology. Teams can either be homogeneous or heterogeneous. The structure of the team is shown in Figure 5.2. Each team consists of its leader and the throttling routers which are k hops away from the server. Participating routers belong to the same administrative domain e.g. an ISP. In case of collaborative domains, each team can belong to a different administrative domain. The number of teams and their structure depend on the underlying topology and network model.

The proposed architecture has the following advantages over the previous one. The system now requires a fewer number of routers to participate in the defence, as the intermediate routers (Section 4.1) are no longer part of it. Furthermore, the system has now simplified responsibilities as signalling agents are no longer installed on participating routers. The learning process is also facilitated as the state space of a reinforcement learning agent is now reduced from four state features to one i.e. containing only the local router load.

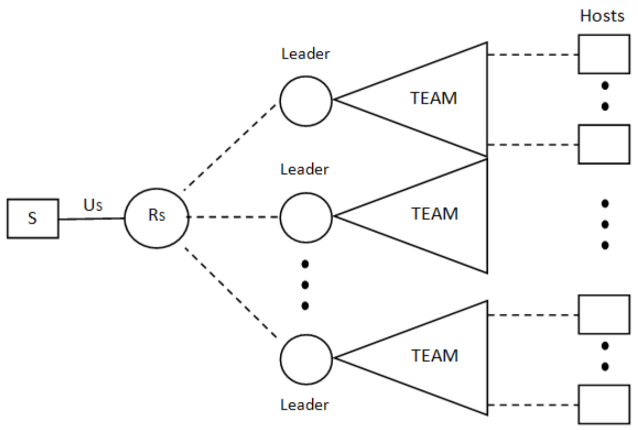


Figure 5.1: Team Formation

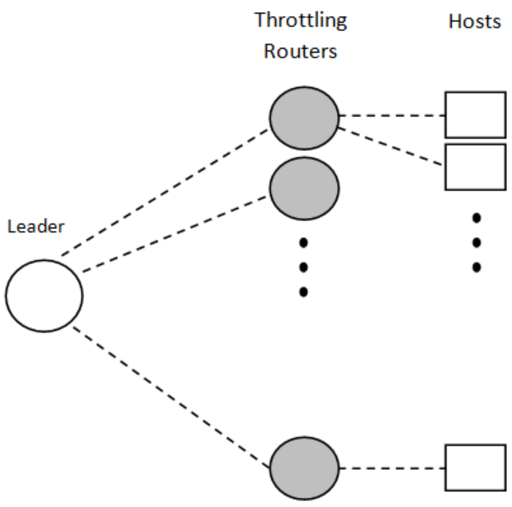


Figure 5.2: Team Structure

For scalability we use task decomposition and team rewards. The modified Coordinated Team (CT) reward function providing "smooth" rewards is given in Algorithm 9. The difference rewards D_{ji} signal for any agent i belonging to team j requires the calculation of the team CT_j reward without the contribution of agent i. To calculate the counterfactual we use the same c_i as defined earlier. Difference rewards can now be calculated using the formula below. We will refer to the combination of CTL with difference rewards as D_CTL.

$$D_{ji}(z) = CT_j(z) - CT_j(z_{-i} + c_i) \tag{5.6}$$

We assume that each learning agent calculates its own difference rewards signal. To achieve that, the victim and team leaders need to communicate to the learning agents the relevant in-

Algorithm 9 Coordinated Team (CT) Reward Function

```
if ($loadRouter_{leader} > (U_s/\#teams)$) AND ($loadRouter_{server} > U_s$) then
    // Punishment in $[-1, -0.5)$
    if ($loadRouter_{server} < 2U_s$) then
        $r = -(loadRouter_{server}/2U_s)$
    else
        $r = -1$
    end if
else
    // Reward in $[0, 1]$
    $r = legitimateLoad_{leader}/legitimateLoad_{team}$
end if
```

formation at each time step; from Algorithm 9 this information is the victim's aggregate load ($loadRouter_{server}$), leader's aggregate load ($loadRouter_{leader}$) and leader's estimate of legitimate load ($legitimateLoad_{leader}$) - see Section 5.7.1 for a walkthrough example. It is emphasised that communication is performed in a hierarchical team-based fashion and no information exchange between the learning agents takes place (i.e. decentralised coordination). This constitutes a simple and minimal communication scheme and the introduced overhead is not restrictive for a real-world deployment. To eliminate the communication overhead completely, an approximation of difference rewards may be possible, but this is subject to future work.

Lastly, the throttling agents start learning i.e. responding to an attack only when an attack is detected, that is, when the victim experiences congestion/overloading. In practise, it is expected that the victim will send an activation or "wake-up" signal to the defensive system, informing it about the need for protection. This reduces the computational complexity of our approach since the amount of information stored at a throttling router is linear in the number of servers under attack (as opposed to offline learning where all potential victims should be considered and thus being infeasible). Section 5.6 provides a discussion on the computational complexity and communication overhead of our proposed approach.

5.4 Parameter Tuning

Our experimental setup is based on the one used described in Section 4.3. In this section we present some preliminary results, and then describe a series of experiments on parameter tuning. Contrary to the offline learning setting where the quality of the learnt behaviour is the main objective, for online learning the learning speed is equally critical. We first examine the reward functions with gradient throughout. We then investigate different default actions for calculating the counterfactual. Then, we examine the effects of hierarchical communication and of exploration strategy on online learning.

5.4.1 Preliminary Results Without Parameter Tuning

The first experiment aims at investigating how the MARL, Comm, ITL+Comm and CTL+Comm approaches used for offline learning in Chapter 4, behave in the online learning setting prior any parameter tuning. We also compare them against the baseline and the popular AIMD throttling approaches (Yau et al. 2005). Figure 5.3a shows how all approaches compare to each other in terms of the percentage of legitimate traffic that reaches the server, in the topology involving 30 learning agents (5 teams) for the constant-rate attack scenario.

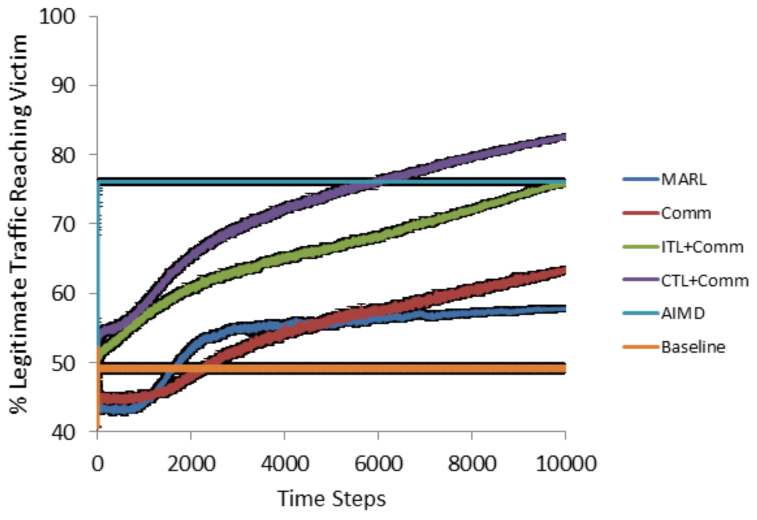
The error bars are very small and therefore hardly visible on the graph. For this reason, we present a close-up look of the same figure in Figure 5.3b to show the presence of error bars. Therefore, when looking back to Figure 5.3a, the presence of error bars is shown by the thin black "borders" on top and bottom of each of the plots. From now on, this applies to all plots which include error bars.

Reinforcement learning agents use an initial $\epsilon = 0.2$ and a decrease rate $\delta = 0.00002$. We initialise ϵ with a lower value than the initial values used for learning universal policies in the previous chapters because as mentioned, the goal of online learning is to learn the best response for only the particular situation the network is currently found in. Therefore, we suspect that a lower exploration rate may facilitate this task. The effect of exploration is examined in Section 5.4.5.

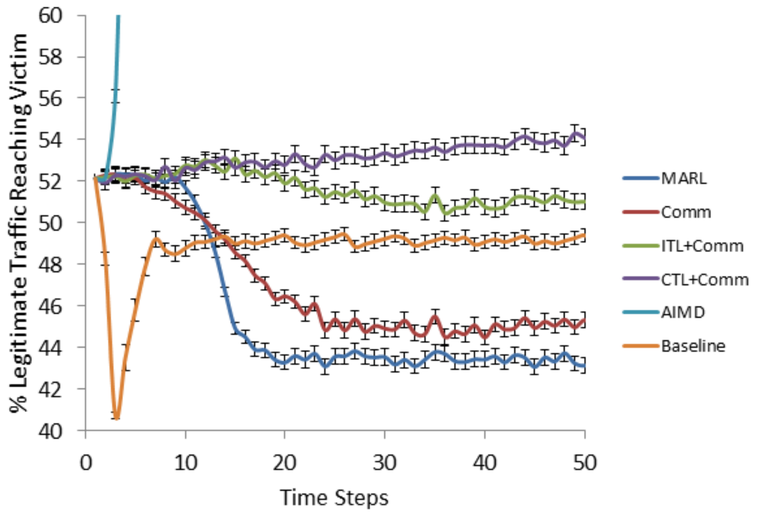
All approaches use an upper boundary for the victim of $U_s = 62$ and the Baseline and AIMD also use a lower boundary of $L_s = 56$. Each episode runs for 10000 time steps. At the start of each episode a new network instance is generated i.e. we re-chooce the legitimate users, attackers and their rates according to the model (described in Section 4.3). The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted; a higher value indicates a better performance.

Hierarchical communication is again shown to be beneficial to the system performance. Noteworthy is the fact that Comm requires a considerably larger amount of time (about 5000 time steps) to overcome MARL. It is also worth mentioning that although it performs better than MARL, this improvement is not as great as it used be in the offline learning setting. We will revisit the effect of hierarchical communication at a later experiment.

The AIMD approach outperforms the Baseline approach as expected. Similarly to the offline learning experimental results, our proposed CTL+Comm is shown to perform very well in the online learning scenario too. Specifically, CTL+Comm outperforms the Baseline, AIMD and all other learning-based approaches. However, it requires about 6000 time steps to overcome AIMD. Contrary to the offline learning setting where the quality of the learnt behaviour is the main objective, for online learning the learning speed is equally critical. The following experiments in this section investigate ways to improve the learning speed.



(a) Full View



(b) Close-up View

Figure 5.3: Preliminary Results Without Parameter Tuning

5.4.2 Reward Functions with a Gradient Throughout

Recall that difference rewards have been empirically demonstrated to work better if the original reward function has a gradient throughout (Agogino & Tumer 2004). This series of experiments examines how the reward functions with gradient throughout affect the approaches that do not use difference rewards.

Figure 5.4 shows how the reward functions with gradient throughout affect the basic MARL

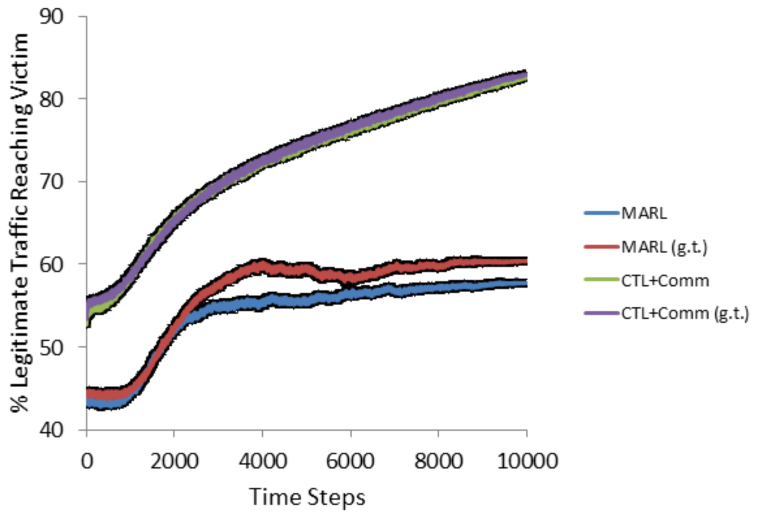


Figure 5.4: Reward Functions with a Gradient Throughout

and CTL+Comm approaches. It is shown that the basic MARL approach that uses the global G reward function with gradient throughout (Algorithm 8) performs better. This is attributed to the fact that agents using the global G reward function without gradient throughout are discouraged to take certain actions due to the big punishment of -1. The CTL+Comm approach has the same performance irrespective of whether the coordinated team CT reward function with gradient throughout (Algorithm 9) is used or not. From now on we use the reward functions with gradient throughout.

5.4.3 Default Action for Difference Rewards

The counterfactual $G(z_{-i})$ can be calculated by introducing the term c_i, which represents a constant action of agent i, which is (theoretically) executed in order to eliminate the contribution of agent i from the system. This experiment aims at determining which action is suitable for c_i.

Figure 5.5 shows how the performance of D_MARL varies with different dropping probabilities of the constant action c_i. The topology contains 30 reinforcement learning agents and the values are averaged over 500 episodes.

Clearly, the best result is when $c_i = 0.0$; in other words, an agent (theoretically) executes action 0.0 or drops 0% of its aggregate traffic in order to eliminate its contribution from the system. Difference rewards can then simply be calculated using:

$$D_i(z) = G(z) - G(z_{-i} + c_i) \tag{5.7}$$

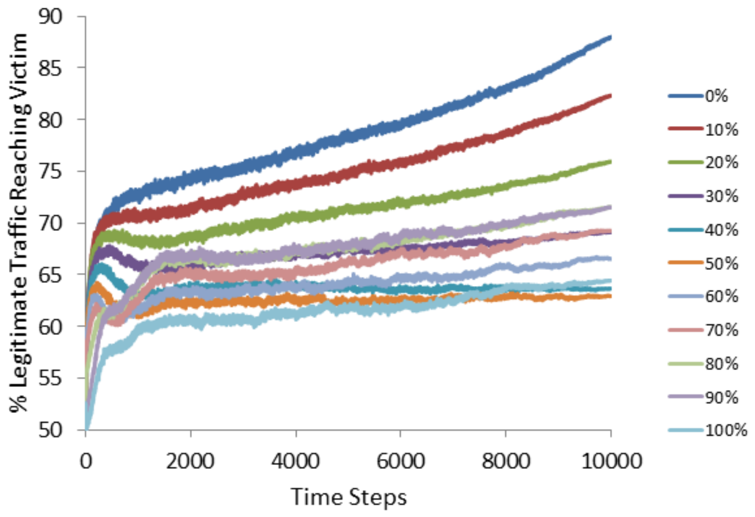


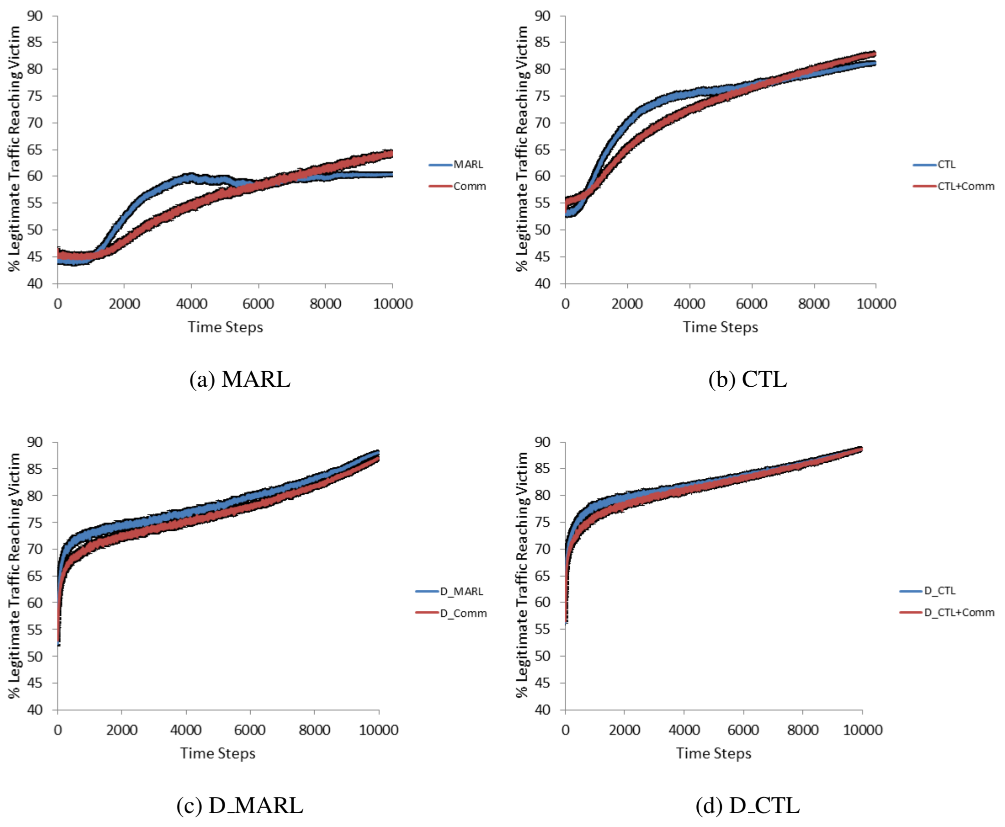
Figure 5.5: Default Action for Difference Rewards

5.4.4 Effect of Hierarchical Communication

This experiment aims at investigating whether hierarchical communication is beneficial in the online learning setting. Figures 5.6a - 5.6d demonstrate the effect of hierarchical communication for the MARL, CTL, D_MARL and D_CTL approaches respectively. The topology contains 30 reinforcement learning agents. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted.

We observe from Figure 5.6a that Comm requires a considerably larger amount of time to overcome MARL. This is expected as in Comm each learning agent has four state features, as opposed to an agent using MARL that has one state feature. It is also worth mentioning that although it performs better than MARL, this improvement is not as great as it used be in the offline learning setting. Recall that the goal of offline learning is for an agent to learn a universal policy, that is, the “best” response for all possible situations the network might be found in. Therefore, the observation makes sense since in the offline learning scenario hierarchical communication helps each agent to distinguish between the different situations and therefore to learn a better policy for similar ones. On the contrary the goal of online learning is to find the best response for the particular situation the network is currently found in.

As we move to more sophisticated approaches, the benefit of hierarchical communication disappears. For example, in Figure 5.6b the added benefit is very small while in Figure 5.6d it hinders the learning speed and offers no benefit to the final performance. In fact, in Figure 5.6c hierarchical communication works to the detriment of the system performance. Again, this makes sense since the incorporation of difference rewards is to provide a cleaner reward signal to the learning agents.



(a) MARL (b) CTL

(c) D_MARL (d) D_CTL

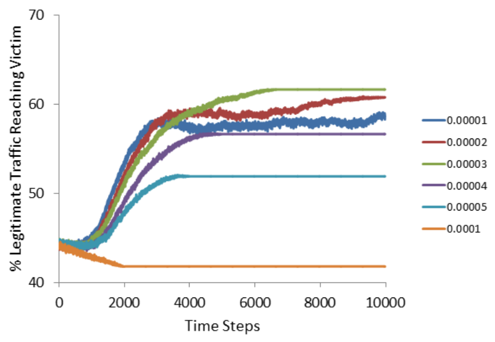
Figure 5.6: Effect of Hierarchical Communication

5.4.5 Effect of Exploration Strategy

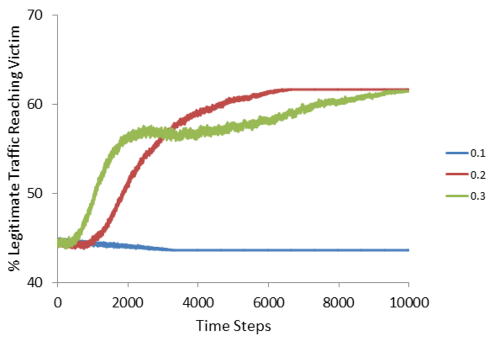
This series of experiments aims at investigating the effect of the exploration strategies on the learning speed. Two exploration strategies are investigated, namely, linearly decreasing and exponentially decreasing ϵ-greedy. For each strategy parameter tuning is performed for the decrease/decay rate δ and the initial value of ϵ. Experiments include 30 reinforcement learning agents.

Figures 5.7a and 5.7b show that the best system performance for MARL is obtained when $\delta = 0.00003$ and $\epsilon = 0.2$ for the linearly decreasing ϵ-greedy. The same is observed for the CTL approach in Figures 5.8a and 5.8b. The same experiment is repeated for the D_MARL approach. Figures 5.9a and 5.9b show that the best system performance is obtained when $\delta = 0.0001$ and $\epsilon = 0.2$.

The effect of exponentially decreasing ϵ-greedy strategy is also investigated for D_MARL approach. Figures 5.10a and 5.10b show that the best system performance is obtained when $\delta = 0.9999$ and $\epsilon = 0.05$. Figure 5.11 shows how the linearly and exponentially decreasing ϵ-greedy strategies compare to each other after parameter tuning. Clearly, the linearly decreasing

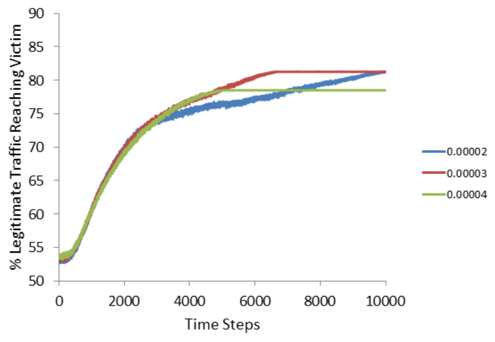


(a) Decrease Rate δ

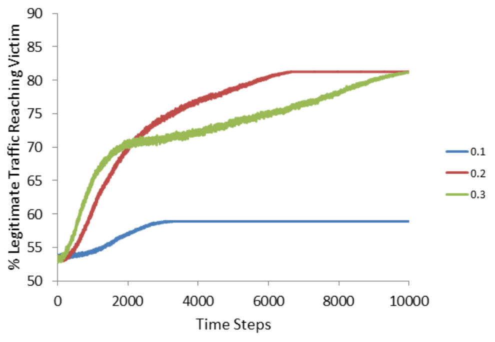


(b) Initial Value of ϵ

Figure 5.7: Linearly Decreasing ϵ-greedy for MARL

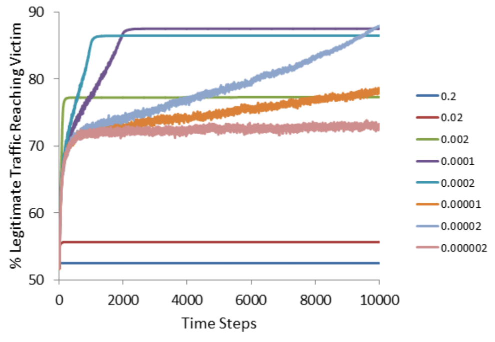


(a) Decrease Rate δ

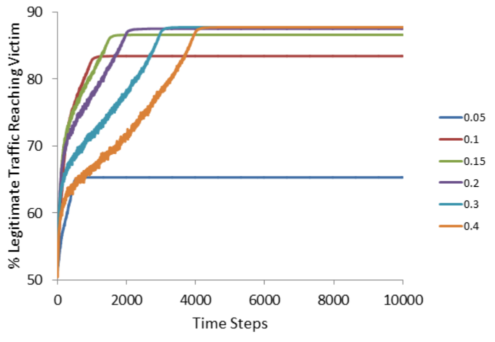


(b) Initial Value of ϵ

Figure 5.8: Linearly Decreasing ϵ-greedy for CTL

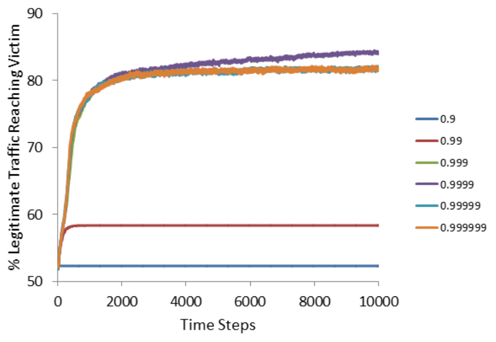


(a) Decrease Rate δ

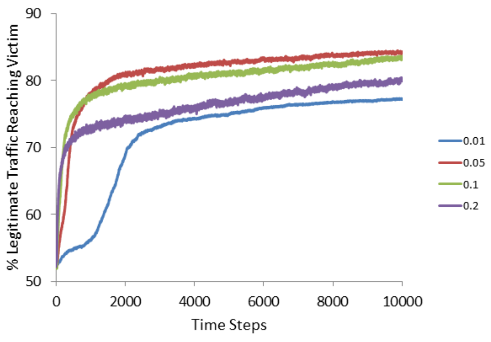


(b) Initial Value of ϵ

Figure 5.9: Linearly Decreasing ϵ-greedy for D_MARL



(a) Decrease Rate δ



(b) Initial Value of ϵ

Figure 5.10: Exponentially Decreasing ϵ-greedy for D_MARL

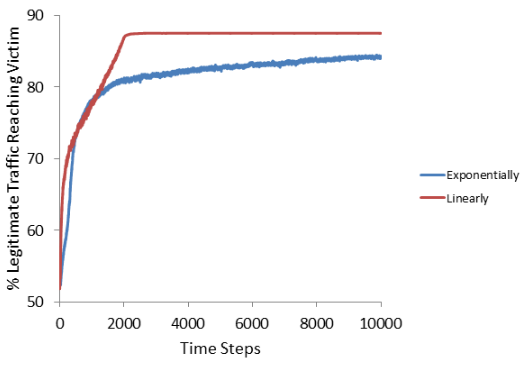
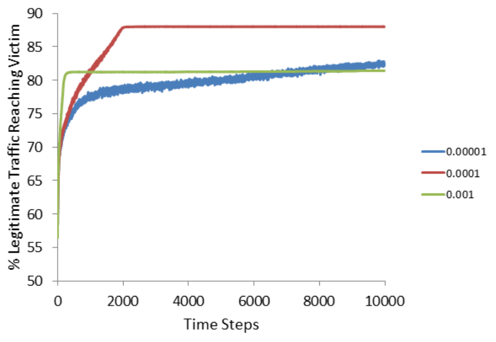
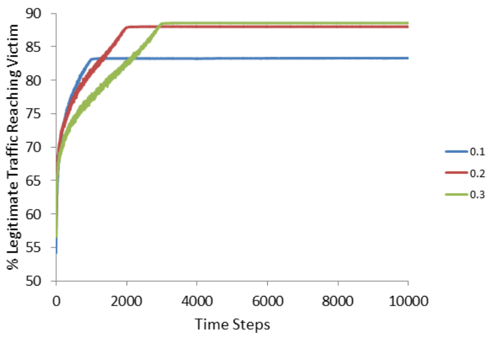


Figure 5.11: Linearly VS Exponentially Decreasing ϵ-greedy (D_MARL)



(a) Decrease Rate δ



(b) Initial Value of ϵ

Figure 5.12: Linearly Decreasing ϵ-greedy for D_CTL

Approach	Decrease Rate δ	Initial ϵ
MARL	0.00003	0.2
CTL	0.00003	0.2
D_MARL	0.0001	0.2
D_CTL	0.0001	0.2

Table 5.1: Parameter Tuning for Linearly Decreasing ϵ-greedy

strategy performs better.

Lastly, Figures 5.12a and 5.12b show that the best system performance for D_CTL is obtained when $\delta = 0.0001$ and $\epsilon = 0.2$ for the linearly decreasing ϵ-greedy. We summarise our findings in Table 5.1 and from now on we use these settings.

5.5 Online Learning Experiments

5.5.1 Scalability

This series of experiments aims at investigating the scalability of our proposed approaches. Figure 5.13 shows the performance of MARL when applied to topologies including up to 102 reinforcement learning agents. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted. As expected, the basic design MARL fails to scale-up; its performance declines as the number of learning agents increases. Specifically, its performance declines by 26% when the number of learning agents is increased from 12 to 102.

Figure 5.14 shows the performance of CTL when applied to topologies including up to 1002 reinforcement learning agents. A very strong result is obtained from this experiment. Our proposed approach is shown to be scalable since its performance remains unaffected by the addition of new teams of learning agents. Strictly speaking, although not visible in the graph, there is 0.7% difference in legitimate traffic reaching the victim between the highest performance (102 RLs) and the lowest performance (1002 RLs).

Figure 5.15 shows the behaviour of D_MARL when applied to topologies including up to 1002 reinforcement learning agents. The D_MARL approach is also shown to be scalable as its performance remains unaffected by the addition of new learning agents. Strictly speaking, although not visible in the graph, there is 1.2% difference in legitimate traffic reaching the victim between the highest performance (1002 RLs) and the lowest performance (30 RLs).

Lastly, Figure 5.16 shows the behaviour of our proposed approach D_CTL when applied to topologies including up to 1002 reinforcement learning agents. The D_CTL approach is highly scalable as well as its performance remains unaffected by the addition of new learning agents. Strictly speaking, although not visible in the graph, there is 1.9% difference in legitimate traffic reaching the victim between the highest performance (102 RLs) and the lowest performance (30 RLs).

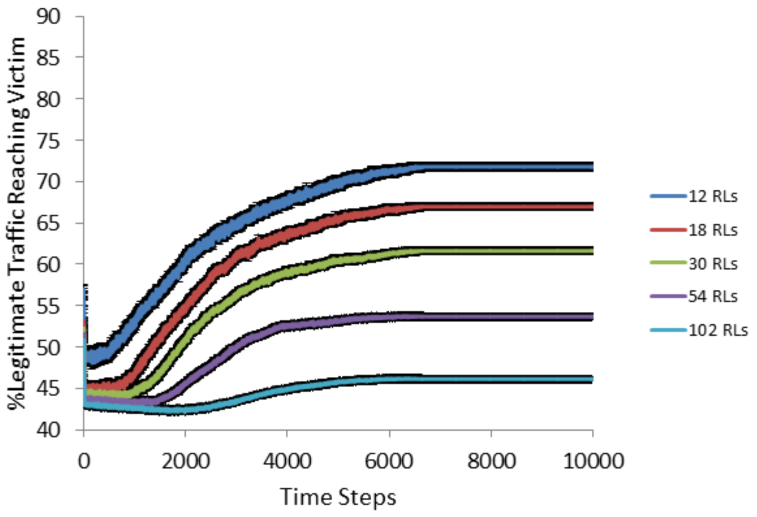


Figure 5.13: Scalability of MARL

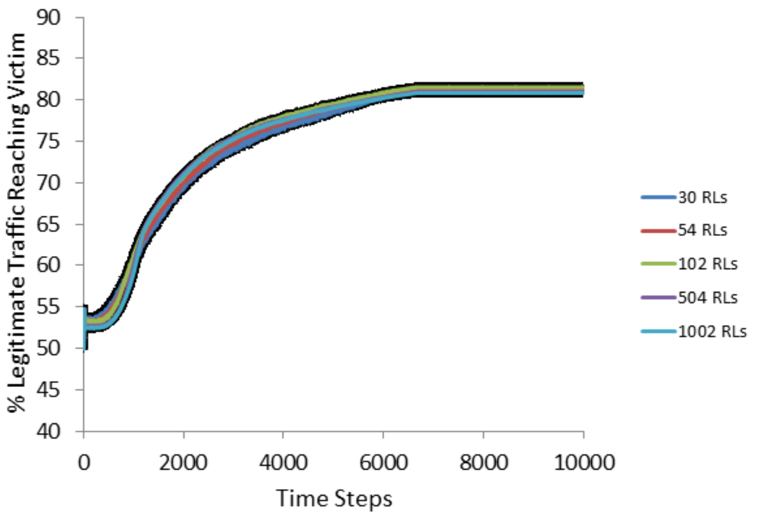


Figure 5.14: Scalability of CTL

Noteworthy are the two small "valleys" that are observed in the case of 504 and 1002 agents. This is attributed to the fact that the exploration parameter ϵ becomes zero at the start point of the "valley". This is often experienced in multiagent learning domains where the system performance starts declining when exploration hits zero. However, after a while the performance is again improved until it finally converges as learning continues even without exploration.

To further investigate how the individual approaches compare to each other the following experiments are conducted. Figures 5.17a -5.17c show how the approaches MARL, CTL, D_MARL

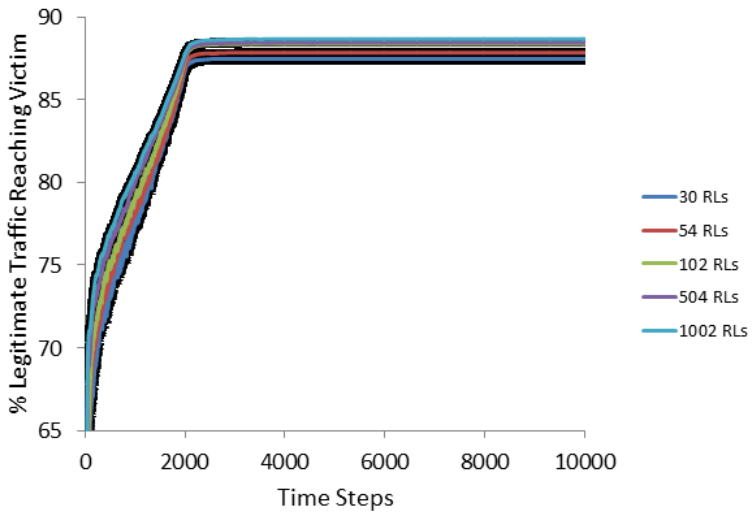


Figure 5.15: Scalability of D_MARL

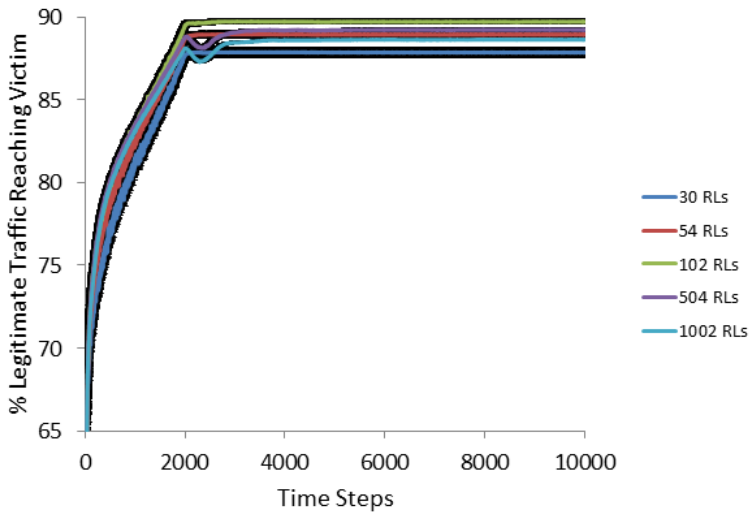
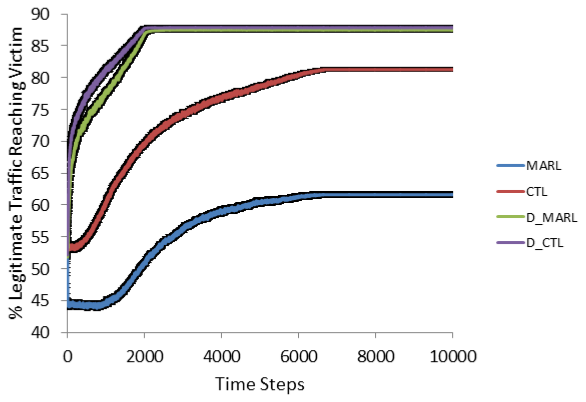


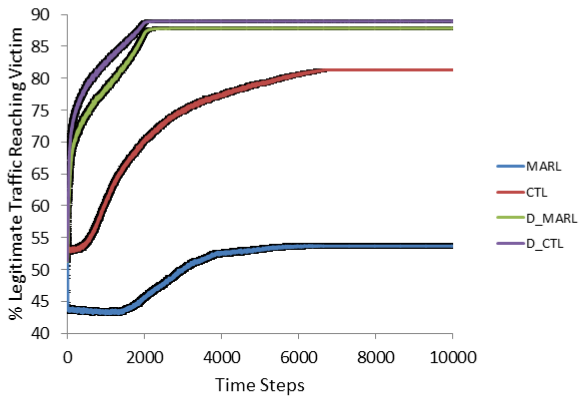
Figure 5.16: Scalability of D_CTL

and D_CTL compare to each other for the topologies involving 30, 54 and 102 reinforcement learning agents respectively. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted.

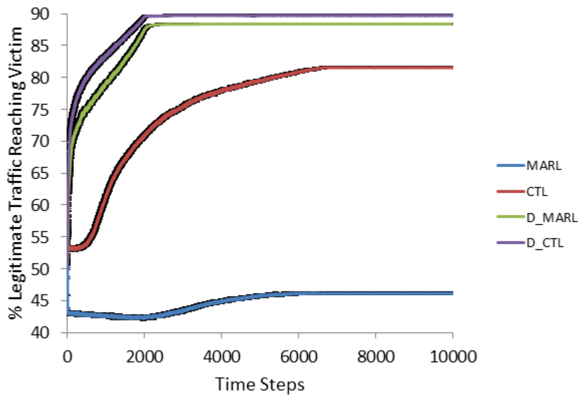
Specifically, for the 102 RLs case (Figure 5.17c) it is demonstrated that the basic MARL approach achieves a performance of 46%. Our proposed CTL achieves 81% thus outperforming MARL by 35%. D_MARL is shown to perform even better; it achieves 88% and outperforms MARL by 42%. Our proposed D_CTL combines the benefits of both CTL and D_MARL and



(a) 30 RLs

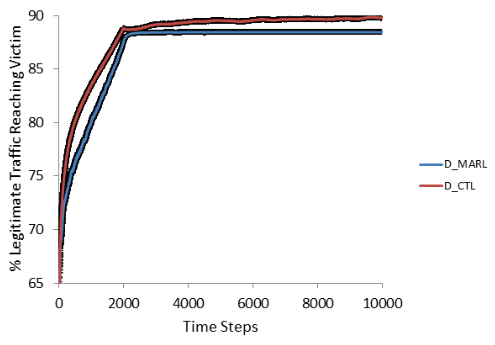


(b) 54 RLs

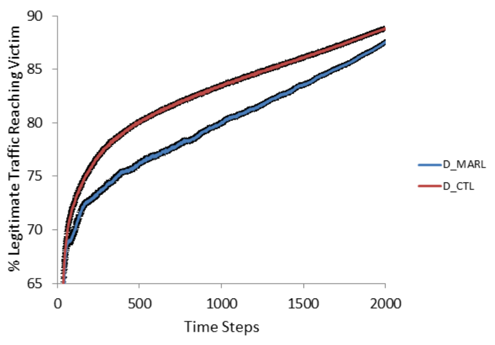


(c) 102 RLs

Figure 5.17: Online Learning for 30 - 102 RLs

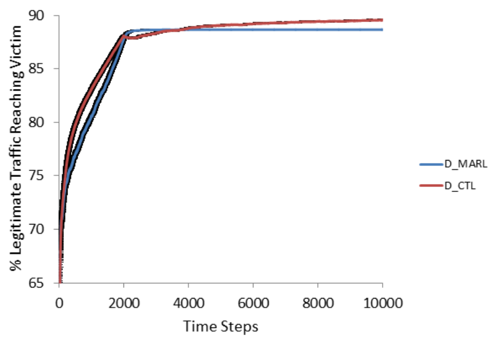


(a) Online Learning

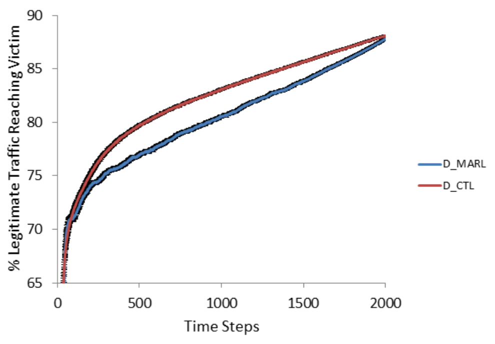


(b) Ramp-up Behaviour

Figure 5.18: Online Learning for 504 RLs



(a) Online Learning



(b) Ramp-up Behaviour

Figure 5.19: Online Learning for 1002 RLs

is shown to achieve 90% thus outperforming the basic MARL by 44%, but more importantly it learns faster than D_MARL.

We further investigate how the approaches D_MARL and D_CTL compare to each other in very large scenarios. This time, to avoid getting the "valleys" (shown earlier in Figure 5.16) we stop decreasing the exploration parameter ϵ for D_CTL in order to reach a value close to zero, but not zero.

Figure 5.18a shows how the two approaches compare to each other in the topology of 504 learning agents. Figure 5.18b shows the same graph but we "zoom" in to observe the ramp-up behaviour i.e. to examine the learning speed. The same experiment is repeated for 1002 learning agents as shown in Figures 5.19a and 5.19b.

As far as the system performance is concerned, D_CTL performs better than D_MARL in both the scenarios with 504 and 1002 agents (although to a lesser extent for the latter). However, we are particularly interested in the learning speed. As mentioned, contrary to offline learning where the quality of the learnt behaviour is the main objective, for online learning the learning speed is equally critical.

This is particularly the case for a real-world application like DDoS defence. Beyond the financial loss caused by DDoS attacks, victims also suffer loss to their reputation which results to customer or user dissatisfaction and loss of trust. It is demonstrated that our proposed approach D_CTL learns faster than D_MARL. For instance, from Figures 5.18b and 5.19b, D_CTL reaches 80% performance in about 500 time steps, while D_MARL requires about 1000.

How Scalable Does the Defensive System Need to Be?

In a study conducted by Spring et al. (2002), the authors consider 10 ISPs and estimate that the number of core (non-customer) routers in an ISP is between 11 and 1018. The former refers to a small ISP in India while the latter refers to a large corporate ISP in the US. It is observed that the difference is 100 times larger than the smallest networks. The study is now more than a decade old but can still provide us with a reasonable estimate.

To provide an estimate of the number of defensive routers consider a tree-structured network topology with a branching factor of two i.e. a binary tree. The root of the tree is the customer or victim router. A binary tree of depth four has a total number of core (i.e. excluding the root) routers $\#R_4 = 14$. Assuming the throttling routers are found in depth four (i.e. three hops away form the root) this gives a total number of defensive routers $\#R_4(3) = 8$. Similarly for a binary tree of depth 10 this gives $\#R_{10} = 1022$ and $\#R_{10}(9) = 512$. Notice that the all the figures mentioned constitute estimates.

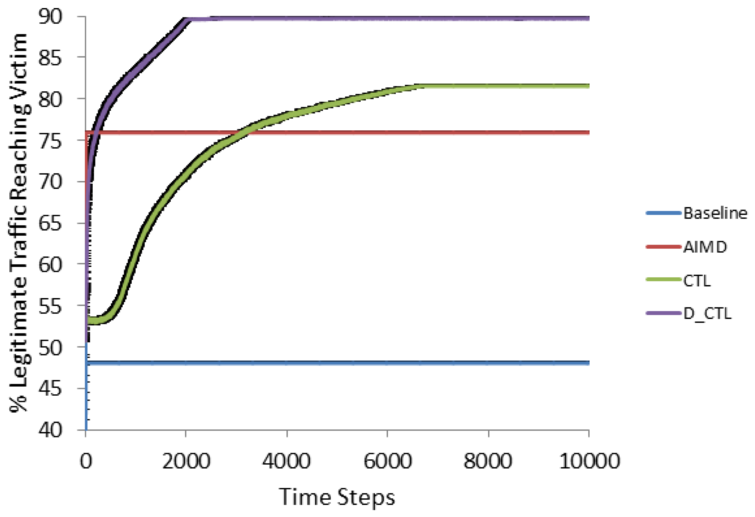
In fact, in practice it is expected that this number would be less than 512. This is because it is unlikely that an ISP would universally support the proposed functionality. Instead, it is expected that the proposed functionality will be supported on routers which see a substantial amount of network traffic. This holds true for the AIMD approach as well. As long as the majority of the DDoS traffic passes through these points our proposed approach would still be effective.

The empirical results in this section suggest that our proposed approach is scalable enough to be potentially deployed even in a large ISP network.

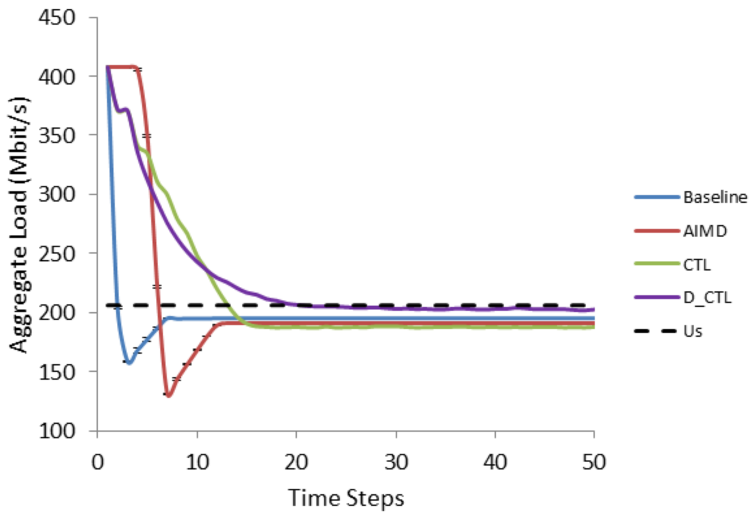
5.5.2 Performance

At this point we concentrate on CTL and D_CTL since they can better scale-up than MARL and D_MARL respectively. This experiment aims at investigating how the CTL and D_CTL approaches compare against the baseline and the popular AIMD throttling approaches (Yau et al. 2005). It further investigates their learning speed and examines the extent to which our approaches are deployable.

Figure 5.20a shows how the four approaches perform in the topology involving 102 learning agents for the constant-rate attack scenario. All approaches use an upper boundary for the victim of $U_s = 206$ and the Baseline and AIMD also use a lower boundary of $L_s = 184$. The values are averaged over 500 episodes and error bars showing the standard error around the mean are



(a) % Legitimate Traffic



(b) Aggregate Load

Figure 5.20: Performance for 102 RLs

plotted.

The AIMD approach outperforms the Baseline as expected. D_CTL, CTL and AIMD achieve a performance of 90%, 82% and 76% respectively. In other words, CTL and D_CTL outperform AIMD by 6% and 14% respectively. The CTL requires about 3000 time steps to equalise the performance of AIMD, and about 7000 to reach its maximum performance. The D_CTL requires about 200 time steps to equalise the performance of AIMD, and about 2000 to reach its maximum performance. In Section 5.7.2 we present a small case study that examines the final performance of AIMD, CTL and D_CTL by observing the exact rate-limits learnt and how they differ from AIMD's throttles.

For the same experiment, we show in Figure 5.20b how the aggregate load at the victim behaves for the four approaches. Recall from Section 3.1 that the server is assumed to be operating normally if its load is below the upper boundary i.e. $r_s \leq U_s$. All three approaches do what they are intended to do, that is, to bring down the aggregate load to acceptable levels i.e. $r_s <= U_s$ and $r_s \in [L_s, U_s]$ for the learning and non-learning approaches respectively. Specifically, the AIMD, CTL and D_CTL require 12, 13 and 19 time steps respectively to bring down the aggregate load to acceptable levels.

Learning Speed and Deployability
The question now is what all these figures mean in practice about the learning speed. Since we use a network emulator and also due to the lack of an actual ISP topology we can only provide an estimate of the learning speed. Recall from Section 3.2 that the monitoring window or time step of the throttling algorithms should be about the maximum round trip time between the victim server S and a router in $R(k)$. Assuming a maximum round trip time of $100 - 200\ ms$ means that the AIMD approach would require a few seconds to bring the aggregate load to acceptable levels (Figure 5.20b) and the same time to reach its maximum value (Figure 5.20a); this is consistent with experiments in (Yau et al. 2005).

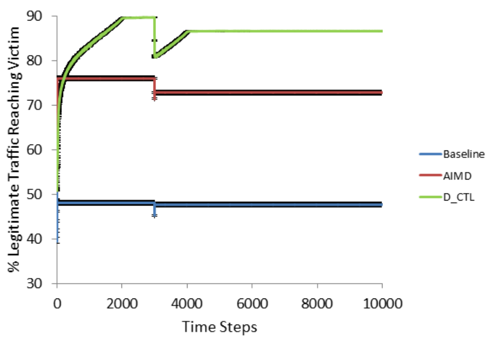
The CTL and D_CTL approaches would require a few more seconds to bring the aggregate load below the upper boundary (Figure 5.20b). However, CTL would require about 5-10 minutes to equalise the performance of AIMD and 12-23 minutes to reach its maximum performance (Figure 5.20a).

On the contrary, D_CTL would require less than a minute to overcome the performance of AIMD and it will reach its maximum performance in $3 - 7$ minutes (Figure 5.20a). We state that the figures just mentioned constitute estimates in order to provide an indication of the learning speed.

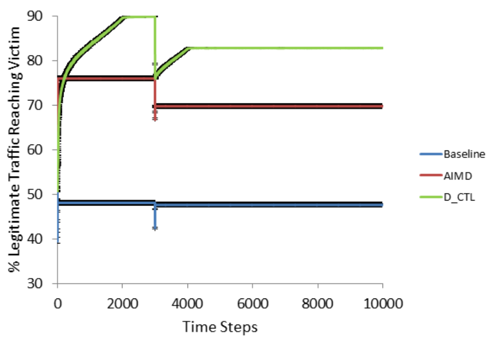
Taking into consideration that 88% of DDoS attacks last for less than an hour (Anstee et al. 2013), this powerful result suggests that our proposed D_CTL approach can potentially be deployed in a realistic ISP network. We will now focus on the D_CTL approach and run more experiments to stress test our reinforcement learning-based approach.

5.5.3 Performance with Agent Failures

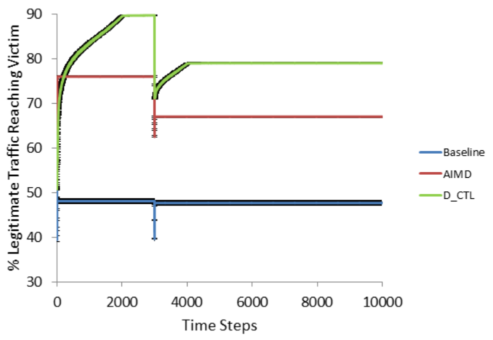
In real-world deployment router failures do and will occur therefore a defensive system must be robust to these failures. A router fails when it drops all of its incoming traffic (i.e. it executes action 1.0), thus not forwarding any traffic at all to the victim server. Figures 5.21a - 5.21d show how the system performance for 102 reinforcement learning agents (17 teams) is affected



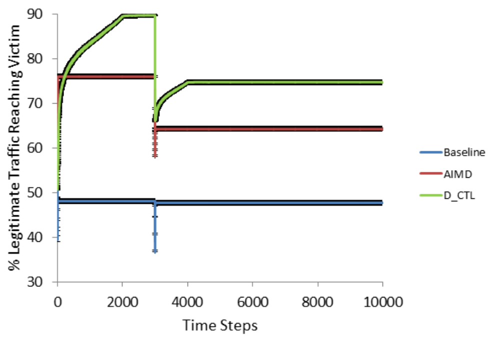
(a) 1 Team ($\approx$ 6% Agents)



(b) 2 Teams ($\approx$ 12% Agents)



(c) 3 Teams ($\approx$ 18% Agents)



(d) 4 Teams ($\approx$ 24% Agents)

Figure 5.21: Agent Failures

when 1, 2, 3 and 4 teams of agents fail after the 3000 time step. This corresponds approximately to 6%, 12%, 18% and 24% of the total number of defensive agents. At time step 3000 the exploration parameter is reset[3] to $\epsilon = 0.1$ and the defensive system re-trains for 1000 episodes to make up for the loss. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted. All three approaches are affected and show signs of performance degradation when the failures occur. However, it is demonstrated that agents in D_CTL are capable of adapting and can recover to make up for the loss. In all scenarios, D_CTL outperforms AIMD by 11-14%. Interestingly, the system even with almost a quarter of agents failing, it achieves a performance of 75%, which is very similar to AIMD's performance without failures (76%).

[3]In practice, it is expected that the victim will instruct the activated system to start exploring if the victim observes a rapid decline in its performance or a decline over a period of time. Alternatively, it is reasonable to expect that the network administrator is aware of any router failures, therefore he can manually instruct the system to start exploring.

5.5.4 Performance with Variable-rate Attack Dynamics

The majority of DDoS attacks use a constant-rate mechanism where agent machines generate traffic at a steady rate (Mirkovic & Reiher 2004). The DDoS impact is rapid but the large and continuous flood can be easily detected. It is also the most cost-effective method for the attacker. However, the attacker can also deploy a variable rate mechanism to delay or avoid detection and response.

Figures 5.22a -5.22c show how the approaches behave against a pulse DDoS attack for 102 learning agents. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted. The attack rate oscillates between the maximum rate and zero, and the duration of the active and inactive periods is the same and set to 500 time steps, that is, about 50-100 seconds (assuming a time step of 100-200 ms). For example, the first attack pulse starts at time step 0 and stops at 500. The second pulse starts at time step 1000 and stops at 1500. It is shown that D_CTL outperforms AIMD during the active periods of the pulse attack; furthermore, agents in D_CTL learn not to rate-limit any traffic during the inactive periods of the attack.

5.5.5 Performance with Measurement Noise

In a realistic network, measurement noise from routers exist. This is the result for example of traffic delays and packet losses. Noise is defined in our model as a random value added to a measurement as shown below:

$$\phi = rand(-2, 2) \tag{5.8}$$

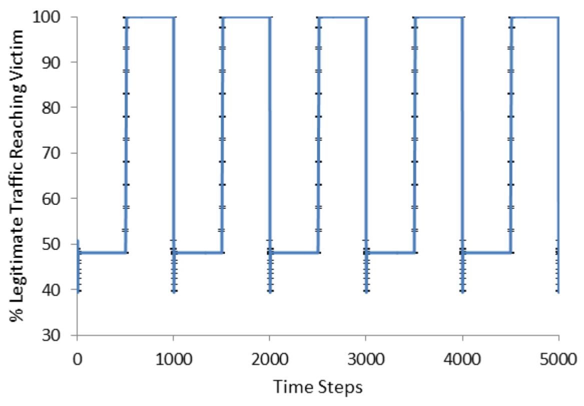
Noise is altered at every time step; the noisy measurement value or reading from a router at time step t is given by:

$$noisy\_reading_t = reading + \phi_t \tag{5.9}$$

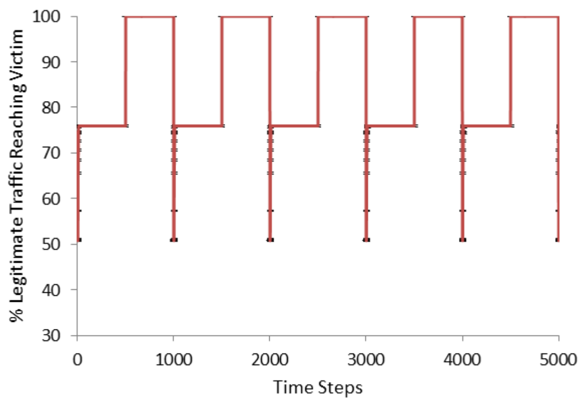
We start by investigating the effect of noise on all the approaches. Figures 5.23a - 5.23b show how the presence of noise affects the approaches MARL, CTL, D_MARL and D_CTL for 102 learning agents. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted.

MARL is once again shown to be an unsuitable approach; the measurement noise combined with the scalability problems results to a very poor performance. CTL performs orders of magnitude better than the basic MARL but is also shown to be heavily affected by the presence of noise.

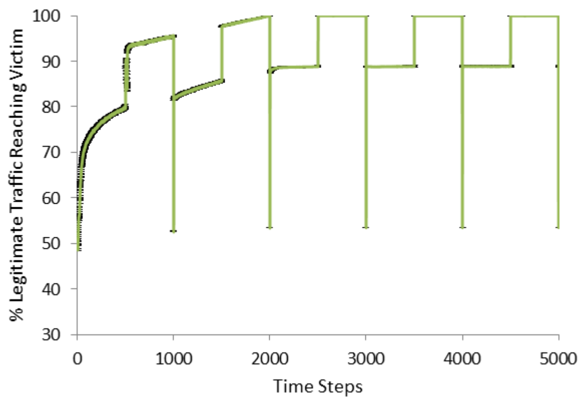
The difference rewards D_MARL method is also affected but to a lesser degree. This is expected as difference rewards have been demonstrated to be robust to measurement noise (Colby & Tumer 2013). Our proposed approach D_CTL is affected the least and is shown to be robust to measurement noise. Initially, it has a similar performance to D_MARL without the presence of noise. Towards the end of the experiment it equalises D_CTL as if no noise was present.



(a) Baseline

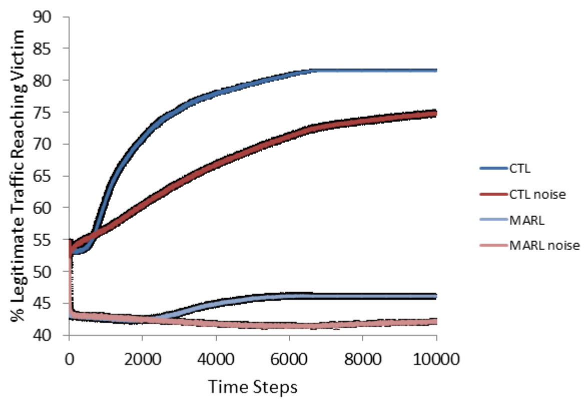


(b) AIMD

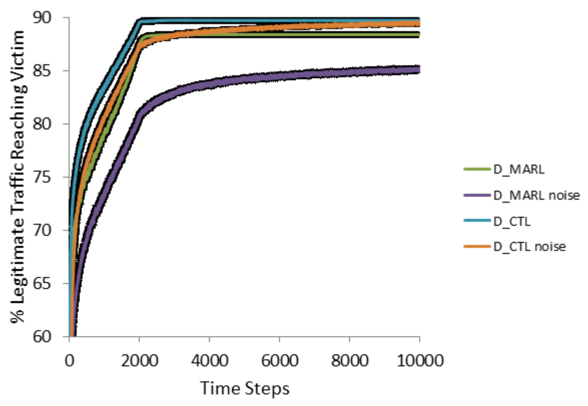


(c) D_CTL

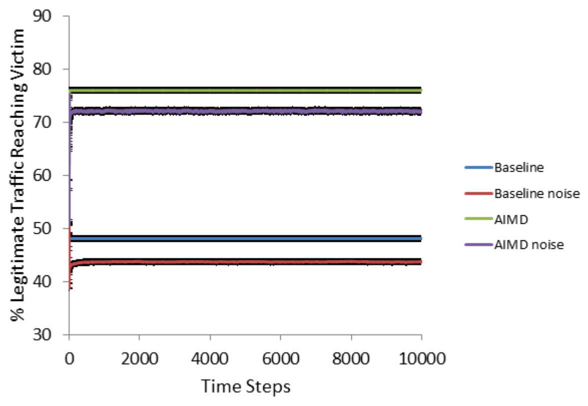
Figure 5.22: Pulse Attack



(a) MARL and CTL

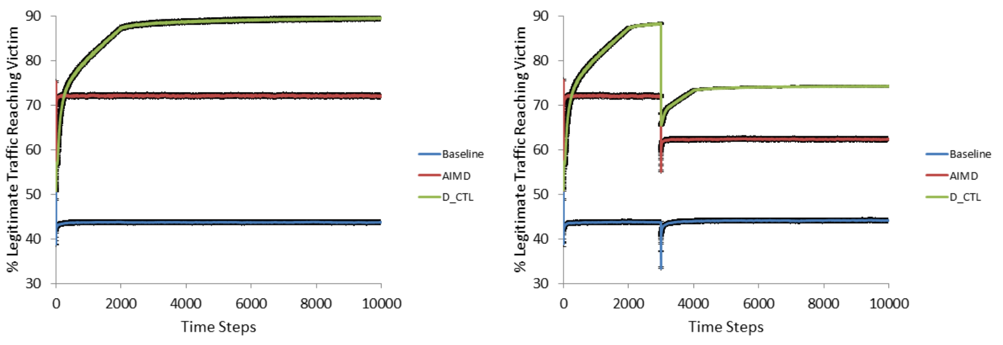


(b) D_MARL and D_CTL



(c) Non-learning Approaches

Figure 5.23: Effect of Measurement Noise



(a) Without failures (b) With $\approx 24\%$ Failures

Figure 5.24: Measurement Noise

This constitutes an another example that shows the benefit of combining our proposed CTL with difference rewards.

At this point recall from Section 5.5.2 that for the topology involving 102 agents the upper boundary is $U_s = 206$; as far as the non-learning approaches i.e. Baseline and AIMD are concerned, their lower boundary parameter so far was set to $L_s = 184$. However, in this highly noisy scenario, by keeping $L_s = 184$ the non-learning approaches experience stability problems as the aggregate load oscillates and never stabilises within $[L_s, U_s]$.

One of the motivations for online learning is that non-learning and offline learning approaches require to have a reasonable knowledge of the network model. If this knowledge is inaccurate, parameter re-tuning and re-training is required respectively. Therefore, we have manually altered the lower boundary parameter for the non-learning approaches to $L_s = 150$ to avoid the stability issues. This is a typical example demonstrating the advantage of online learning and being capable of adapting to the new situation.

Figure 5.23c shows how the presence of noise affects the Baseline and AIMD approaches. Under the new setting of $L_s = 150$, their performance is not much affected and are shown to be robust to measurement noise.

Figure 5.24a shows how D_CTL compares against the Baseline and AIMD approaches in the presence of noise for 102 learning agents. The D_CTL approach outperforms AIMD by 17%. Interestingly, this constitutes an improvement over the situation where noise was not present; recall from Figure 5.20a that the difference was 14%. Lastly, Figure 5.24b shows how D_CTL compares against the Baseline and AIMD approaches in the presence of both noise and $\sim 24\%$ agent failures. The D_CTL approach outperforms AIMD by 12%. D_CTL is robust to both measurement noise and agent failures.

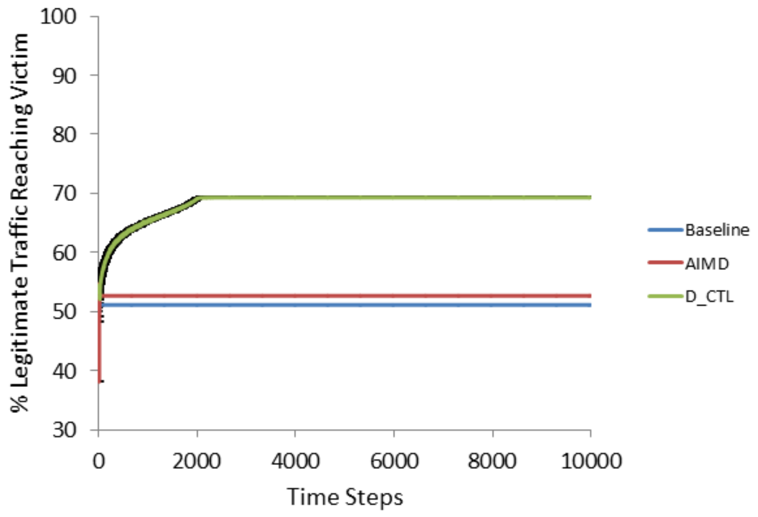


Figure 5.25: Meek Attackers

5.5.6 Performance with "Meek" Attackers

We have assumed so far that agent or zombie machines are aggressive i.e. their sending rate is significantly higher than that of a legitimate user. Indeed, that is a realistic assumption since in the case of "meek" attackers i.e. when a zombie machine sends at a similar rate to a legitimate user, this would require the attacker to compromise a considerably larger number of machines in order to launch an attack with a similar effect.

Both legitimate users and attackers send traffic at constant rates, randomly and uniformly drawn from the range $[0, 1]$ $Mbit/s$ (recall from Section 4.3 that the range for aggressive attackers used to be $[2.5, 6]$). Figure 5.25 shows the performance of Baseline, AIMD and D_CTL in the presence of "meek" attackers. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted. All approaches use an upper boundary for the victim of $U_s = 56$ and the Baseline and AIMD also use a lower boundary of $L_s = 50$.

It is observed that the Baseline and AIMD approaches have now a similar performance. This is expected as their authors (Yau et al. 2005) admit that "... our approach is most useful under the assumption that attackers are significantly more aggressive than regular users. [Otherwise] our solution is then mainly useful in ensuring that a server under attack can remain functional within the engineered load limits". Although the D_CTL approach is heavily affected as well, it achieves a system performance of 69%, allowing 17% of legitimate traffic more than AIMD to reach the victim server.

To more effectively tackle the "meek" attackers problem it would require the designer to enrich the state feature space, that is, to introduce more statistical features other than the router load. This would be helpful because in the case of "meek" attackers, the system cannot differentiate between legitimate and attack traffic by just taking into account the traffic volume.

5.5.7 Number of Teams

This experiment aims at investigating how the system behaviour is affected by the number of teams involved in the defence mechanism. In practice, this is typically depended on the underlying network topology. Also, if the defence spans multiple collaborative network domains, then agents in the same domain will likely form a separate team.

We assume homogeneous teams of agents. Figure 5.26a shows the performance of D_CTL with 2-18 teams in the topology involving 18 reinforcement learning agents. For example in the case of 2 teams each team has 9 reinforcement learning agents while in the case of 18 each team has a single learning agent. It is observed that D_CTL with 2-6 teams perform the same and better than D_CTL with 9-18 teams, but they all outperform AIMD. We can deduce that the final performance declines when a large number of teams (i.e. fewer learning agents per team) is used.

Figure 5.26b shows the same graph but focuses on the ramp-up behaviour to examine the learning speed. It is observed that D_CTL with 18 teams has the fastest learning speed respectively. We can deduce that as the number of teams increase (i.e. fewer learning agents per team) the system starts learning faster. To verify this we have included the D_MARL approach and observe that it indeed has the slowest learning speed.

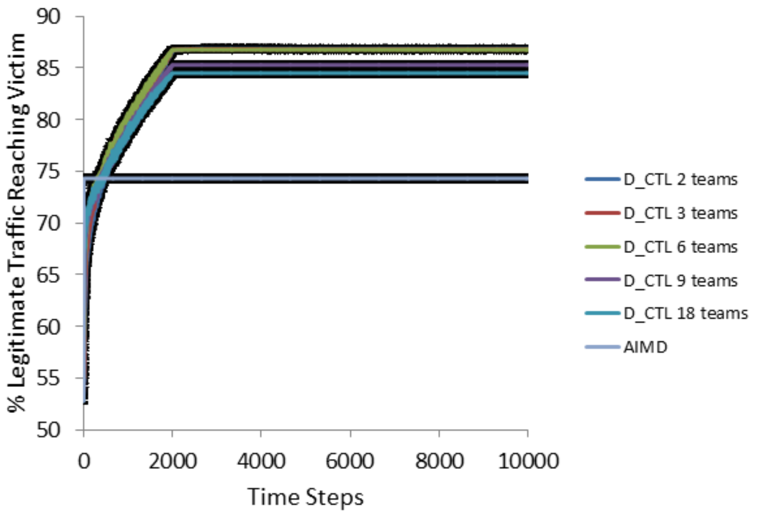
These findings are attributed to the fact that in cases where the number of teams is large i.e. fewer learning agents per team, the agents receive more localised rewards (hence the faster learning speed), but this discourages coordination (hence the poorer performance). However, the important point is that the performance of the proposed D_CTL approach is not heavily affected by the number of teams of learning agents in the system as it always outperforms the AIMD approach.

5.5.8 Incremental Learning

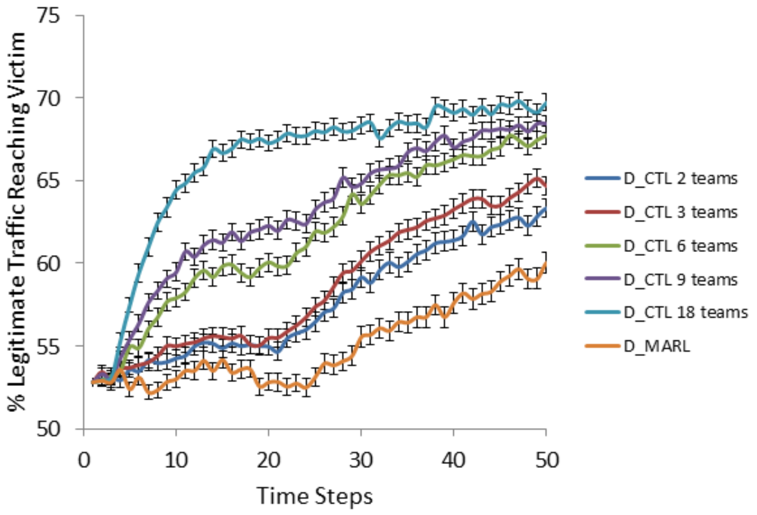
So far we have assumed that agents learn from scratch during online learning. However, to facilitate the online learning process and specifically to increase the learning speed, agents can be initialised with prior, reasonably good behaviours and then continue learning as normal. We call this incremental learning.

One way to achieve that is to initialise the Q-tables of the reinforcement learning agents with their universal policies learnt during offline learning (Chapter 4). Recall that during offline learning agents learn a universal policy, that is, the best behaviour over all possible situations the network might be found in.

Before digging further into incremental learning we need to repeat some of the offline learning experiments using the new reward functions with gradient throughout. We repeat the experiments from Section 4.4.1 regarding the constant-rate attack scenario. Previous results are shown in Figure 4.4.



(a) Performance



(b) Ramp-up Behaviour

Figure 5.26: Number of Teams

Figure 5.27 shows the new results for the CTL and CTL+Comm approaches using the reward functions with gradient throughout, and the difference rewards-based approaches. For offline learning the best approach remains the CTL+Comm. In other words, the incorporation of difference rewards (D_CTL+Comm) slightly worsens the system performance. The second best is the D_CTL approach.

The policies learnt during offline learning for the D_CTL approach are now used to initialise

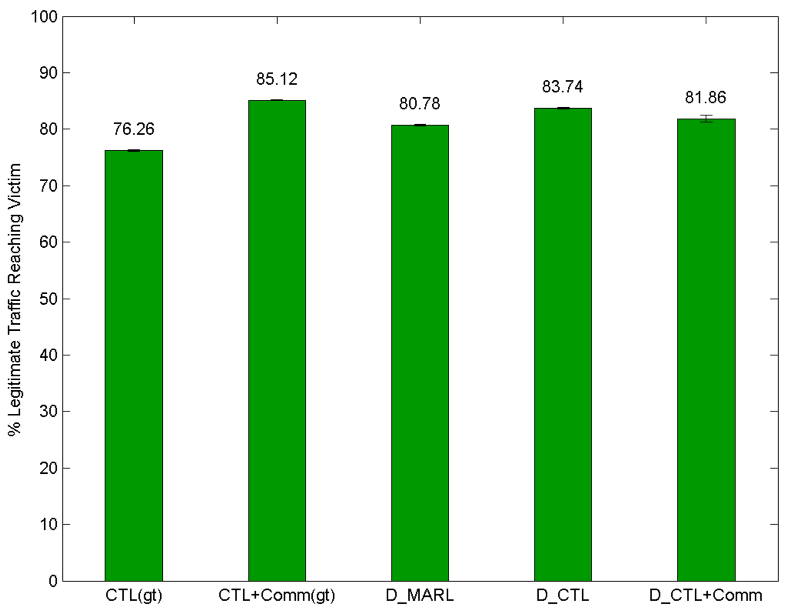


Figure 5.27: Offline Learning

the Q-tables of the agents in order to facilitate the online learning process. Incremental learning is depicted in Figure 5.28 for 30 reinforcement learning agents. The values are averaged over 500 episodes and error bars showing the standard error around the mean are plotted. The graph shows how behaviours learnt during offline, online and incremental compare against the AIMD approach.

The first thing to notice is that the behaviour learnt during offline learning outperforms the AIMD approach. This is in alignment with results from Figure 5.27. Secondly, the behaviour learnt during online learning outperforms the universal behaviour learnt during offline learning. This is somewhat expected as the best behaviour for a particular situation is likely to be better than the best behaviour across all situations.

Lastly, incremental learning combines the advantages of both offline and online learning and is shown to be beneficial to the system. D_CTL with incremental learning immediately starts with the same performance as AIMD and outperforms it throughout the experiment until it reaches its maximum value at time step 2000 (notice that the starting point of incremental learning is lower than offline learning's because of exploration).

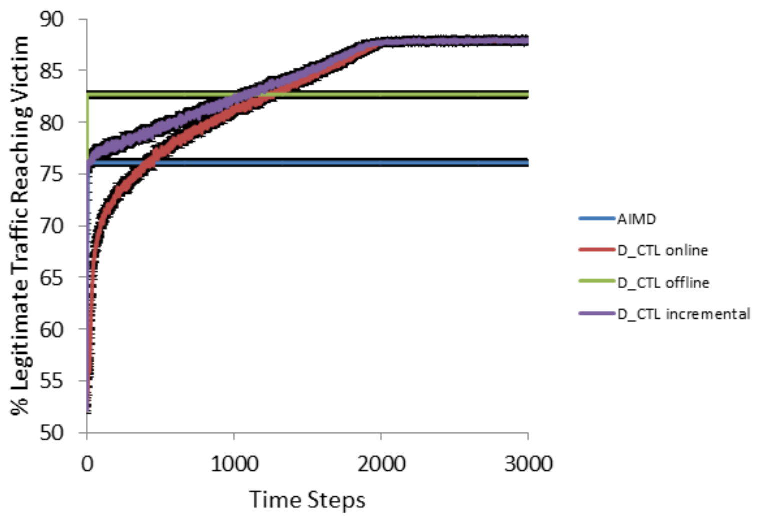


Figure 5.28: Incremental Learning

5.6 Computational Complexity and Communication Overhead

Let us start by discussing the set of defensive routers. For the selection of upstream defensive routers we have adopted the same method by Yau et al. (2005). However, in practise there are hundreds of potential victims and it is unlikely that for each victim i, an ISP would support the proposed functionality on a separate set of defensive routers $R_i(k_i)$. This holds true for the AIMD approach as well. It is expected that in real-world deployment an ISP will support the defensive functionality on distributed fixed locations i.e. routers which see a significant amount of network traffic. As long as the majority of the DDoS traffic passes through these points our proposed approach would still be effective. In other words, the set of defensive routers is fixed irrespective of the number of victim servers.

Let us now talk about the number of victim servers that need to be protected. As discussed in Section 5.3.2 the servers under attack can request protection on-demand (possibly by providing a payment). The amount of information stored at a throttling router is linear in the number of servers under attack (as opposed to offline learning where all potential victims should be considered and thus being infeasible). Our approach is feasible because it is expected that at most only a minor portion of the network will ask for protection as the DDoS attacks are not the norm, but the exception (Yau et al. 2005).

We now discuss the communication overhead of our approach. As mentioned, the proposed D_CTL approach requires the victim (or the network administrator) to send signals for activating the defensive system and for instructing the system to explore more (e.g. in cases of component failures).

Furthermore, as discussed in Section 5.6, we have assumed that each learning agent calculates its own difference rewards signal. To achieve that, the victim and team leaders need to communicate to the learning agents the relevant information at each time step. It is emphasised that communication is performed in a hierarchical team-based fashion and no information exchange between the learning agents takes place (i.e. decentralised coordination). Although we haven't actually implemented our approach in a realistic network, there are many existing techniques that have been deployed which make use of hierarchical communication (examples include Pushback and DefCOM both described in Section 2.3.5). Therefore, we believe that the proposed communication scheme is simple and minimal, and the introduced overhead is not restrictive for a real-world deployment. To eliminate this overhead completely, an approximation of difference rewards may be possible as in (Tumer & Agogino 2007), but this is subject to future work.

In real-world deployment we need to ensure a reliable and secure delivery of the communicated information. In practise, our proposed system will require the designer to include mechanisms like message authentication, priority transmission and retransmission in case of losses. Notice that this applies to other approaches as well such as the AIMD approach in order to have a reliable and secure delivery of the throttle signals (Yau et al. 2005).

Lastly, we examine the mechanism behind rate-limiting. For the purpose of our experiments we have used a simplistic method called preferential dropping (Mahajan et al. 2001). For instance, assume that the aggregate's arrival rate is r_a (e.g. 10 Mbps) and a specified bandwidth limit for the aggregate is r_l (e.g. 7.5 Mbps). The rate-limiter will drop each arriving packet in the aggregate with probability $1 - r_l/r_a$ (e.g. 0.25). No packet will be dropped when $r_a \leq r_l$. Despite the simplicity of this approach, its drawback is the fact that a random number needs to be generated for every arriving packet in the aggregate, in order to decide whether it will be dropped or not. This could be infeasible for large DDoS attacks.

An alternative approach is to use a virtual queue for rate-limiting (Mahajan et al. 2001). A virtual queue is slightly more complex to implement but eliminates the aforementioned overhead. As a consequence, our reinforcement learning design will need to be modified as well; instead of learning dropping probabilities, agents will need to learn appropriate sizes for the virtual queue. This is subject to future work. Lastly, when considering a realistic deployment, it is useful to know that virtual queues are found in some commercial routers like Cisco's (Mahajan et al. 2001).

5.7 Case Studies

5.7.1 A Walkthrough Example on Calculating the Difference Rewards

Consider the network topology depicted in Figure 5.29. A dashed line means that two routers are not necessarily directly connected. Legitimate users H1, H2, H4 and H5 send at a rate of 1 while attackers H3 and H6 send at a rate of 5 (shown next to the hosts in the figure). Also shown is the

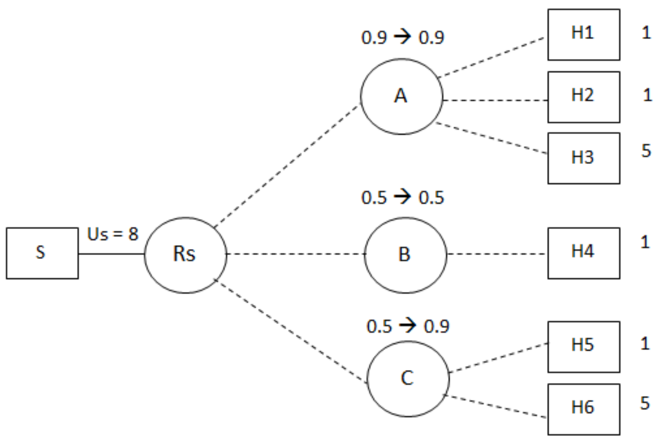


Figure 5.29: Example of a Network Topology

upper boundary for the victim server which is set to $U_s = 8$.

Reinforcement learning agents are installed on routers A, B and C. Recall that for calculating the counterfactual the default action of an agent to eliminate its contribution is to drop nothing i.e. to allow everything. At time step 1 agents execute actions 0.9, 0.5 and 0.5 respectively. At time step 2 agent C performs action 0.9 while A and B perform the same i.e. 0.9 and 0.5 respectively (shown above the routers in the figure).

To demonstrate the application of difference rewards we use the MARL and D_MARL approaches that use the global G reward function as given by Algorithm 8 (for CTL and D_CTL the difference rewards can be calculated in a very similar manner by considering the hypothetical victims i.e. the team leaders instead of the actual victim).

Calculating the Global Reward

At this point we repeat that we are interested in cases where the detection mechanism cannot precisely characterise the attack traffic i.e. when attack signatures cannot be derived. Inevitably, in practice legitimate traffic can only be estimated.

Let us now calculate the global reward G_1 at the first time step. The victim observes a total traffic of 4.2 since routers A, B and C perform actions 0.9, 0.5 and 0.5 respectively. This is below the victim's upper boundary therefore no punishment will be given.

The victim knows that the usual total legitimate traffic it receives is 4. This was measured (prior the attack) for example by observing the behaviour or habits of customers and regular users or visitors of the victim's services. Also, during the ongoing DDoS attack the victim can estimate that it is currently receiving 1.2 of legitimate traffic. Therefore, the victim sends out the global G_1 reward shown below to the learning agents.

$$G_1 = \frac{1.2}{4} \tag{5.10}$$

Calculating the Counterfactual

It is assumed that the difference rewards signal of an agent, is calculated by the agent itself[4]. This will require extra communication from the victim to each learning agent. Specifically, the victim needs to send out three pieces of information as follows:

$$< current_{legit}, current_{aggr}, usual_{legit} >$$

where $current_{legit}$ and $current_{aggr}$ are the estimated legitimate traffic and the aggregate traffic respectively that the victim receives during an attack at a particular time step, and $usual_{legit}$ is the legitimate traffic the victim usually receives (obviously the latter does not need to be communicated at every time step).

Let us now calculate the difference rewards D_{1_C} for agent C at the first time step. This will be given by:

$$D_{1_C} = G_1 - G_{1_{-C}} \tag{5.11}$$

where $G_{1_{-C}}$ is the counterfactual i.e. the global reward G_1 without the contribution of agent C.

Agent C receives from the victim $< 1.2, 4.2, 4 >$. Agent C can therefore calculate the first part of the equation i.e. the global reward. Specifically, since 4.2 is less than U_s then the global reward is given by $G_1 = 1.2/4$.

The counterfactual $G_{1_{-C}}$ is calculated as follows. If agent C was eliminated from the system (i.e. had agent C performed action 0.0), the victim would receive a total traffic of $4.2-3+6 = 7.2$ which is below U_s and therefore no punishment would be given. Therefore, the counterfactual given would be as shown below:

$$G_{1_{-C}} = \frac{1.2 - 0.5 + 1.0}{4} = \frac{1.7}{4} \tag{5.12}$$

Demonstration of Difference Rewards

With the same reasoning we calculate the rewards for the rest of the agents. At time step 1 the global reward G_1 and the difference rewards for each agent D_{A1}, D_{B1}, D_{C1} are calculated as shown in Equations 5.13-5.16. For time step 2 we calculate the rewards as shown in Equations 5.17-5.20.

[4]An alternative implementation is the victim to calculate the difference rewards signal for each throttling agent. This however would require the victim to be able to derive statistics on a per throttling router basis.

$$G_1 = \frac{1.2}{4} = 0.3 \tag{5.13}$$

$$D_{1_A} = G_1 - G_{1_{-A}} = \frac{1.2}{4} - (-\frac{10.5}{16}) = 0.96 \tag{5.14}$$

$$D_{1_B} = G_1 - G_{1_{-B}} = \frac{1.2}{4} - (\frac{1.7}{4}) = -0.13 \tag{5.15}$$

$$D_{1_C} = G_1 - G_{1_{-C}} = \frac{1.2}{4} - (\frac{1.7}{4}) = -0.13 \tag{5.16}$$

$$G_2 = \frac{0.8}{4} = 0.2 \tag{5.17}$$

$$D_{2_A} = \frac{0.8}{4} - (-\frac{8.1}{16}) = 0.71 \tag{5.18}$$

$$D_{2_B} = \frac{0.8}{4} - (\frac{1.3}{4}) = -0.125 \tag{5.19}$$

$$D_{2_C} = \frac{0.8}{4} - (\frac{1.7}{4}) = -0.225 \tag{5.20}$$

Recall from section 5.2 that any action taken that increases (or decreases) D_i simultaneously increases (or decreases) G. Indeed this is what we observe here as shown in Equations 5.21-5.22.

$$G_2 - G_1 = (-)ve \tag{5.21}$$

$$D_{2_C} - D_{1_C} = (-)ve \tag{5.22}$$

5.7.2 Comparison Between (D_)CTL and AIMD throttles

We consider a particular instance of the topology involving 12 RL agents and 24 host machines with a lower and upper boundaries of $L_s = 20$ and $U_s = 26$ respectively. This is shown by columns $D - F$ in Table 5.2. For example, throttling router $R1$ (column D) receives 1.55 and 5.30 $Mbit/s$ of legitimate and DDoS traffic respectively (column F), thus making a total of 6.85 (column E). The bottom part (last couple of rows) of the table provides some statistics. For example, the total legitimate traffic is 6.50 and the aggregate traffic is 47.35 which is way above the upper boundary.

After running the AIMD algorithm for some time, it settles at a throttle of 2.24 and sends it to the upstream throttling routers. This is depicted in the remaining columns $A - C$ of Table 5.2. For example, in order for router $R1$ to bring its load down from 6.85 (column E) to 2.24 (column B) it has to execute action $1 - 2.24/6.85 = 0.67$ (column C) or drop about 67% of the traffic. Therefore, 0.51 legitimate and 1.73 DDoS traffic (column A) will reach the victim server. Let us now consider router $R4$; since its incoming load (0.75) is below the throttle (2.24) then it does not drop any traffic. Again, the bottom part of the table shows some statistics. After AIMD throttling, the aggregate load is brought to 21.25 which is in the desired range, and allows about 70% of the legitimate traffic to reach the victim during the DDoS attack.

Table 5.3 shows the same example for our proposed CTL approach. We can now observe that after the learnt dropping probabilities are performed, the defensive system allows about 80% of the legitimate traffic to reach the victim server during the attack i.e. about 10% more than the AIMD approach.

Similarly, Table 5.4 shows the same example for D_CTL approach where the defensive system allows about 90% of the legitimate traffic to reach the victim server during the attack i.e. about 20% and 10% more than the AIMD and CTL respectively.

Some examples on why this is case are the following. In the case of router $R10$ the AIMD, CTL and D_CTL drop 46%, 10% and 0% respectively; D_CTL allows more legitimate traffic to pass through than the other approaches. In the case of router $R2$ the AIMD, CTL and D_CTL drop 57%, 80% and 90% respectively; D_CTL allows less DDoS traffic to reach the victim.

For the record, the optimal policy performs about 94% and occurs when the routers drop 0%, 90%, 10%, 0%, 90%, 0%, 40%, 0%, 0%, 0%, 90% and 90% of their aggregate traffic respectively.

A. Outgoing	B. Router Load	C. AIMD drop	D. Router ID	E. Router Load	F. Incoming
0.51	2.24	0.67	R1	6.85	1.55
1.73					5.30
0.00	2.24	0.57	R2	5.26	0.00
2.24					5.26
0.13	2.24	0.48	R3	4.28	0.24
2.11					4.03
0.75	0.75	0.00	R4	0.75	0.75
0.00					0.00
0.00	2.24	0.24	R5	2.96	0.00
2.24					2.96
0.75	0.75	0.00	R6	0.75	0.75
0.00					0.00
0.11	2.24	0.77	R7	9.81	0.50
2.12					9.31
0.35	0.35	0.00	R8	0.35	0.35
0.00					0.00
1.50	1.50	0.00	R9	1.50	1.50
0.00					0.00
0.34	2.24	0.46	R10	4.13	0.62
1.90					3.50
0.00	2.24	0.57	R11	5.15	0.00
2.24					5.15
0.08	2.24	0.60	R12	5.56	0.19
2.16					5.37
Legitimate	Aggregate			Aggregate	Legitimate
4.51 (69.90%)	21.25 ($\in [20, 26]$)			47.35	6.50

Table 5.2: Throttles Derived Using AIMD

A. Outgoing	B. Router Load	C. CTL Drop	D. Router ID	E. Router Load	F. Incoming
1.55	6.85	0.00	R1	6.85	1.55
5.30					5.30
0.00	1.05	0.80	R2	5.26	0.00
1.05					5.26
0.15	2.57	0.40	R3	4.28	0.24
2.42					4.03
0.53	0.53	0.30	R4	0.75	0.75
0.00					0.00
0.00	0.30	0.90	R5	2.96	0.00
0.30					2.96
0.45	0.45	0.40	R6	0.75	0.75
0.00					0.00
0.20	3.93	0.60	R7	9.81	0.50
3.73					9.31
0.18	0.18	0.50	R8	0.35	0.35
0.00					0.00
1.50	1.50	0.00	R9	1.50	1.50
0.00					0.00
0.56	3.71	0.10	R10	4.13	0.62
3.15					3.50
0.00	2.58	0.50	R11	5.15	0.00
2.58					5.15
0.08	2.22	0.60	R12	5.56	0.19
2.15					5.37
Legitimate	Aggregate			Aggregate	Legitimate
5.18 (80.21%)	25.85 ($\leq$ 26)			47.35	6.50

Table 5.3: Rate Limits Learnt Using CTL

A. Outgoing	B. Router Load	C. D_CTL Drop	D. Router ID	E. Router Load	F. Incoming
1.55	6.85	0.00	R1	6.85	1.55
5.30					5.30
0.00	0.53	0.90	R2	5.26	0.00
0.53					5.26
0.07	1.28	0.70	R3	4.28	0.24
1.21					4.03
0.75	0.75	0.00	R4	0.75	0.75
0.00					0.00
0.00	1.48	0.50	R5	2.96	0.00
1.48					2.96
0.75	0.75	0.00	R6	0.75	0.75
0.00					0.00
0.05	0.98	0.90	R7	9.81	0.50
0.93					9.31
0.35	0.35	0.00	R8	0.35	0.35
0.00					0.00
1.50	1.50	0.00	R9	1.50	1.50
0.00					0.00
0.62	4.13	0.00	R10	4.13	0.62
3.50					3.50
0.00	1.03	0.80	R11	5.15	0.00
1.03					5.15
0.19	5.56	0.00	R12	5.56	0.19
5.37					5.37
Legitimate	Aggregate			Aggregate	Legitimate
5.84 (90.39%)	25.18 ($\leq$ 26)			47.35	6.50

Table 5.4: Rate Limits Learnt Using D_CTL

5.8 Summary

In this chapter we have combined a form of reward shaping called difference rewards (D) with our Coordinated Team Learning (CTL) approach to enable online learning. The difference rewards mechanism rewards an agent based on its contribution to the system's (or the team's) task by removing a large amount of the noise created by the actions of the other agents in the system (or the team). The proposed D_CTL approach has been stress tested in experiments simulating realistic conditions; experimental scenarios were large-scale and involved constant-rate and variable-rate attack dynamics, router failures, measurement noise and "meek" attackers. D_CTL has been shown to learn really fast and be highly scalable. It has been shown to be effective against DDoS attacks, and adaptive and robust to the aforementioned conditions.

It has been demonstrated that D_CTL achieves a high performance, consistently outperforming the popular AIMD approach. For instance, in the topology involving about 100 learning agents, D_CTL achieves a system performance of 90%, outperforming AIMD by 14%. We further examined the case of a more sophisticated attack strategy with variable rate dynamics, specifically a pulse attack, where similar results are obtained. Moreover, although our approach and AIMD are more useful under aggressive attackers, D_CTL achieves a 69% performance, outperforming AIMD by 17%.

We have further demonstrated that our proposed approach learns remarkably fast. It is estimated that had the proposed system been deployed, it would require a few seconds to bring the victim load below the upper boundary, less than a minute to equalise the performance of the AIMD approach and 3-7 minutes to reach its maximum value.

We have successfully demonstrated the scalability of our approach in experiments involving up to 1000 reinforcement learning agents. A very strong result is obtained as it is empirically shown that the system performance remains unaffected by the addition of new teams of learning agents. The significant improvements on learning speed and scalability lay the foundations for a potential real-world deployment.

The adaptability of our approach was demonstrated in a DDoS pulse attack scenario. The system not only learns to effectively respond to the attack, but also learns not to drop any traffic during the inactive periods of the DDoS pulses. The advantage of being adaptable has also become apparent in the scenarios involving measurement noise where the non-learning approaches required re-tuning in order to avoid stability issues.

The proposed system meets the real-world requirements of robustness to agent failures and measurement noise. Router failures do and will occur. We have shown that the proposed approach is robust to agent failures in scenarios involving up to 24% failures. The reinforcement learning agents adapt and recover to make up for the loss. Measurement noise is always present in a realistic network. The system has been shown to be robust to measurement noise; in this case, D_CTL outperforms AIMD by 17%. Robustness to both noise and agent failures has also been demonstrated.

To bridge the gap between offline and online learning we have used incremental learning, that is, when agents are initialised with prior, reasonably good behaviours and then continue the online learning process as normal. The policies learnt during offline learning in Chapter 4 were used to initialise the Q-tables of the online learning agents. It was shown that incremental learning is beneficial to the system as the learning speed is significantly improved.

CHAPTER 6

Conclusion

The final chapter presents on overview of this study and revisits the research hypothesis. It summarises the major contributions made, the deployment issues of our proposed approach and provides directions for future work on how to resolve these issues and extend our proposed approach.

6.1 Overview

Router Throttling (Yau et al. 2005) is a popular approach to defend against DDoS attacks, where the victim server sends throttle signals to a set of upstream routers to rate-limit traffic towards it. Similar techniques to throttling are implemented by network operators (Douligeris & Mitrokotsa 2004).

We revisit the central research direction or hypothesis of this study:

> Distributed reinforcement learning will create an automated and adaptive response, thus producing dynamic and robust behaviours to effectively mitigate the impact of DDoS attacks and improve recovery rates over existing state-of-the-art rate-limiting approaches.

To investigate the hypothesis we have proposed in Chapter 3 Multiagent Router Throttling, a novel approach to defend against DDoS attacks where multiple individual reinforcement learning agents are installed on a set of upstream routers and learn to rate-limit or throttle traffic towards a victim server. This constitutes the basic design of Multiagent Router Throttling. A novel characteristic of our approach is that it has a decentralised architecture and provides a decentralised coordinated response to the DDoS threat.

In Chapter 4 we have proposed Coordinated Team Learning (CTL), an extension of the basic design, to tackle the scalability challenge. CTL is based on the divide-and-conquer paradigm; it forms teams of agents and proposes the use of task decomposition and coordinated team rewards. For effective offline learning we have also proposed hierarchical team-based communication (Comm) and combined it with our approach (CTL+Comm).

In Chapter 5 we have investigated online learning. We have considered a form of reward shaping called difference rewards that tackles the multiagent credit assignment problem. We have proposed its combination with our system (D_MARL and D_CTL) to achieve the benefits of both worlds. Particularly, D_CTL achieves a significant improvement on scalability, learning speed and final performance.

Throughout the thesis we have compared our proposed approach against a baseline and a popular state-of-the-art throttling technique from the network security literature, namely, Baseline and AIMD Router Throttling respectively (Yau et al. 2005).

Our extensive empirical results show promising evidence to support the research hypothesis. We summarise our findings in the next sections.

6.2 Summary of Contributions

We present the most significant contributions made by this study in two main areas: Network Intrusion/DDoS Response and Multiagent Reinforcement Learning.

6.2.1 Contributions in Network Intrusion/DDoS Response

The proposed D_CTL approach has been stress tested in experiments simulating realistic conditions; experimental scenarios were large-scale and involved constant-rate and variable-rate attack dynamics, router failures, measurement noise and "meek" attackers. D_CTL has been shown to learn really fast and be highly scalable. It has been shown to be effective against DDoS attacks, and adaptive and robust to the aforementioned conditions. Specifically, our proposed technology improves over state-of-the-art DDoS intrusion response in the following ways.

Effective Response to DDoS Attacks

Effectiveness has two aspects; these are learning speed and final performance. The system performance is the percentage of the legitimate traffic that reached the victim server during an attack. The learning speed is critical for online learning. If the learning speed is slow, the defensive system may be practically useless to the victim, even if it eventually achieves a high performance.

It was demonstrated that D_CTL achieves a high performance, consistently outperforming AIMD. For instance, in the topology involving about 100 reinforcement learning agents, D_CTL achieves a system performance of 90%, outperforming AIMD by 14%. As we will later discuss in the presence of noise the improvement is larger.

We further examined the case of a more sophisticated attack strategy with variable rate dynamics, specifically a pulse attack, where similar results are obtained. Moreover, although our approach and AIMD are more useful under aggressive attackers, D_CTL achieves a 69% performance, outperforming AIMD by 17%. In addition, as far as the original AIMD approach is concerned, performing throttling becomes very challenging as the range $[L_s, U_s]$ becomes smaller.

If D_CTL was deployed, we have estimated that it would require a few seconds to bring the aggregate load below its upper boundary, thus allowing the victim to remain functional during the DDoS attack. Furthermore, it would require less than a minute to overcome AIMD and about 3-7 minutes to reach its maximum performance. Recall that 88% of the DDoS attacks last for less than an hour (Anstee et al. 2013). This is a powerful result; D_CTL's remarkable learning speed and scalability (discussed below) lay the foundations for a potential real-world deployment. Notice that this occurs when the agents learn from scratch.

We have further shown that when the agents' policies are initialised to reasonably good behaviours, the online learning speed improves. We call this incremental learning. Specifically, we have demonstrated that when the agents' policies are initialised to their universal policies learnt during offline learning, the proposed system starts with a similar behaviour to AIMD and online learning improves over that.

Scalable Learning-based DDoS Response

The original AIMD throttling approach has been demonstrated to be highly scalable (Yau et al. 2005). In the previous section we have discussed about the D_CTL's effectiveness to defend against DDoS attacks. However, if our proposed approach is not scalable, it may be practically useless as it will never be considered, let alone adopted, by a company or organisation.

In a study conducted by Spring et al. (2002), the authors estimate that the number of core routers in an ISP is between 11 and 1018. It is observed that the difference is 100 times larger than the smallest ISP networks. According to these figures, we have estimated that the number of defensive routers in an ISP is between 8 and 512. In fact, it is expected that this number would be less than 512 since it is unlikely that an ISP would universally support the proposed functionality.

We have successfully demonstrated the scalability of D_CTL in experiments involving up to 1000 reinforcement learning agents. A very strong result is obtained as it is empirically shown that the system performance remains unaffected by the addition of new teams of learning agents. The empirical results suggest that our proposed approach is scalable enough to be potentially deployed even in a large ISP network.

Adaptive DDoS Response

We have demonstrated in offline learning experiments that the proposed approach is more responsive to the attackers' dynamics than AIMD, since the system learns the router throttles during offline training and as a result it does not require any oscillations to bring down the aggregate load to a desired level. Recall that for the AIMD approach, performing throttling becomes challenging as the range $[L_s, U_s]$ becomes smaller.

One of the core advantages of reinforcement learning is its capability for online learning. As an example, we have demonstrated the adaptive nature of our proposed D_CTL in a DDoS pulse attack scenario. The system not only learns to effectively respond to the attack, but also learns not to drop any traffic during the inactive periods of the DDoS pulses.

One of the motivations for online learning is that non-learning and offline learning approaches require to have a reasonable knowledge of the network model and topology. If this knowledge is inaccurate, parameter re-tuning and re-training is required respectively.

Indeed, in the experiments that examined scenarios with measurement noise, the non-learning approaches suffered from stability problems. To overcome these issues we manually re-tuned the approach; specifically, we have set a lower value for L_s to allow the aggregate load to stabilise within $[L_s, U_s]$. Online learning relaxes the assumption of prior knowledge availability required, therefore D_CTL could adapt to the new situation.

Robustness to Agent Failures and Measurement Noise

We have demonstrated that the proposed system meets the real-world requirements of robustness to agent failures and measurement noise.

In real-world deployment network component failures do and will occur. Therefore, a defensive system must be robust to these failures. This is particularly the case for our approach and AIMD which are router-based. We have shown that our proposed D_CTL approach is robust to router failures. When routers fail, the remaining active learning agents adapt and recover to make up for the loss. Specifically, we have demonstrated that even when almost a quarter of the defending routers fail, the remaining active learning agents recover and achieve an impressive system performance of 75%, 11% more than AIMD.

Noise is always present in a realistic network. The proposed system has been demonstrated to be robust to measurement noise; in this case, D_CTL outperforms AIMD by 17%. Robustness to both measurement noise and agent failures has also been achieved.

Resiliency to DDoS Attacks

The original AIMD throttling approach is victim-initiated, that is, the victim controls and sends the throttle signals to the upstream routers. However, it is based on the assumption that either the victim remains operational during a DDoS attack or that a helper machine is introduced to deal with the throttle signalling (Yau et al. 2001). The first assumption can be violated in a real-world scenario. As far as the second assumption is concerned, the helper machine can also become a target of the attack.

In essence, the problem may arise because the existing throttling approach is victim-initiated i.e. it has a single point of control. In other words, although it offers a distributed response, it is still a centralised approach. Our proposed approach has a decentralised architecture as it consists of multiple reinforcement learning agents that learn to rate-limit or throttle traffic towards the victim server. It provides a decentralised coordinated response to the DDoS problem, thus being resilient to the attacks themselves.

6.2.2 Contributions in Multiagent Reinforcement Learning

Coordinated Team Learning (CTL) for Scalability and Agent Coordination

Despite the elegant theory behind temporal difference learning, multiagent reinforcement learning experiences difficulties when applied to real-world problems. There have only been a limited number of successful large-scale applications.

In this study, we have shown that our proposed CTL approach that uses task decomposition and coordinated team rewards, can significantly scale up to large scenarios. The system performance is demonstrated to remain unaffected by the addition of new teams of learning agents in experiments involving up to 1000 reinforcement learning agents.

The combination of the proposed CTL with difference rewards has been shown to be beneficial as it can better scale-up than the use of difference rewards or CTL on their own. It has also been demonstrated that their combination is much more robust to scenarios with measurement noise. Dealing with noisy environments is important since noise is always present in real-world applications.

Communication between learning agents is not always desirable. For example, it is often the case that agents in a multiagent system have different designers or owners. Furthermore, there exist domains where communication between agents is not possible at all. For instance, it is often the case where a learning agent is not aware of the other agents' existence. Furthermore communication signals can be costly, noisy, corrupted, dropped or received out of order, and mechanisms to ensure a reliable delivery are necessary. In addition to all these, communication poses a security risk and mechanisms to ensure security are needed. Decentralised coordination is a major challenge in multiagent learning and CTL provides a solution to the problem.

In this study we have viewed the DDoS problem as a resource congestion problem. Although no strong claims can be made yet, we believe that our proposed CTL approach can be used in other related multiagent domains such as congestion problems. Such examples include air and road traffic management where the goal is to reduce congestion in sectors (airspace which handles 10-40 flights on a given day) and intersections respectively.

Real-world Application of Multiagent Reinforcement Learning

As discussed earlier, mainly because of scalability issues there have only been a limited number of successful real-world applications such as playing backgammon (Tesauro 1994) and robotic soccer (Stone et al. 2005), and more recently air traffic management (Tumer & Agogino 2007) and distributed sensor networks (HolmesParker et al. 2013; Colby & Tumer 2013).

The work conducted in the thesis applies multiagent reinforcement learning to the area of network intrusion *response*, specifically DDoS response. DDoS attacks constitute a rapidly evolving threat and one of the most serious in the current Internet. While there is an extensive literature regarding the application of machine learning to network intrusion detection, to the best of our knowledge this is the first time multiagent learning is applied to intrusion response. It is worth mentioning that this is also the first time the difference rewards method is applied to a problem in the area of network security.

6.3 Limitations and Future Work

Despite promising empirical results and the contributions made, there are some deployment issues that need to be resolved in order for our approach to be applied in practice. These are outside the scope of the thesis and are discussed below. The discussion also proposes ways to resolve these issues and extend our approach.

6.3.1 Secure and Reliable Communication

All of our proposed approaches require minimum levels of communication; although note that since we are particularly interested in decentralised coordination, no communication between the learning agents exists. For example, in the MARL approach the victim communicates the global reward to the learning agents. This is encountered in almost every application of a reinforcement learning-based solution. Another example is the D_MARL approach. As discussed in Section 5.7.1, the difference rewards signal of any agent i is calculated by the agent i itself. In order for agent i to be able to do that, the victim sends to agent i all the necessary information required for the calculation of difference rewards.

In real-world deployment we need to ensure a reliable and secure delivery of the communicated information. In practice, our proposed system will require the designer to include mechanisms like message authentication, priority transmission and retransmission in case of losses. Notice that this also applies to the case of the AIMD approach in order to have a reliable and secure delivery of the throttle signals (Yau et al. 2001).

6.3.2 Enrichment of the State Space

Our approach is mostly useful under aggressive, rather than "meek" (non-aggressive) attackers. A "meek" attacker sends at a similar rate to a legitimate user, therefore, the system cannot differentiate between legitimate and attack traffic by taking into consideration only the traffic volume. Future work should investigate the enrichment of the state space of the reinforcement learning agents as this will enable a better response to the "meek" attackers case.

The enrichment of the state space does not imply homogeneity of the sensors. Each throttling router in practice may have a different number and type of sensors. In other words, reinforcement learning agents will have a different set of state features. Taking everything into account, the need for a decentralised coordination between sensors and response units becomes more apparent in real-world deployment. In this thesis we have shown that our proposed approach is a strong candidate to address this requirement.

6.3.3 Theoretical Model

Our work in this thesis is entirely empirical. An important future direction would be to derive a theoretical throttling model to be optimised. This might shed light on what the optimal policy or the optimal DDoS response is in a truly large-scale scenario. In addition, as it is empirically demonstrated in Chapter 5, the performance of the CTL (and D_CTL) approach remains almost unaffected by the addition of new learning agents in the system. It is therefore worth examining whether any theoretical properties or guarantees can explain this behaviour.

6.3.4 Model-based Reinforcement Learning

We have argued that if the detection mechanism is accurate enough to derive attack signatures then the DDoS problem can be simply solved by filtering the attack traffic. We have focussed in cases where the detection mechanism cannot precisely characterise the attack traffic i.e. when attacks signatures cannot be derived. Throughout the thesis we have applied model-free or direct reinforcement learning i.e. our goal was to learn a response against DDoS attacks involving scenarios with constant-rate and variable-rate attack dynamics such as pulse attacks.

Future work should investigate the application of model-based reinforcement learning i.e. algorithms that attempt to also learn a model of the environment apart from the optimal policy. An agent can use such a model to predict how the environment will respond to its actions i.e. given a state and an action, a model can produce a prediction of the next state and reward. This would require the integration of planning and learning. Model-based reinforcement learning constitutes a promising future direction as it might reveal new attack rate patterns and might help the human expert to better understand the problem and ultimately facilitate his task on deriving new attack signatures.

6.3.5 Human Intervention

Kantchelian et al. (2013) argue that in order for a machine learning-based security system to be of practical interest, it must engage with the human operator beyond its engineering. For instance, as discussed in Section 2.2.5, for an intrusion detection system the human should be integrated beyond feature selection and training data labelling.

There is a growing body of work integrating human advice into reinforcement learning agents to improve their learning speed (Knox & Stone 2009). The advice can be given in a variety of forms, for example, by providing action suggestions to the learning agents or by guiding them via online reinforcement. A real-life analogy is the game of soccer where players are given advice by their coach. Considering the human advice requires the integration of the human in the learning loop.

For the intrusion response domain, we argue that the human expert must be able to intervene at any time and for any reason he deems it necessary. For example, the human operator can suggest to the defensive system an action h_a perhaps after a telephone conversation with the victim under attack. Another simple example could be the following. If the human operator has any reason to believe that the environmental reward r is no longer useful, for instance if one of the detection criteria is no longer accurate, he must be able to intervene and provide human reinforcement h_r so learning agents use h_r, rather than r, as a reward signal. Future work should also investigate more complicated cases such as ways to take into consideration both r and h_r. Moreover, it should investigate how the human intervenes with the CTL approach where the defensive system needs to deal with h_a and h_r on a per team basis.

The adaptive nature of our approach could rapidly address the needs of the new situation. We believe integrating human advice into our system is so important that we deem it necessary for its real-world deployment.

References

Agogino, A. K. & Tumer, K. (2004). Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, (pp. 980–987). IEEE Computer Society.

Akchurina, N. (2009). Multiagent reinforcement learning: algorithm converging to nash equilibrium in general-sum discounted stochastic games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, (pp. 725–732). International Foundation for Autonomous Agents and Multiagent Systems.

Anderson, J. P. (1980). Computer security threat monitoring and surveillance. Technical report, Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

Anderson, R. (2008). *Security Engineering*. John Wiley & Sons.

Anstee, D., Cockburn, A., & Sockrider, G. (2013). Worldwide infrastructure security report. Technical report, Arbor Networks.

ArborNetworks (2010). The growing threat of application-layer ddos attacks. Technical report, Arbor Networks.

ArborNetworks (2014). The peakflow platform. `http://www.arbornetworks.com/products/peakflow`.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, *53*(4), 50–58.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming* (1st ed.). Athena Scientific.

Boyan, J. & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, 369–376.

Brenton, C. (2007). Egress filtering faq. Technical report.

Breslau, L., Floyd, S., Heidemann, J., Estrin, D., Fall, K., Yu, H., Xu, Y., Varadhan, K., Helmy, A., McCanne, S., et al. (2000). Advances in network simulation. *Computer*, *33*(5), 59–67.

Buşoniu, L., Babuška, R., & De Schutter, B. (2008). A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *38*(2), 156–172.

Busoniu, L., De Schutter, B., & Babuska, R. (2005). Multiagent reinforcement learning with adaptive state focus. In *BNAIC*, (pp. 35–42).

Cannady, J. (2000). Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In *Proceedings of the 23rd national information systems security conference*, (pp. 1–12).

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(3), 15.

Cisco (2011). Cisco ios netflow. `http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html`.

Claus, C. & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, (pp. 746–752).

Colby, M. & Tumer, K. (2013). Multiagent reinforcement learning in a distributed sensor network with indirect feedback. In *Proceedings of the Twelveth International Joint Conference on Autonomous Agents and Multiagent Systems*, (pp. 941–948).

De Hauwere, Y.-M., Vrancx, P., & Nowé, A. (2011). Solving delayed coordination problems in mas. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, (pp. 1115–1116). International Foundation for Autonomous Agents and Multiagent Systems.

Debar, H., Becker, M., & Siboni, D. (1992). A neural network component for an intrusion detection system. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, SP '92, (pp. 240–). IEEE Computer Society.

Devlin, S. & Kudenko, D. (2011). Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, (pp. 225–232). International Foundation for Autonomous Agents and Multiagent Systems.

Devlin, S. & Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, (pp. 433–440). International Foundation for Autonomous Agents and Multiagent Systems.

Devlin, S., Yliniemi, L., Kudenko, D., & Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, (pp. 165–172). International Foundation for Autonomous Agents and Multiagent Systems.

Douligeris, C. & Mitrokotsa, A. (2004). Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, *44*(5), 643–666.

Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.-N., Dokas, P., Kumar, V., & Srivastava, J. (2003). Detection of novel network attacks using data mining. In *Proc. of Workshop on Data Mining for Computer Security*. Citeseer.

Feng, W. & Kaiser, E. (2012). Systems and methods for protecting against denial of service attacks. US patent 8321955 B2.

Ferguson, P. (2000). Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. Technical report.

Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1996). Computer immunology. *Communications of the ACM*, *40*, 88–96.

Ghorbani, A. A., Lu, W., & Tavallaee, M. (2010). *Network Intrusion Detection and Prevention: Concepts and Techniques* (1st ed.). Springer Publishing Company, Incorporated.

Ghosh, A. K., Wanken, J., & Charron, F. (1998). Detecting anomalous and unknown intrusions against programs. In *Computer Security Applications Conference, 1998. Proceedings. 14th Annual*, (pp. 259–267). IEEE.

Greene, T. (2011). The ddos hall of shame. *NetworkWorld*.

Hansman, S. & Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, *24*(1), 31–43.

Hart, E. & Timmis, J. (2005). Application areas of ais: The past, the present and the future. In *Proc. of the 4th International Conference on Artificial Immune Systems, LNCS 3627*, (pp. 483–497). Springer.

HolmesParker, C., Agogino, A., & Tumer, K. (2013). Combining reward shaping and hierarchies for scaling to large multiagent systems. *Knowledge Engineering Review*.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report.

Hu, J. & Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, *4*, 1039–1069.

Hussain, A., Heidemann, J., & Papadopoulos, C. (2003). A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, (pp. 99–110). ACM.

Ioannidis, J. & Bellovin, S. M. (2002). Implementing pushback: Router-based defense against ddos attacks. In *Proceedings of Network and Distributed System Security Symposium*.

Issariyakul, T. & Hossain, E. (2011). *Introduction to Network Simulator NS2* (2nd ed.). Springer.

Iyengar, A. K., Srivatsa, M., & Yin, J. (2010). Protecting against denial of service attacks using trust, quality of service, personalization, and hide port messages. US patent 8250631 B2.

Jung, J., Krishnamurthy, B., & Rabinovich, M. (2002). Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of the 11th international conference on World Wide Web*, (pp. 293–304). ACM.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(12), 99 – 134.

Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., & Tygar, J. (2013). Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, (pp. 99–110). ACM.

Kephart, J. O. & White, S. R. (1993). Measuring and modeling computer virus prevalence. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, (pp. 2–15). IEEE.

Kerner, S. M. (2013). Ddos attacks: Growing, but how much? `http://www.esecurityplanet.com/network-security/ddos-attacks-growing-but-how-much.html`.

Keromytis, A. D., Misra, V., & Rubenstein, D. (2002). Sos: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, *32*(4), 61–72.

Knox, W. B. & Stone, P. (2009). Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, (pp. 9–16). ACM.

Lauer, M. & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer.

Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., & Shenker, S. (2001). Controlling high bandwidth aggregates in the network (extended version).

Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., & Shenker, S. (2002). Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, *32*(3), 62–73.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, (pp. 181–189).

Matarić, M. J. (1998). Using communication to reduce locality in distributed multi-agent learning. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Learning in DAI Systems, Gerhard Weiss, ed.*, *10*(3), 357–369.

McKeown, N. (2008). Handouts on quality of service, fairness and delay guarantees (introduction to computer networks). web.stanford.edu/class/cs244a/handouts.

Mirkovic, J., Prier, G., & Reiher, P. (2002). Attacking ddos at the source. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, (pp. 312–321). IEEE.

Mirkovic, J. & Reiher, P. L. (2004). A taxonomy of ddos attack and ddos defense mechanisms. *Computer Communication Review*, *34*(2), 39–53.

Mirkovic, J., Robinson, M., & Reiher, P. (2003). Alliance formation for ddos defense. In *Proceedings of the 2003 workshop on New security paradigms*, (pp. 11–18). ACM.

Mirkovic, J., Robinson, M., Reiher, P., & Oikonomou, G. (2005). Distributed defense against ddos attacks. Technical report, University of Delaware CIS Department Technical Report CIS-TR-2005-02.

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill series in computer science. McGraw-Hill.

Mitrokotsa, A. & Douligeris, C. (2005). Detecting denial of service attacks using emergent self-organizing maps. In *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on*, (pp. 375–380). IEEE.

Moore, D., Shannon, C., et al. (2002). Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, (pp. 273–284). ACM.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, (pp. 278–287).

Oikonomou, G., Mirkovic, J., Reiher, P., & Robinson, M. (2006). A framework for a collaborative ddos defense. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, (pp. 33–42). IEEE.

Onut, I.-V. & Ghorbani, A. A. (2007). A feature classification scheme for network intrusion detection. *IJ Network Security*, *5*(1), 1–15.

Papadopoulos, C., Hussain, A., Govindan, R., Lindell, R., & Mehringer, J. (2003). Cossack: Coordinated suppression of simultaneous attacks. In *DARPA Information Survivability Conference and Exposition,*, volume 1, (pp.$\tilde{2}$). IEEE Computer Society.

Park, K., Kim, G., & Crovella, M. (1996). On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Network Protocols, 1996. Proceedings., 1996 International Conference on*, (pp. 171–180). IEEE.

Park, K. & Lee, H. (2001). On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In *ACM SIGCOMM Computer Communication Review*, volume 31, (pp. 15–26). ACM.

Peng, T., Leckie, C., & Ramamohanarao, K. (2003). Protection from distributed denial of service attacks using history-based ip filtering. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, (pp. 482–486). IEEE.

Pfleeger, C. P. & Pfleeger, S. L. (2006). *Security in Computing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Poole, D., Mackworth, A. K., & Goebel, R. (1998). *Computational intelligence - a logical approach.* Oxford University Press.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc.

Randlov, J. & Alstrom, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, (pp. 463–471).

Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, (pp. 229–238).

Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (Second ed.). Prentice Hall.

Savage, S., Wetherall, D., Karlin, A., & Anderson, T. (2000). Practical network support for ip traceback. *ACM SIGCOMM Computer Communication Review*, *30*(4), 295–306.

Sculley, D., Otey, M. E., Pohl, M., Spitznagel, B., Hainsworth, J., & Zhou, Y. (2011). Detecting adversarial advertisements in the wild. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 274–282). ACM.

Servin, A. (2009). *Multiagent Reinforcement Learning for Intrusion Detection*. PhD thesis, Department of Computer Science, University of York, UK.

Servin, A. & Kudenko, D. (2008a). Multi-agent reinforcement learning for intrusion detection. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, volume 4865 of *LNCS*. Springer Berlin Heidelberg.

Servin, A. & Kudenko, D. (2008b). Multi-agent reinforcement learning for intrusion detection: A case study and evaluation. In R. Bergmann, G. Lindemann, S. Kirn, & M. Pchouek (Eds.), *Multiagent System Technologies*, volume 5244 of *Lecture Notes in Computer Science* (pp. 159–170). Springer Berlin Heidelberg.

Sommer, R. & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, (pp. 305–316). IEEE.

Specht, S. M. & Lee, R. B. (2004). Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *ISCA PDCS*, (pp. 543–550).

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP Flow-Based intrusion detection. *Communications Surveys & Tutorials, IEEE*, *12*(3), 343–356.

Spring, N., Mahajan, R., & Wetherall, D. (2002). Measuring isp topologies with rocketfuel. In *ACM SIGCOMM Computer Communication Review*, volume 32, (pp. 133–145). ACM.

Stallings, W. (2010). *Cryptography and Network Security: Principles and Practice* (5th ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Stone, P. (2007). Learning and multiagent reasoning for autonomous agents. In *The 20th International Joint Conference on Artificial Intelligence*, (pp. 13–30).

Stone, P., Sutton, R. S., & Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, *13*(3), 165–188.

Stone, P. & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, *8*(3), 345–383.

Sutton, R. S. & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press Cambridge, MA, USA.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 330–337).

TCPDUMP (2011). `http://www.tcpdump.org/`.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, *6*, 215–219.

Tumer, K. & Agogino, A. (2007). Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, (pp. 255). ACM.

Wang, X. & Sandholm, T. (2002). Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Advances in neural information processing systems*, (pp. 1571–1578).

Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *J. Artif. Intell. Res.(JAIR)*, *19*, 205–208.

Wolpert, David H., K. R. W. & Tumer, K. (2000). Collective intelligence for control of distributed dynamical systems. *Europhysics Letters*, *49*(6).

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (Second ed.). Wiley.

Xu, X., Sun, Y., & Huang, Z. (2007). Defending ddos attacks using hidden markov models and cooperative reinforcement learning. In *Intelligence and Security Informatics*, volume 4430 of *LNCS* (pp. 196–207). Springer Berlin Heidelberg.

Yan, J., Early, S., & Anderson, R. (2000). The xenoservice-a distributed defeat for distributed denial of service. In *Proceedings of ISW*.

Yau, D. K., Lui, J., Liang, F., & Yam, Y. (2005). Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE/ACM Transactions on Networking (TON)*, *13*(1), 29–42.

Yau, D. K. Y., Liang, F., & Lui, J. C. S. (2001). On defending against distributed denial-of-service attacks with server-centric router throttles. Technical Report CSD TR #01-008, Department of Computer Science, Purdue University.

Zetter, K. (2014). An unprecedented look at stuxnet, the worlds first digital weapon. `http://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/`.

Zhou, C. V., Leckie, C., & Karunasekera, S. (2010). A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, *29*(1), 124–140.