Multi-Agent Reinforcement Learning for Intrusion Detection

Arturo Lev Servin

Ph.D. Thesis
This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

The University of York

Artificial Intelligence Group
Department of Computer Science
United Kingdom

June 2009

Abstract

This thesis presents a novel approach to provide adaptive mechanisms to detect and categorise *Flooding-Base DoS* (FBDoS) and *Flooding-Base DDoS* (FBDDoS) attacks. These attacks are generally based on a flood of packets with the intention of overfilling key resources of the target, and today the attacks have the capability to disrupt networks of almost any size. To address this problem we propose a Multi-Agent Reinforcement Learning (MARL) approach. In Reinforcement Learning (RL) agents learn to act optimally via observations and feedback from the environment in the form of positive or negative rewards.

The thesis also investigates new methods of how to overcome some of the problems that Multi-Agent RL (MARL) faces. The proposed approach uses an architecture of distributed sensor and decision agents. Sensor agents extract network-state information. They receive only partial information about the global state of the environment and they map this local state to communication actions signals. Decision agents are located at a higher hierarchical level than sensor agents. Without any previous semantic knowledge about the signals, decision agents learn to interpret them and consequently interact with the environment. By means of this on-line process, sensor and decision agents learn the semantics of the communication action-signals. To expand our proposal to a large number of agents we deployed a hierarchical architecture composed of several levels. In this hierarchical architecture, communication signals flow from lower to higher hierarchical layers.

To evaluate our architecture with large numbers of agents and a variety of information sources we used two simulated environments and created diverse tests emulating attacks under different network conditions. We found that our approach yielded positive results in its performance levels using predefined criteria. In the network environment we evaluated the performance of our proposal versus hand-coded solutions emulating

simple misuse intrusion detection and a hybrid approach using misuse and anomaly methods. We found that our learning approach generates better results than the simple hand-coded misuse methods. Even though the hybrid hand-coded approach shows slightly better results than the learning mechanism, the main advantage of our learning method is that it does not need a designer with deep prior knowledge about the network environment.

The agent architecture and the RL for signalling approach presented in this research can be applied to domains other than IDS. Domains where this methodology could be applied are Intrusion Prevention Systems, Network Management and Quality of Service enforcement.

Contents

List of Tables

List of Figures

Parts of the present research have been previously presented or published in:

- **Multi-Agent Reinforcement Learning for Intrusion Detection**. Arturo Servin and Daniel Kudenko. In Proceedings Adaptive Learning Agents and Multi Agent Systems. Universiteit Maastricht, Maastricht, The Netherlands. April 2007.

- **Towards Traffic Anomaly via Reinforcement Learning and Data Flow**. Arturo Servin. In Proceedings 1st York Doctoral Symposium on Computing. University of York, York, United Kingdom. October 2007.

- **Multi-Agent Reinforcement Learning for Intrusion Detection**. Arturo Servin and Daniel Kudenko. Lecture Notes in Computer Science Series. Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning. Volume 4865. Springer 2008.

- **Multi-AgentReinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. In Proceedings ALAMAS+ALAg Workshop. Estoril, Portugal. March 2008.

- **Multi-AgentReinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. Frontiers in Artificial Intelligence and Applications; Vol. 178 archive. Short paper in Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence. Pages 873-874. University of Patras, Patras, Greece. July 2008

- **Multi-Agent Reinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. Lecture Notes in Computer Science Series. Proceedings of the 6th German conference on Multi-agent System Technologies. Kaiserslautern, Germany. Volume 5244. Springer 2008

- **Automatic Signature Generation via Reinforcement Learning and Data Flows**. Arturo Servin. In Proceedings CIC 2008: 17th International Conference on Computing. Mexico City. December 2008

Acknowedgments

Thanks are due to Dr. Daniel Kudenko, my supervisor for his invaluable help and advice. Without his time, guidance and wisdom this thesis would not have been possible. I am grateful to CONACYT (National Council for Science and Technology) in Mexico that granted me a scholarship to fund my doctoral studies and the people who followed my progress and assisted me with administrative procedures throughout the duration of my studies. I wish to thank my examiners Professor Ann Nowé and Professor John Clark for their advice to improve this thesis.

I am very grateful to all my family, to the ones that are here and the ones that have gone, their support along this journey has been and was invaluable. Special thanks to Alejandra for her support and help during these years. And many thanks for the little girls, Renee and Kendrah for their smiles that cheered me up.

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.

I hereby give consent for my thesis, if accepted, to be made available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Chapter 1

Introduction

Today's organisations are heavily dependent on IT infrastructure and this is a cause for considerable concern. The Internet, computer networks and information are critical assets of an organisation and their protection has increased in importance in recent years. Any attempt, successful or unsuccessful to compromise the confidentiality, integrity and availability of any information resource or the information itself is considered a security attack or an intrusion. This activity is so common today that the CERT/CC (Computer Emergency Response Team/Coordination Center) [37] decided in 2003 to stop reporting the number of computer attacks and focus on looking for new methods to measure the activity of security incidents over the Internet [37]. The increase of IT security incidents has an important economical impact as reflected by the *CSI Computer Crime & Security Survey* [145]. Since 1996 the Computer Security Institute (CSI) and the FBI have conducted this annual survey about computer security issues among organisations in the USA. In 2008 this survey reported an average loss of $289,000 USD per organisation related to IT security incidents.

Denial of Service (DoS) and Distributed Denial of Service (DDoS) Attacks are two important threats to today's computer network infrastructure. These attacks are not new and they appear in different forms. In general the attacker denies a service to the legitimate users by exhausting key resources from the target or exploiting vulnerabilities that renders the target inaccessible. According to the CSI Computer Crime & Security Survey, on 2008 21% of security incidents were either a DoS or a DDoS attack.

Special cases of DoS are the *Flooding-Base DoS* (FBDoS) and *Flooding-Base DDoS*

(FBDDoS) attacks. These are generally based on a flood of packets intended to exhaust key resources of the victim. The resource may be CPU power, memory, link availability, server connections or any combination of them. These attacks are usually performed remotely using an army of compromised hosts. The fact that they masquerade as normal traffic makes them difficult and complex to identify. Researching, designing and producing tools and mechanisms against these threats is essential because of the negative impact that they have. In their annual *Worldwide Infrastructure Security Report* [115], Arbor Networks stated that in 2008 a single DoS attack reached a peak of 40 Gbps. They also forecast that in 2009 a major attack could reach the 100 Gbps peak. This itself constitutes a major malicious degradation of bandwidth and is a major concern for ISPs and data-centres owners.

1.1 Motivation

As mentioned, the Internet along with computer networks have assumed an indispensable role in our lives. Computer networks transport vital information for us and allowing us to control key infrastructure, including financial, education and medical systems. If the operation of these networks were disrupted, it could affect our lives enormously. The economic benefits that controlling this infrastructure or accessing the information that these networks carry have caught the attention of criminal entities. One important threat to the normal operation of these networks are FBDoS and FBDDoS. Today these attacks have the capability to disrupt networks of almost any size.

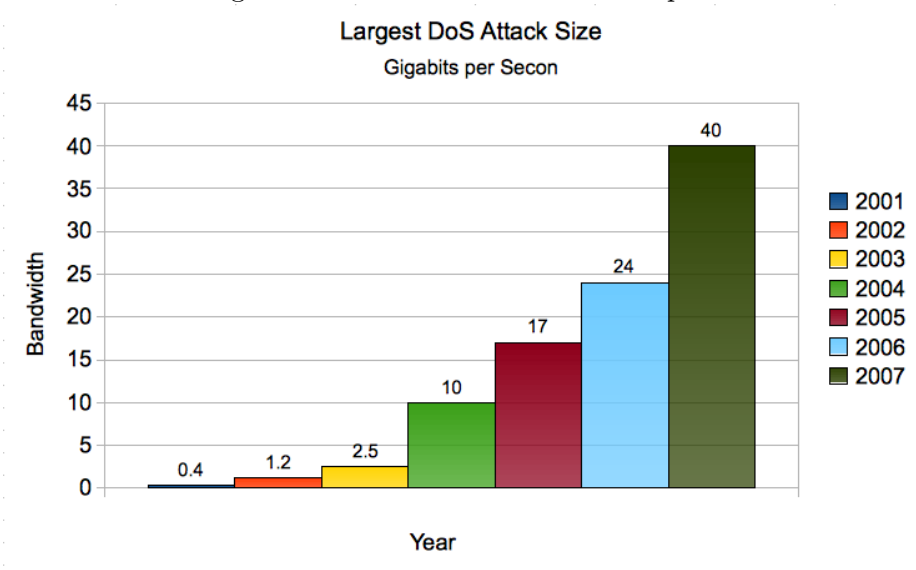
Intrusion Detection Systems (IDSs) have been used for many years to protect computer networks. Since their conception, IDSs played an important role in the protection of computer networks and information systems from intruders and attacks. Despite previous research efforts there are still areas where IDSs have not satisfied all requirements of modern computer systems. One of these areas is the detection of FBDoS and FBDDoS attacks. A major problem with such attacks is that they often possess many of the characteristics of normal traffic making them difficult to identify as malicious activity. To address this problem the use of multiple sources of data has shown promising results. However these solutions require manual and complex configuration that may not suit the scalability needs of dynamic environments.

DoS and DDoS attacks pose a latent threat for the Internet infrastructure. These attacks can easily disrupt important service infrastructure such as e-mail and web services [49]. In their 2008 report, Arbor Networks describe how the aggregate traffic generated by single DDoS attacks has increased (See Figure 1.1). According to the report, in 2008 a single DDoS reached a peak of 40 Gbps and they forecast that in 2009 the peaks could reach 100 Gbps. The report also indicates that even though ISP are using less discriminate techniques such as BGP announcements and are employing inline mitigation infrastructure, there were some attacks that only stopped because the attacker was paid. Furthermore, today it is easy to hire a botnet to generate SPAM and DDoS attacks as it has been reported by Nazario [123] in the Source Conference in 2009 and demonstrated by the BBC after hiring a botnet with 22,000 infected PCs [23]. To demonstrate how easy it is for cyber-criminals to perform their activities, the *Click* team of the BBC hired a botnet to generate SPAM and DDoS attacks. The SPAM was sent to specific e-mail accounts set up for this purpose and the DDoS attacks were targeted to a controlled server collocated in a security company's datacenter. Despite the ethical concerns related to the use of criminal services and hijacked computers that this exercise raised, it exposed how easy and cheap it is to perform these criminal activities.

Another source that shows DoS and DDoS activity is *Shadow Server* [65]. Shadow Server is a source of data regarding the behaviour and trends of DoS and DDoS attacks. Statistics from this source show an important number of DoS attacks across the Internet since 2006 as depicted in Figure 1.2. The site obtains its data from submitted log files across the whole Internet and one of its purposes is to analyse activity in *bot networks*. Bot networks are a group of compromised hosts controlled by hackers. The motivations of the attacker to use the network against a specific target vary from social or cultural beliefs (such as the sense of thrill and pleasure; recognition and power; challenge and hunger of knowledge) to criminal activities (like fraud and intellectual property or piracy) [57]. In its *Worldwide Infrastructure Security Report* [115], Arbor Networks reports that the main activities of bot networks today are sending SPAM and launching DDoS attacks.

DoS and DDoS attacks can masquerade as normal traffic and they are difficult to detect in the early stages. It has been observed that attackers choose to use traffic that looks very similar to common applications such as HTTP, e-mail and DNS [122].

Figure 1.1: DoS Bandwidth Consumption



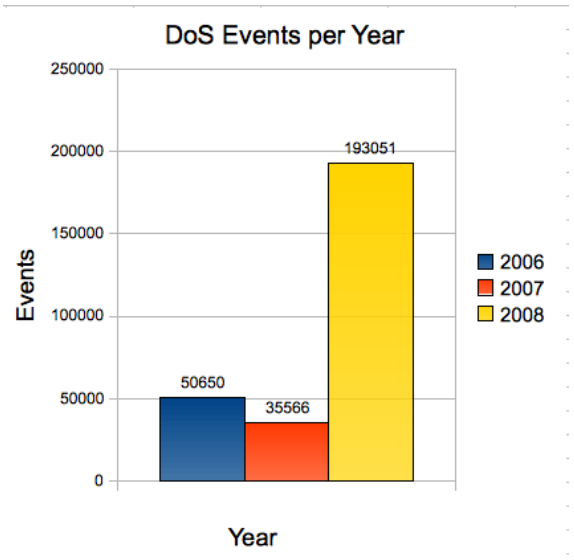
Source: Arbor Networks, Inc

Using simple detection mechanisms can lead to false positives and negatives as result of confusing normal activity with abnormal activity. The anonymity that covers these attacks is another important problem. FBDoS and FBDDoS attacks need only to flow from source to destination to succeed because they do not need a path back to the source to be launched. This allows the use of invalid IP or hijacked IP addresses space to mask the identity of the attackers [76].

Global DDoS attacks are difficult to stop because the source and control of the attack activity is distributed over the Internet. This distributed nature complicates their detection and their containment as demonstrated by the attacks launched against Estonia in May of 2007 [122, 102]. This event brought new fears of these threats because it was shown that not only small organisations were at risk, also large ISPs, web services providers and even countries may be targets of sucessful attacks.

After a thought and wide-range review of how correctly detect abnormal activity in computer networks as result of FBDoS and FDDoS we decided to formulate it as a reinforcement learning (RL) problem. We presumed that RL and Multi-agent systems

Figure 1.2: DoS Events



Source: Shadow Sever

could provide a feasible alternative to current IDSs approaches to address the problem of identifying abnormal activity within the network. RL algorithms could provide the adaptability capabilities that intrusion detection systems require to automatically detect zero-day exploits or unknown attacks without requiring complex modelling and training examples. Additionally, using distributed learning and autonomous agents we could collect and process network information from different places and in a variety of forms.

The use of RL in a Multi-Agent Systems (MAS) is generally referred as Multi-Agent Reinforcement Learning (MARL). In Reinforcement learning, agents do not require training examples or a predefined model to learn. They learn optimal behaviour from the actions that they execute and the reward that they receive from the environment. The use of these capabilities in a distributed environment, makes MARL a very interesting approach to solve interactive and complex problems. However, MARL environments face some challenges related to their scalability to a large number of agents and a large state-action space. To address these problems, some authors remark the need to research more

in the applicability of MARL to problems in real world domains [132, 21, 30]. Among these domains are Hierarchical Multi-Agent Systems Problems, Network Management and Routing. These problems became a strong motivation for us to investigate the use of MARL in the Distributed Intrusion Detection Domain. We think that this investigation can provide novel mechanisms to address some of the problems related to the action coordination and the scalability to large numbers of agents and state-space that the field of MARL faces.

1.2 Hypothesis

After discussing the problems that FBDoS and FBDDoS attacks generate and how we can construct a feasible solution to overcome these problems by means of a MARL architecture, we present the central hypothesis of this research:

> *A hierarchical architecture of Distributed Intrusion Detection Systems using RL-IDS agents is capable of detecting and categorising Flood-Based Denial of Service and Flood-Based Distributed Denial of Service Attacks at local and global scopes. This architecture is capable of detecting and categorising these attacks with high values of intrusion detection metrics similar to hand-coded approaches, but with the ability to adapt to new attack patterns.*

1.3 Novelty and Contributions

This thesis presents a MARL applied to the Distributed Intrusion Detection domain. In the proposed system architecture, autonomous agents learn to communicate and coordinate their actions by sending and receiving action-signals representing their partial observations from the environment. We have called this mechanism *RL for signalling*. Initially, the signals carry no semantics, but with time the agents learn to interpret the sent and received signals and communicate effectively. We have applied this architecture to the Intrusion Detection domain where agents try to detect intrusions in the form of normal and abnormal network states. Sensor Agents (SA) extract network state information using *tile-coding* as a function approximation technique and send signals

to Decision Agents (DA). By means of an online process and without any previous knowledge Decision Agents (DA) learn to interpret those signals to categorize different network states. To expand our proposal to a large number of agents we deployed a hierarchical architecture of agents. In this architecture, communication signals flow from lower to higher hierarchical layers. With this approach agents are able to detect Flood Based Denial of Services (DoS) and Distributed Denial of Service (DDoS) attacks in local and global scopes.

One important remark relating to the use of RL in this research concerns the use of the reward function. In this research agents are not modifying the environment directly after they perform an action. To address this problem we are providing the reward by means of a trainer, in this case a background program. This characteristic gives to our approach the appearance to use labelled examples and to function as a supervised learning mechanism. In the last chapter of this thesis we discuss and explain some mechanisms that would help to include rewards directly from the environment.

We consider that a major contribution of this research is the RL for Signalling approach used by the agent architecture to detect and categorise intrusions. The use of a hierarchical approach where local agents have detailed state information which they pass up as a summarised signal to an agent higher in the hierarchy has not been studied in detail. In order to fulfil the requirements imposed by the intrusion detection domain, this research provides novel mechanisms to extend past research on the area of MARL. We have extended the architecture to larger number of agents and states by applying some heuristics related to the number of attack states and the exploration/exploitation strategy. Also, along with the signalling mechanisms to summarise state information we have employed successfully tile coding as function approximation technique. A final contribution of our research in the area of MARL is its use to address a real world problem.

Related to the intrusion detection domain, this research presents significant contributions in the areas of adaptability, use of multiple data sources and a semi-decentralised architecture of DIDS. The proposed architecture is capable of adapting to new attacks with minimal environment analysis and modelling requirements. To increase the detection capabilities the presented approach uses multiple sources of data, which causes other problems related to the data collection and processing. These problems are ad-

dressed by means of a semi-centralised architecture of distributed intrusion detection systems. More details on these aspects are provided in Chapter 5.

1.4 Thesis Overview

This thesis is divided into nine chapters. Chapters 2, 3, 4 present the theoretical background of this thesis. The first part of Chapter 2 provides an overview of some important security threats and details the operation of FBDoS and FBDDoS attacks. In addtion, we explore how a solution using multiple sources of information could improve the detection of these attacks. The second part describes the foundations of Intrusion Detection Systems (IDS) and discusses in depth what we consider the most important research in the area. In this chapter we also discuss the use of Machine Learning in this domain and the security implications that it could have. Chapter 3 introduces the foundations of Reinforcement Learning. The first part of the chapter is the theoretical background of RL. The second part describes the challenges faced in applying RL to the Multi-Agent System domain and it explores a diverse set of approaches taken to address these difficulties. We follow with Chapter 4, in which the different methods employed to apply RL to the computer networks and to the intrusion detection domains are analysed. In this chapter we discuss our arguments about the feasibility and the advantages of the use of RL in those domains. We detail the mechanisms used to detect attacks through RL.

In Chapter 5 we outline the RL mechanism employed and referred to as Multi-Agent RL of Signalling. Multi-Agent RL of Signalling is a distributed architecture of autonomous learning agents. Using RL, agents send action-signals that are used to learn how to categorise normal and abnormal network states. The chapter details the mechanisms used by the proposed approach. It explains how it was used in a small group of agents called cell and how it was expanded to large numbers of agents with the help of a hierarchical model.

Chapter 6 describes the research methodology followed. In the chapter we discuss the difficulties in evaluating IDS and we propose two evaluation environments to deal with this problem. We explore different criteria to evaluate the results of our learning approach applied to the IDS domain and we state the set of metrics used.

Chapters 7 and 8 present the results of applying our proposed solution in two simulation environments. Chapter 7 describes and analyses a series of tests using the agent architecture in an abstract environment. The abstract environment was the initial test bed for our algorithm. We were interested in evaluating the coordination and scalability mechanisms required to develop an intrusion detection engine based on RL. The chapter describes the tests that we performed and explains the results obtained. It also establishes how those results were used to provide better and more realistic experiments in the network simulator. Chapter 8 details a more complex set of tests performed in a realistic network simulation. These tests were performed using the network simulator NS-2. They were aimed at evaluating the agent architecture and the RL for signalling algorithm under more realistic conditions than in the abstract environment. In this chapter we evaluated a single agent, small multi-agent and hierarchical multi-agent architectures. The chapter also presents results using a fully online learning approach where agents never stop learning.

The last chapter of this thesis is Chapter 9. In it we discuss our conclusions related to this research. It briefly describes the tests evaluated and it summarises the obtained results. We discuss what we consider are the most important and relevant elements of our research as well its limitations that we have found about it. Finally, the chapter and the thesis ends with the identification of areas of further work to improve the proposed IDS approach and other means of adapting the proposed agent architecture to other domains.

Chapter 2

Intrusion Detection Systems

Organisations rely critically on the Internet and computer networks, generally to communicate and to provide or obtain all manner of information services. As dependance on computer networks has grown, they have increasingly became major targets for attack. Files, e-mail, e-commerce transactions are not the only information carried by modern computers networks. They commonly transport real time information such as voice, video and control information. Intrusion Detection Systems (IDS) play an important role in the protection of computer networks and information systems from intruders and attacks. Though *Denial of Service* (DoS) and *Distributed Denial of Service* (DDoS) have been the subject of research, they still remain a problem with many issues of detection to be resolved. The emergence of botnets coupled with increasing vulnerabilities in end-user software [85] means that these attacks are increasing in number and potential for causing damage. As a consequence, research into their effective detection and management is dearly sought.

A special case of DoS are the *Flooding-Base DoS* and *Flooding-Base DDoS* attacks. These are generally based on a flood of packets with the intention of exhausting in some way the network resources of the victim. Flood-Based DoS and DDoS attacks change the normal behaviour of the network in different ways and spotting these differences could help us to detect the presence of attacks [117]. Dispite that, traffic patterns on computer networks are difficult to model. Leland and colleagues [101] showed these complexities in *local area networks* (LANs); Paxon and Floyd [134] found similar results on wide area networks (WANs) and in a later work [64] in the global Internet traffic

as well. Fowler [66] summarises these characteristics as *long-range dependence* (LRD), self-similarity, infinite variance, heavy-tailed probability distribution functions and burst patterns. For these reasons and the distributed operation of these attacks it is especially difficult to create a flexible hand-coded IDS. We believe that machine learning provides the opportunity to use a distributed and adaptable platform to tackle the problem.

This chapter is divided into *DoS and DDoS Attacks*, *Intrusion Detection Systems* and *Intrusion Detection Systems and Learning*. The first part presents an overview of a special type of computer attacks referred as Denial of Service and Distributed Denial of Service. We also discuss the requirements of the defence mechanisms aimed to offer protection against them. The section about Intrusion Detection Systems describes these devices as a more general defence against computer attacks and it provides some discussion as to how researchers have applied a diverse set of computer science techniques to them. The final section summarise the research work done to date in merging the fields of machine learning and intrusion detection.

2.1 DoS and DDoS Attacks

Denial of Service (DoS) attacks are not new and they appear in a while in different forms of intrusions.. We can find them as explicit attacks to network infrastructure [76] or as a part of self replicated code or worms. In a DoS attack, the attacker tries to exhaust key resources of the target to deny a service to the legitimate users. The resource may be CPU power, memory, server or link bandwidth, server connections or any combination of them. If the attack is launched from several sources, it is defined as a Distributed Denial of Service (DDoS) attack.

The classification of DoS and DDoS has proven to be a complex task. Mirkovic and Reiher [118] provide a taxonomy of DoS/DDoS attacks and defences.They classify attacks according to the degree of automation of the attack, the communication mechanism used by the controller to launch the attack, the vulnerability that the attack exploits, the anonymity level of the attack, the level of complexity needed to discriminate legitimate from attack activity, the level of impact of the attack, the type of victim (i.e. network, host, etc.), and finally the transmission rate of attack activity or *rate dynamics* as they called it. Douligeris and Mitrokotsa [59] focus their classification on

Distributed DoS attacks, nevertheless their conclusions are very similar to the taxonomy proposed by Mirkovic and Reiher. Douligeris and Mitrokotsa classify DDoS attacks as: Degree of automatation, exploited vulnerability, attack rate dynamics and level of impact. Hussain *et al.* [83] take a more pragmatic approach and classify DoS/DDoS attacks according to head information in attack packets, *ramp-up behavior* or attack rate dynamics and *spectral analysis*. Although this classification is not as complete as the Mirkovic-Reiher and Douligeris-Mitrokotsa it is useful to understand some DoS/DDoS behaviour and to construct defences mechanisms. Specially interesting is the proposed spectral analysis that tracks the intensity and rate of the attack.

In the following section we analyse and describe a special case of DoS: The *Flood-Base DoS* (FBDoS) and *Flood-Base DDoS* (FBDDos) attacks. These attacks are generally based on a flood of packets with the intention of exhaust the network resources of the victim.

2.1.1 Flood-Based DoS and DDoS Attacks

Flood-Based DoS and DDoS Attacks are usually performed by an attacker remotely controlling an army of compromised hosts. The effect of the attack depends on the number of compromised hosts, the available aggregate bandwidth from attackers, the victim bandwidth and the vulnerability that they are exploiting [118]. The protection against Flood-Based DoS and DDoS is a complex problem because of different factors. We can summary and explain the most challenging as:

- **Complex activity:** Many DoS and DDoS attacks can masquerade as normal traffic and to accurately identify them it is necessary to analyse different factors in the network behaviour. Using a single source of information (i.e. link utilisation) can create false positives in environments where high utilisation of some resource is expected [118]. This forces the creation of mechanisms capable of distinguishing attack activity from high use of resources by legitimate applications [76].

- **Anonymity:** FBDoS and FBDDoS attacks flow from source to destination and they do not need a path back to the source. This allows the use of non-valid or hijacked IP addresses space to hide the identity of the attackers [76].

- **Distributed control:** The source and control of the attack activity is distributed over the Internet. To stop the attack it is necessary to coordinate all the ISPs along the path of the attack. This requires a global coordination to identify and stop global Flood-Based DoS and DDoS which is very complex due to ISPs or organisation policies [120, 192].

Flood-Based DoS and DDoS attacks change the normal behaviour of the network in different ways. Some researchers [117] argue that spotting these differences could help us to detect the presence of attacks. One characteristic of FBDoS/FBDDoS is that they generally have a high rate of *unidirectional traffic*. This characteristic is the result of a large amount of packets sent to the victim from *spoofed* IP addresses and the inability of the victim to reply to any connection. As many of the *conversations* on the Internet are bidirectional [117], unidirectional activity in the form of non-responsive hosts and aggressive sending rate is an important indicator of possible malicious activity.

Another anomaly in FBDoS/FBDDoS is the presence of spoofed IP addresses. *Spoofing* is the creation of TCP/IP connections with someone else's IP address [63]. In local networks, IP address spoofing can be easily detected by comparing the source IP address with the local pool assigned. Today this functionality is well documented [63, 17] and available in most routers, firewalls and IDS. Unfortunately, as Mirkovic and Reiher point out, the functionality is not yet widely deployed. Detecting spoofed IP addresses in the victim network is highly complex because there is no mechanism in the IP protocol to verify the authenticity of the source address. Nevertheless, there is the possibility of detecting unused IP address space, IP address space from private pools [143] and the *black lists* of recent attack activity [86].

Other anomalies generated by FBDos/FBDDoS are in the flow of traffic within the network. In normal Internet flows, TCP connections present a constant flow of acknowledgements (ACK). When any of the parties involved in the TCP connection detect lost acknowledgements it assumes that there is congestion in the communication channel. As a consequence, the flow control mechanism of TCP slows the transmission rate. ICMP traffic under normal conditions has similar characteristics as TCP. Furthermore, the send/receive ratio must be under certain maximum values. UDP is harder to profile because applications using this protocol do not always require acknowledgement of the received packets. Although difficult, it is possible to profile some UDP applications. For

example *Voice over IP* (VoIP) and multimedia applications use the *Real-time Transport Protocol* (RTP) [151]. RTP packets include payload-type identification, sequence numbering, and time stamping to calculate delays/jitter which can be helpful to identify anomalies when packet loss or high delay/jitter occurs. Other applications that offer similar characteristics are *Domain Name System* (DNS), *Network Time Protocol* (NTP), *Network File System* (NFS), *Session Initiation Protocol* (SIP) and routing protocols such as *Routing Information Protocol* (RIP) and *Open Short Path* First (OSPF).

Later in this document we will discuss how some researchers have tried to tackle the problem of identifying and containing DoS/DDoS using a variety of methods from simple statistical data analysis using a single source of information to machine learning algorithms merging data from several sources. We will also analyse how computer worms generate a DoS attack as collateral damage of their spreading and how after they are inserted into hosts, these can be used to launch DoS/DDoS attacks.

2.1.2 Worms

Worms are programs that self-propagate across the Internet by exploiting security flaws in widely-used services [164]. Contrary to viruses, worms do not need an external mechanism to spread [184]. We are especially interested in worms for various reasons. One of them is the collateral damage that they produce while they are in the distribution state or spreading. In this stage they may cause a Denial of Service as a result of the high volume of traffic while propagating. Another important factor is that worms are the point of entry for hackers to control hosts. Through this control they may be able to perform DoS attacks and access any sensitive information to sell, disrupt or corrupt it. This could even give the attacker full control of an organisation's network [164].

The life cycle of worms is explained by Staniford *et al.* [164] and Weaver *et al.* [184]. The former analyses the behaviour of worms and how they can be modified to spread faster and more efficiently. The latter presents a taxonomy of computer worms based on their behaviour and the techniques that they use to accomplish their goal. It is also intended as an initial guide for the construction of possible defences. The states of the worm's life cycle inferred from those works are *Discovery*, *Carrier*, *Activation* and *Attack*.

In the Discovery state, the infected host starts searching for new targets using a

variety of methods. A simple mechanism is to search randomly for new targets. To accelerate the infection rate the infected host can have pre-coded lists of possible targets. They can passively monitor for target information or they can look for local data inside the infected hosts. For example some techniques use the victim's hosts file, the ARP cache or its open connections. Complex implementations are able to use several mechanisms and task sharing to accelerate the spreading rate. When the worm finds a vulnerable target they try to infect it, and if it does, the Carrier state will begin. To infect a target, the attacker needs to insert code onto the vulnerable host. There are different ways to accomplish this action. The worm can be self-carried; this means that the infected code is embedded in the infection process as a payload. An alternative is to command the vulnerable host to download the infection code through a second channel (i.e. TFTP, HTTP, FTP, IRC, etc.). The next state is Activation. Some worms are immediately active after the infection, but others wait for human activation, a scheduled activation at a specific date, or they may be activated by an event on the infected host (i.e. the launching of a specific application). Attack is the last state and the primary objective of the worm. The attack can be a DoS attack, sending SPAM or the stealing and/or corruption of information.

We are interested in understanding the life cycle of worms because through it we can design strategies to model this behaviour, to identify them and distinguish them from other types of faults or normal heavy traffic. Worms may appear to network management systems as faults or heavy traffic flows, but never as an attack because these systems do not have the capabilities to differentiate them. To security devices such as firewalls they may look like heavy but normal traffic if these attacks are directed to common applications such as the http or e-mail infrastructure. Intrusion detection systems cannot distinguish when a worm is impacting the performance of the system and they cannot generate the proper alarms to get the attention of the security managers. We think that in order to identify and to defend from complex threats such as worms it is necessary to merge information, technologies and strategies from different disciplines. This would also allow us to construct a global state of the network.

2.1.3 Attack Tools and Techniques

Denial of Service and Worms are just a small fraction of computer attacks. A survey of every specific security attack is beyond of the scope of this document. Nevertheless, to design efficient defences against security attacks it is important to understand how they work. Below there is a brief description of some security attacks and how the attackers use various tools and techniques to compromise the confidentiality, integrity and availability of the information.

Reconnaissance tools allow the attacker to profile the target. Through network scanners the attacker can verify open services in the target and then map them to known vulnerabilities. Intrusion Detection Systems (IDS) can easily detect scanning, but the problem is that worm scan activity is so common today that IT professionals find it very hard to tune IDS to differentiate irrelevant worm activity from reconnaissance hacking activity [136, 193, 80, 104]. Also, if the attacker performs scanning cautiously, the scanning can be indistinguishable from normal activity [105, 95, 156].

Packet sniffers allow the attacker to capture data from the network, this is also known as eavesdropping. The attacker can see, change or even destroy the data. Sniffers are very useful in shared networks such as the old Ethernet hub and many modern wireless networks. The use of LAN switches instead of Ethernet hubs and the deployment of VLANs minimize such types of attacks. Nevertheless, techniques such as *arpspoofing*, *arp poisoning* and *802.1q tagging* attacks allow the use of sniffers even in LAN switches with VLANS [36]. Arpspoofing and ARP poisoning refers to the change by an attacker of the TCP/IP ARP packet to disguise it as a legal host, for example as the subnet gateway. 802.1q tagging attacks falsify VLAN tagging information to trick LAN switches. In this attack switches may forward packets belonging to other VLANs to the attacker. A partial defence against sniffers is the use of encrypted information [10]. This helps to protect against the confidentiality and integrity of the information but only partially solves the availability problem.

In *masquerade attacks*, the attacker pretends to be another entity or host. To perform them, the attacker can use tools to spoof address information. Firewall Packet filtering and LAN switch ARP analysis are defence mechanisms against address spoofing. The use of IDS also allows the detection of address spoofing events. Other means to impersonalize identities are the *man in the middle attack* and *session hijacking*. In

these attacks the attacker takes a position between the hosts interchanging information. If the attacker can get between two hosts before they start a session, the attacker can even steal encrypted information of weak encryption protocols [117].

Authorisation is a critical element to assure the confidentiality, integrity and availability of the information. Authorisation is based on access control and authentication. While access control can be compromised with address spoofing and man in the middle attacks, authentication may be compromised by password crackers. If authentication is based on passwords, the attacker can use brute force attacks to discover weak passwords on the target. Another option for the attacker is to use dictionary attacks. These attacks use dictionary words to discover weak passwords, some of them may use several dictionaries in different languages. Most sophisticated password crackers use a combination of dictionary and brute force attacks [69]. Just like reconnaissance tools, passwords crackers can be recognised by IDS because of their activity, however with care and patience attackers can bypass detection controls [117].

Buffer overflows are a very common vulnerability for remote attacks. They are flaws in the code that allow the attacker to insert malicious code in the target. The code injected will have the same privileges as the code compromised [50]. Buffers are a space of memory reserved to allocate data and buffer overflows occur when a buffer is overfilled and the extra data take up the space reserved to execute some code [135]. Similar attacks are web hacks [57], these attacks exploit flaws in web severs code used to generate dynamic content. Web hacks are not necessarily complex and basically exploit sites with low levels of security. The risk of buffer overflows and web hacks are reduced with adequate software maintenance and IDS/IPS detection. IDS/IPSs detect buffer overflows and web hacks using signatures of the attacks. These attacks are based on a pattern of bytes that overflow a buffer or by specific URLs to access non-authorised information, so it is relatively easy for IDS/IPS to detect them. Firewall techniques such as packet filtering with URL reconnaissance are also helpful. Still the real challenge in the detection of these attacks is when new and unknown buffer overflows or web hacks are discovered (zero-day attacks). Moore *et al.* [120] simulated the spreading of computer worms exploiting zero-day attacks and how they can infected a large number of hosts in short time. Due to the fact that the vulnerability is not previously known, there is no signature to compare. Because of these attacks use normal connections and

applications; so, it is almost impossible to distinguish them from normal traffic.

Viruses are well known in the security field. They pass malicious code to other applications or programs generating unanticipated or undesired effects on them [135]. Examples of malicious code are *trojan horses*, *logic bombs* and *spyware*. Trojan horses appear to be valid programs that execute authorised and desirable actions but in reality they are performing non-authorised actions [124]. A logic bomb is a trojan horse that has the ability to destroy objects and it can be activated on a specific date or when an event occurs [129]. Spyware collects information about the target without the consent of the user and sends it to the attacker. These types of code are normally detected by anti-virus engines and recently spyware has also been detected by IDS/IPS through behavioural analysis. Today, client applications can download and execute code from servers. The problem with this new model is the integrity of the code sent. If there are no mechanisms to verify the integrity of the code, the attacker can inject remotely malicious code to the target.

2.1.4 Defences

Securing information is not an easy task. Researchers and IT professionals have been working to provide environments and tools that allow a flexible flow of information without compromising security. The problem is so complex that there is not a straight forward answer or a single application to solve it. To provide secure environments it is necessary to use several applications, devices and procedures, each of which is focused on one part of the problem. We cannot analyse the whole range of possible defences against security attacks and we will focus on the defences that we think are relevant to our research. We have categorised these defences as *Security Policies*, *Access Control*, *Filtering* and *Intrusion Detection*.

Security policies are a set of rules that specify the expected behaviour of users, system administrators and managers. They authorise people to do things or access information and they set the consequences of violating policies. They define the procedures to protect information and specify which threats need to be eliminated, which information resources to protect and the cost of securing the information [121]

Access control ensures that information resources are only accessed by users entitled to do so [135]. Access control models vary depending on the security approach. Some

techniques are discretionary access control, mandatory access control, role based access control, rule set access control, list based access control and token based access control. Access control is based on authentication. Examples of techniques to protect access control are strong password enforcement, periodical password change, digital identity keys, hardware identity keys and biometrics.

Filtering is a technique to *enforce* access control at the network or application level. Packet filtering can be done by access control lists in routers. This type of filtering is stateless, meaning that it cannot track information about connections. To overcome this problem firewalls are based on *stateful* filtering. Stateful firewalls track layer 3 (IP) and layer 4 (TCP) information to allow or deny connections, in other words they track the state of connections. Nevertheless, some attacks can use TCP/UDP ports open to normal traffic, for example port 80 of HTTP. Application firewalls or proxies solve this problem but at the expense of a trade-off in speed. A solution in the middle between firewalls and proxies is *stateful inspection firewalls*. Stateful inspection firewalls are basically stateful firewalls but they inspect information of well known protocols such as HTTP, SMTP, DNS, etc. to ensure that the connection is not an attack [48]. Firewalls are able to work with content filtering engines to allow or deny the access of certain content. Examples of this content are URLs, media files, file-sharing applications, etc. Finally, firewalls can work with IDS to provide an Intrusion-Prevention solution where the IDS detects attacks and signals the firewall to filter the offending connections.

According to Heady *et al.*[72], *Intrusion* is defined as the set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource. Denning [56] defines *Intrusion Detection* as a system or a methodology that aims to detect a wide range of security violations from attempted break-ins by outsiders to systems penetration and abuses by insiders. Intrusion Detection is performed by *Intrusion Detection Systems*. We will discuss them further in this document. Intrusion Detection Systems have been criticised as a passive and reactive security tool due to the fact that the action that they mostly take is just to raise an alarm [141]. To address these disadvantages, *Intrusion Prevention Systems* (IPS) is another type of defence that has been under research. IPS are devices capable of detecting and preventing attacks using techniques of anomaly detection and rule matching. Some experts [137] consider IPS as a natural evolution from IDS and firewalls; in fact, tight coupling of firewalls and IDS have been

proposed by Zang [195]. A more ambitious approach proposes that IPS would be not just composed of network firewalls and IDS, but also with anti-virus and application firewalls [48].

2.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) have been important elements in secure computer network architectures. IDSs have been analyzed and implemented using a variety of approaches and techniques from different fields of computer science. IDS monitor hosts or networks searching for abnormal or non permited activity. When IDS find attack activity, they record the event and they may perform defensive actions.

2.2.1 Background

IDS were defined first by Anderson [6] as a "Computer Security Threat Monitoring and Surveillance System", but it was Dorothy Denning's [57] work that really marked the beginning of research on IDS [14]. Denning defines IDS as an expert system capable of detecting security violations originating from outside break-in attempts or by abuse from inside users. The framework of her work describes the use of statistical methods to detect the intrusions and abnormal user patterns.

There are two basic categories of IDS: *anomaly IDS* and *misuse IDS*. Anomaly IDS states that intrusions are deviations of normal traffic. These systems create profiles of different variables over time to get a usage pattern [18]. A significant difference between pattern and current activity triggers an alarm. In other words, anything outside normal is considered an intrusion. Generally, anomaly IDS uses statistical methods but more recent research use machine learning to find patterns in normal and abnormal activity. Anomaly IDS are capable of detecting unknown attacks [125], but they normally have also a high rate of *false alarms*. Another disadvantage of anomaly IDS is that they can be trained by an attacker to accept malicious activity and consider it as normal [57, 20].

Misuse IDSs on the other hand do not learn anomalous behaviour. The pattern of a particular intrusion is hard coded as a rule or signature [14]. Misuse IDS compare current activity with predefined rules and, if they find a match, they execute an action such as triggering an alarm. That is, contrary to Anomaly IDS, anything not in the

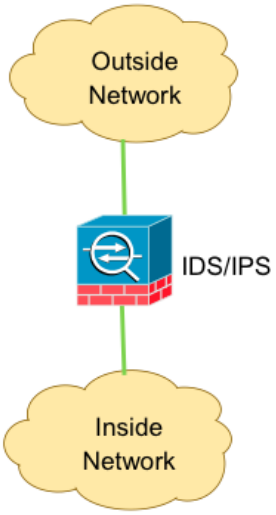
rules is normal. They are considered fast [141], reliable and typically easy to implement [146]. The biggest disadvantage for misuse IDS is that they are unable to detect new attacks. Other problems are that the accuracy of the rules depends on the designer and slight differences in attacks are not detected.

Another functional division on Intrusion Detection are *Host IDS* (HIDS) and *Network IDS* (NIDS). Host IDS work inside a host by analysing logs files, memory, disk and network utilization [112]. HIDS search for anomalous or abnormal patterns in the current behaviour of the host. Because they reside inside the host, they are even capable of analysing encrypted information. They are network-topology independent and suitable for working transparently in complex network environments. This characteristic makes them very effective in detecting attacks from insiders which may be overlooked by network security devices such as firewalls [175]. The disadvantages of Host IDS are they are as good as the log files are [56], there is extra processing in the host due to IDS tasks and they are OS dependant.

Network IDS can be connected in the network between the protected hosts (inside network) and the non-trusted network (outside network) as shown in Figure 2.1. They examine the entire network traffic through both networks and search for patterns that match attacks or intrusions. The first NIDS is described as a Network Security Monitor [73]. The proposed architecture and operation of this system are the foundation of any modern NIDS. In this research Heberlein *et al.* [73] proposed a *packet catcher* that captured traffic off the network in real time, a *parser* that extracted information from the different protocol layers, a *matrix generator* that holds information of the 4-tuple $< source, destination, service, connectionID >$, a *matrix analyzer* that used a *hybrid approach* between anomalous and misuse IDS to identify attacks and a *matrix archiver* that wrote information to disk.

In practice, with just one NIDS it is possible to protect several hosts with heterogeneous operative systems. This characteristic makes this approach easy and cheap to deploy. Other advantages are that they do not produce extra processing in the host and it is more difficult for an attacker to remove evidence from them [175]. The big disadvantage is that they are network topology dependent which makes them difficult to deploy in complex network environments. As depicted again in Figure 2.2 they can be deployed in-line or out-line. In in-line all the network traffic passes through the IDS,

Figure 2.1: Network Intrusion Detection System



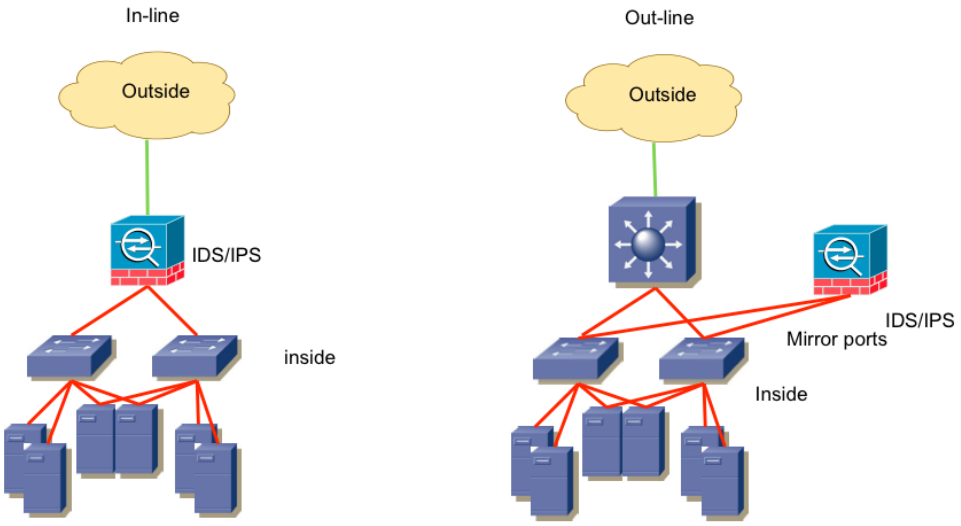
the advantage of this topology is that the IDS can perform defensive actions to filter the attack. The downside is that any problem in the IDS affects the connection of the inside to the outside network. In out-line, the IDS is transparent to the network infrastructure. The monitored port is mirrored from the network to the IDS. The advantage is that any failure or overload of the IDS does not affect the network operation. The disadvantage is that the device can only monitor the network and defensive actions are restricted.

Distributed Intrusion Detection Systems (DIDS) are groups of IDS or sensors coordinated to detect anomalies or intrusions [19]. The system can be homogeneous with every sensor of the same type or heterogeneous with a mixture of types. The architecture of DIDS varies from central control and management [161], hierarchical central management and clustering [181] to peer to peer without central management [87]. We will discuss DIDS in the next section.

2.2.2 Distributed Intrusion Detection Systems

There are several reasons that support the distribution of Intrusion Detection Systems. Computer networks are distributed systems that allow sharing resources. Modern net-

Figure 2.2: NIDS: In-line vs. Out-line



works are also complex architectures comprising hundreds and thousands of devices. Users and information are geographically and topologically dispersed around the network. This makes it almost impossible to connect a single Network IDS to monitor all the activity within it. The processing capacity and scalability of the IDS is another reason to employ a distributed system. Current IDS can handle throughput in gigabits per second, but in the past increasing throughput was one of the primary reasons to deploy a distributed environment [161]. Furthermore, in terms of security and reliability, a single IDS is also a single point of failure.

In large computer data centres, it is not hard to find several intrusion detection systems running on severs on the form of HIDS or analyzing network information as NIDS. The management of this huge amount of sensors and the data generated is another important reason to look for coordination strategies. Different approaches varying from central processing [161, 194, 60, 45] to full autonomous agents [87, 104, 81] have been used to tackle this problem. Finally, some authors [125, 22, 112, 117] argue that to effectively and accurately detect abnormal and malicious activity it is necessary to use information from several sources and not just from intrusion detection systems.

Some of the earlier works on DIDS [161] aimed to distribute the intrusion detection

task to several homogeneous agents to solve the problem of processing high amounts of data and to split the sensor activity in multiple locations. This system used a central IDS to collect all the information collected by the remote sensors. The central agent also used an expert system based on misuse detection to search for intrusions. Another early work was NIDES [5]. NIDES collected data from several IDS to a central agent that used statistical models to detect intrusions. These systems were just information collectors with a central processor.

Although Distributed Intrusion Detection Systems with homogeneous sensors have simplified management activities, in large scale IDS implementations some researchers have pointed out that they may not be able to identify complex attacks [22, 112]. An alternative is to use data from a variety of sensor types. The information from these heterogeneous sensors may be in the form of logs from hosts; firewalls and network equipment; network traffic information such as *netflow data*, *Simple Network Management Protocol* (SNMP) from network management systems and network devices, or *Remote Network MONitoring* (RMON). Netflow [44] is a protocol defined by Cisco Systems and today is de-facto standard to monitor IP flows of data. A flow is a group of IP packets that are identified by the same fingerprint, the information in this finger print includes a set of attributes: IP source address, IP destination address, source port, destination port, Layer 3 protocol type, Class of Service, router or switch interface. To be considered part of the same flow, the packets must share the same source/destination IP address, source/destination ports, protocol interface and class of service. On TCP/IP networks the SNMP is used to monitor and to manage networks. Its definition includes a set of standards, an application layer protocol, a database definition and a set of data objects [163]. In order to know which data to retrieve or write, SNMP defines the Management Information Bases (MIB). The MIBs define the structure of the management data of the monitored device. An Important MIB is the MIB-2 that defines a standard set of managed object definitions such as TCP data, interface utilisation, error rates and system uptime among others. RMON (also known as RMON1) is a monitoring protocol for Local Area Networks (LANs). It defines groups to monitor layer 2 real-time statistics such as utilisation, bytes received, mac addresses of host sending data, and more. The RMON2 is similar to RMON1 but it processes information in the application layer.

Related to the richness of a multi-source information analysis, Barford and colleagues

[19] state some reasons that explain why sensor diversity can increase the reliability of intrusion detection. Different sensors receive different information about the network. In this context, if one sensor cannot detect an attack, another with different information may do it. This correlation of events that occur at different times, places and collected by different sensors can improve the detection of intrusions because of their varied information. Different approaches have been taken to address the problem of how to merge a variety of information to detect attacks. Zhang *et. al* [194] integrate IDS and SNMP data. In their research a misuse IDS pre-processes intrusion information and after it detects an attack, it uses SNMP stored data to check for possible false positives. This implementation may reduce the number of false positives, but it fails to detect false negatives missed by misuse IDS. Lai, Kuo and Hsieh [98] use Netflow information to infer DoS and worm activity. They send the information collected from different router sensors along their network to a central system that identifies abnormal traffic patterns. They use signatures of IP spoofing use and statistical analysis to detect malicious activity. More complex architectures use more diverse types of sensor information. An example is *EMERALD Event Monitoring Enabling Responses to Anomalous Live Disturbances* [125]. EMERALD is the evolution of NIDES, a DIDS that we previously described. It uses a variety of sensors or service monitors that report to a central engine that performs data correlation and pattern matching. The information received from service monitors changes in abstraction and according to the sensor location in the network. Bass [22] proposes a model of how several sensors should interact to provide distributed coordination fusion of data. This model proposes an architecture to collect low level data from sources and then transforms it into high level information to present to network managers. Bartford *et al.* [19] expand their work on DIDS to a new detection engine that uses different types of sensors. Their work address how to combine data from multiple and heterogeneous sensors and the specification of which data is required to be sent.

Today there are a variety of commercial DIDS using the basic principles of collecting remote information for further central processing. One of them is Dshield [60]. Dshield relies on volunteer data that is submitted from across the Internet. Thousands of sensors such as firewalls, intrusion detection systems, home broadband devices, and nearly any operating systems are constantly collecting information about unwanted traffic arriving

from the Internet. These devices feed the Dshield database where the information is processed manually and automatically to look for abnormal trends and behaviour. The resulting analysis is posted to the main web page of the Internet Storm Center (ISC) [86]. There it can be automatically retrieved by simple scripts or can be viewed in near real time by any Internet user. The Active Threat Level Analysis System (ATLAS) [11] is an Internet scoped threat analysis network. It collectively examines the data traversing disparate networks to develop a globally scoped view of malicious activity traversing the Internet. ATLAS provides a public portal that summarises views of collected information. It is categorised as host/port scanning activity, zero-day exploits and worm propagation, security events, vulnerability disclosures and dynamic botnet and phishing events. Another commercial approach with central processing is the *Cisco Security Monitoring, Analysis, and Response System* (MARS) [45]. It collects data from a heterogeneity of network devices and host applications. It provides an end-to-end topological view of the network, that helps improve threat identification, mitigation responses, and compliance. It uses a hybrid approach based on anomaly, behavioural and misuse intrusion detection.

One important problem of central processing DIDS is system scalability. Adding more sources increases the amount of information that has to be sent to the central repository. Under heavy traffic conditions it may be infeasible to send all the sensor information to a central location. Furthermore the amount of time and processing power required on the central facility can make the system impractical to apply to large networks. To address the scalability problem researchers have taken different approaches, one is scaling the system using hierarchical levels. GrIDS [165] is a graphical DIDS using the hierarchical approach where the attacks take the form of graphs. To form the graphs several sensors send information to a graph controller. In turn, several graph controllers send data to a higher hierarchical controller. The research of *Architecture for Intrusion Detection using Autonomous Agents* (AAFID) [18] is a DIDS using autonomous agents. It has tree types of agents, *sensor*, *transceivers* and *monitors*. Sensors are hosts and network IDS that monitor the network and execute some intrusion detection. Sensors do not communicate with each other but send information to transceivers. Transceivers perform monitoring and control activities. The higher levels in the hierarchy are monitors that perform similar activities to transceivers but focus on global

scopes.

The *Distributed Overlay for Monitoring InterNet Outbreaks* (DOMINO) [192] is another research project in this area. DOMINO is composed of autonomous and heterogeneous NIDS agents that share information about intrusions. The system is a hierarchical architecture where information about the state of the network is held and summarised at different levels. The authors also define the communication protocol between entities in order to query and exchange information in a secure fashion. To detect intrusions, each node uses a hybrid approach of misuse and anomaly IDS. Sterne *et al.* [166] applied DIDS to *Mobile Ad-hoc Networks* (MANET). The MANET domain poses specific challenges because of the mobility of agents. To adapt DIDS to this scenario, the authors propose a *dynamic hierarchy*. To construct the hierarchy they use *clustering* techniques found in mobile networks where nodes select their peers in the cluster based on proximity and available bandwidth. Clustering selection occurs at different levels: local and regional.

A more radical approach is to design the DIDS based on *cooperative multi-agent systems* (MAS) [169] techniques with full autonomous sensors. In this architecture sensors are independent but they share information to perform intrusion detection actions. Indra [87] is a peer to peer IDS that uses rule-matching technique to detect attacks. Each agent is independent of other agents but when one of them detects an intrusion it broadcasts a signal through a secure channel. The signal contains information about the attacker. Another DIDS that uses a Peer-2-Peer architecture is *Peer- to-Peer Information Exchange and RetrievaI* (PIER) [82, 81]. It uses *Distributed Hash Table* (DHT) technologies to detect intrusion at global scales. DHT provides each node and object with a distributed hash table. Each PIER node shares and queries for attack fingerprint information. The architecture also provides SQL alike language to query for intrusion information. Similarly to PIER, Li and colleagues [104] present a DIDS based on DHT. In this architecture autonomous sensors categorise network information looking for diverse sets of attacks such a DDoS attacks, port scanning activity, virus, worms and bot activity. When attack activity is detected, each sensor communicates it to a *Sensor Fusion Center* node (SFC) in the form of an *object*. The SFC creates a hash with the object. The hash is then used to group alarms related to the same intrusion.

2.2.3 DoS and DDoS Detection and Containment

On the subject of defences against DoS attacks, Householder and colleagues [76] recommend different strategies to defend against them. However, this approach is focused on operational measures in computer networks and it does not propose any new technology. Cabrera *et. al* [32, 31] and Qin *et. al* [140] use SNMP and the variables provided by the MIB-II [114] set to detect DoS/DDoS attacks. Cabrera and colleagues analyzed which SNMP variables are useful to track anomalies generated by the attacks, they concluded that icmpInEchoReps, tcpInErrs, tcpInSegs, udpInErrors, udpInDatagrams and udpOutDatagrams are the most important variables to track. Another important conclusion by Cabrera, Qin and colleagues is that relying solely on SNMP information to track attacks generates a high rate of false positives.

D-WARD is a denial of service attack detection and containment work by Mirkovic and Reiher [117]. D-WARD sensors' autonomously detect and filter attack traffic from the closest point to its source using *semantic traffic differentiation*. The sensor engines are installed within in-line devices such as routers where they monitor and collect connections statistics and incoming and outgoing traffic. The information collected is used to generate profiles of suspicious activity. Vectors used to profile applications are the unidirectional traffic, IP spoofing, non-responsive hosts and aggressive sending rates. After detecting anomalous activity, D-WARD sensors rate-limit the offending traffic using a throttling down mechanism. The effect is the decrease of the bandwidth available for the attacker. The authors make these important contributions in their work:

1. They proposed vectors to detect anomalies generated by DoS/DDos attacks in Internet traffic.

2. They confirm that the best place to filter these malicious activities is as close as possible the source.

3. They propose a throttle mechanism to rate-limit the attack traffic.

Rate-limiting or *throttle* mechanisms have been applied by other researchers as a defence against DoS/DDoS attacks [9, 191, 38, 186] and computer worms [120, 98, 42]. Atighetchi *et al.* [9] developed a network architecture aimed to defend against a diverse range of attacks, this work is similar to EMERALD [125]. The architecture is made

of several defence layers and devices. One of the mechanisms to control attacks is the rate-limit of malicious activity. Yau *et al.* [191] propose an algorithm called *max-min fairness* that statistically measures if the traffic received by the protected server is in the range $L_S \leq r \leq U_S$. If the traffic is above the limit, the server triggers a command where upstream routers start rate-limiting the source traffic from the offending sources. Although this work presents good results and it helps to reduce the overload generated to the service, authors noted that the detection mechanisms fail to accurately separate attack traffic from normal high load. A very similar approach was followed by Chan *et al.* [38]. They triggered a rate-limit mechanism when routers found heavy traffic going to the same destination. However, this approach has the same flaws as max-min fairness [191] alternative. Wong and colleagues [186] present a rate-limit mechanism based on Domain Name System (DNS). They based their hypothesis on the fact that worm programs induce a visible difference on DNS statistics compared to normal activity. They show interesting results but we considered that DNS information cannot be used as the sole source of data because the attacker can easily perform its attacks by only using IP address information without name resolution.

Moore *et al.* propose a set of requirements [120] for containing self-propagating code, also know as worms. They propose *Prevention*, *Treatment* and *Containment*. To evaluate their work, the authors modeled worm spreading using the mathematical foundations regulating the spread of infections. They evaluate several scenarios where the detection mechanism was based on a set of rules applied to contain the worm spreading. These rules include software patches to fix the exploited vulnerability, anti-virus or IDS signatures updates and network filtering among others. This would be analogous to the application of vaccines when there is an infectious disease spreading. The scenarios were: i) the whole Internet users have applied the set of rules and the rules are available immediatly from the beginning of the worm spreading, ii) the whole Internet users have applied the set of rules that is available after some specific period of time, and iii) a more realistic scenario where only specific percentages of the Internet apply the set of rules and the rules are available after some time. Not surprisingly, they conclude the more amounts of entities deploying the rules and the least time to have them available is better to stop the worm spreading. This study also backup the hypothesis that filtering malicious activity as close as the source produces better

containtment results.

Lai, Kuo and Hsieh [98] also present research about defence mechanisms against worms and propagating malicious code. They applied a hybrid approach that evaluates connections using misuse IDS techniques and statistical methods to find suspicious activity using IP address information, protocol and the number of flows. Sensors are distributed over the network and they collect netflow data that is sent to a local repository for processing. Chen and Tang [42] propose an architecture to slow down and filter the malicious activity generated by worms. They modeled the worm behaviour through probability models using a *connection failure* approach. They based this on the supposition that worms scanning and trying to infect other nodes are not very smart and they will try to connect to non-existent hosts. This produces a significantly high number of connection failure activity characterised by specific transmission patterns of TCP SYC, TCP RESET and ICMP packets. Tracking this information allowed the researchers to identify worm activity. They too used a rate-limit approach to filter DoS/DDoS traffic.

Aime *et al.* [4] applied IDS to detect attacks in wireless networks. They focused their work on Wi-Fi networks where each host is an active monitor. Agents (hosts) collect and share relevant information about the state of the network and their connections. Each agent uses a misuse IDS technique to analyse packet information at the MAC level as well as counters related to the wireless transmission.

It is especially difficult to create a flexible hand-coded IDS for Flood-Based DOS and Flood-Based DDOS attacks. Factors such as the requirement of a variety of sources, a diverse number of parameters to tune in each source type in order to achieve an accurate detection, and the amount of information to process lead us to consider that the use of machine learning is a promising avenue to tackle the problem.

2.3 Intrusion Detection Systems and Learning

Intrusion Detection Systems are divided into two functional categories: misuse and anomaly. Misuse Intrusion Detection is based on rules that match specific intrusion patterns; these rules are commonly know as signatures. Misuse IDS have been shown to be very fast, reliable, and with low levels of false alarms. Even so, they cannot recognise new attacks that do not have a signature. Anomaly Intrusion Detection models the

behaviour of the system. They identify normal behaviour and everything outside of the normal pattern is considered an intrusion. This characteristic allows them to discover new attacks with the trade-off that they generate more false alarms than IDS based on misuse. To overcome this problem, researchers have started to use machine learning techniques instead of just statistical methods. Debar, Becker and Sibone [55] state that artificial intelligence techniques are the only way to built adaptive and efficient intrusion detection systems.

The work of Debar, Becker and Sibone [55] is one of the first to propose neural networks in intrusion detection systems. Their work couples a misuse intrusion detection as an expert system with a neural network. This work is based on HIDS. In a later work Rhodes *et al.* [144] include the use of self organising maps with neural networks in a hybrid misuse-anomaly IDS. In a similar work but focused on NIDS, Bivens *et al.* [25] use neural networks and self organizing maps. Liu, Florez and Bridges [108] used neural networks, back-propagation algorithms, self organizing maps and radial basis function networks to provide learning capabilities to HIDS. Bolzoni, Etalle and Hartel [26] use self-organising maps to develop an anomaly based IDS. They use the *The DARPA off-line intrusion detection evaluation* [105] to benchmark and train the system. We will cover DARPA evaluation in more detail later.

Kim and Bentley [94] proposed the use of an artificial immune model for network intrusion detection. This view was followed by Hang and Dai [71] to train a supervised learning system to perform anomaly detection. Interestingly they trained their system using not only positive examples but also negatives. They conclude that this approach helps to reduce the number of false alarms.

As with standalone IDSs, researchers have realised that including learning capabilities in DIDS could improve their reliability and their adaptability to new types of attacks. In addition to the challenges that intrusion detection in single agents presents to machine learning, in DIDS we need to address coordination and scalability problems. We will review these problems in detail in the next chapter.

The Collaborative Intrusion Detection System (CIDS) [54] uses homogeneous IDS agents within the Cougaar (Cognitive Agent Architecture) architecture, which is an open source software that enables building distributed agent-based architectures. CIDS agents use a blackboard system to communicate. The agents are the sensors of the

network and they communicate with a central agent that processes the information using fuzzy logic. Helmer and colleagues [74] propose the use of data mining to detect intrusions. In their work they describe a DIDS architecture using homogeneous agents to collect information, mobile agents to examine data in collectors and supervised learning algorithms to identify abnormal behaviour.

Zhang *et al.* [196] proposed a DIDS based on clustering. They use *clustering techniques* to group data that show similar characteristics according to some predefined metrics. The data mining technique is aimed at discovering patterns from the large amount of data that distributed sensors generate. The big advantage of the proposed DIDS is that it does not require a labeled dataset. Bayesian methods have also been proposed to merge the large amounts of data produced from several stand alone IDS [22]. Manikopoulos and Papavassiliou [112] describe the use of bayesian methods and neural networks for network intrusion and fault detection. Their system is capable of detecting UDP flood-based denial of service attacks through the statistical analysis of the sensor data that is input to a neural network classifier. They presented positive results applying their technique to a simulated wireless ad-hoc network using OPNET [130]. Using clustering but applied to routing attacks in sensor networks is the work by Loo *et al.* [110]. They analyze attacks against the *Ad hoc On-demand Distance Vector* (AODV) routing protocol and show how to provide protection using an adaptive algorithm based on clustering. They evaluate their anomaly intrusion detection approach using the network simulator NS-2 and they show that their system is capable of detecting unknown attacks.

Abraham [1] presents the *Distributed soft computing intrusion detection system* D-SCIDS. This is a hierarchical DIDS architecture with several levels and a central controller. In each level there are aggregation controllers. To perform intrusion detection they use a large set of discrete and continuous variables as features that are processed with *soft computing*. The authors describe soft computing as a hybrid system composed of neural networks, fuzzy systems, approximate reasoning, and derivative free optimisation techniques.

Following the approach of using data from several sources, Siaterlis and Maglaris [160] use data from misuse IDS (Snort [162]), Netflow collectors and SNMP to construct a DIDS to detect denial of service attacks. The data is locally collected and pre-

processed by autonomous agents with a certain degree of intelligence based on expert knowledge as described by the authors. The local beliefs of sensors are not enough to guarantee accurate detection and they send summarised information to a central repository. In the repository and by means of a Bayesian method a final decision is made. Using this approach the authors significantly reduce the number of false positives in the monitored network.

The use of machine learning techniques to provide adaptation capabilities to intrusion detection is a promising area of research. However, it has received some critique [57, 20] relating to how IDS using these techniques could be maliciously manipulated by the attacker. Although this may not be easily performed, it is possible as demonstrated by Chang and Mok [43]. They were capable to successfully produce theoretical attacks against intrusion detection systems that automatically generate signatures from malicious activity. The authors also describe some design alternatives that can be employed to minimise the risk of malicious manipulation. The malicious manipulation of machine learning IDS it is an interesting problem related to the trade-offs that security researchers have to deal with. The balance between very smart applications capable off detecting zero-day exploits but susceptible to manipulation is still a grey area with very little research. As security mechanisms become more intelligent and attacks more complex, we expect that this area will receive more attention.

2.4 Conclusions

In this chapter we have reviewed a number of research works that deal with the complex problem of providing reliable intrusion detection. We analysed different types of IDSs from functional and architectural points of view. From a functional point of view we analysed the two major divisions of these systems: anomaly IDS and misuse IDS.

From the architectural point of view we reviewed Network IDS (NIDS), Host IDS (HIDS) and Distributed IDS (DIDS). Anomaly IDSs analyse network, system or/and user activity and assume that any deviation of from the normal behaviour is an intrusion. This characteristic allows them to detect new and unseen attacks but they also suffer high rates of false alarms compared to other IDSs. Misuse IDSs detect intrusion using pattern matching rules, this makes them very reliable but also they cannot detect new or

modified (according to the rules) types of attacks. In this chapter we analysed a variety of techniques aimed to provide to IDSs with both reliable detection and adaptability to new attacks.

Network IDS analyses flows of data within the network searching for intrusion. Host IDS are programs that analyse a diverse set of host activities. Distributed IDS is a group of homogeneous or heterogeneous IDSs working together. The level of collaboration may vary, some systems report sensor information to a central repository while some other share and process data autonomously.

In our analysis we have also focused our attention to describe some approaches that use machine learning techniques to address the problems of reliability, adaptability and distributed processing on intrusion detection systems. The approaches presented have used anomaly intrusion detection to provide adaptability to new attacks, they have used machine learning techniques to improve misuse detection based systems or they have dealt with the problem of how to merge data from several sources in distributed environments. There is some potential in the use of machine learning to tackle the problem of identifying complex and new types of attacks. There is some concern relating to how IDSs using these techniques could be maliciously manipulated by the attacker.

In this chapter we analyse a special type of Denial of Service and Distributed Denial of Service attacks referred to as Flood-Based DoS and Flood-Based DDoS. These attacks are based on a flood of packets with the intention of exhausting in some way the network resources of the victim. Although these attacks have been widely studied and they present a high risk to almost any organisation providing information services through the Internet, their identification and containment is still a challenge and an area of active research. A variety of techniques using rule-matching, statistical methods and machine learning have been proposed to deal with problems related to intrusion detection.

We think that in order to accurately identify malicious activity, we require a solution based on a distributed approach, the use of a variety of source information, and an ability to recognise new attacks and traffic patterns. To provide all these capabilities we consider that we need a machine learning technique capable of working in distributed environments, able to learn online to modify its learned profiles to new network environments and capable of learning from environmental signals to minimise relying on training data. One suitable way to provide these learning capabilities is Reinforcement

Learning (RL). In the following chapters we will analyse how RL can be used to this task.

Chapter 3

Reinforcement Learning

It can be said that a program is *learning* when it executes a task T and acquires experience E and the measured performance P of T improves with experience E [119]. To provide learning capabilities to a program, computer science researches have taken different approaches. The learning process may be supervised, unsupervised or reinforced (Reinforcement Learning) [153]. Other authors just describe the learning process as supervised or unsupervised [126]. In *supervised learning*, the program learns a concept through an external teacher providing learning information in a set E of categorized training examples. In order to learn the concept, the program generates a hypothesis or function f using the training set E. After learning, any new subset of examples will be accordingly categorized. Examples of supervised learning are decision trees and artificial neural networks.

Contrary to supervised learning, in *unsupervised learning* the learner received a training set of examples but these are not categorized. These methods try to split the training set into subsets in order to accommodate examples in a proper way. Unsupervised learning is commonly used in taxonomy problems aimed at classifying data into meaningful categories. Unsupervised learning methods include non-negative matrix factorization and self-organising maps (SOM) [152].

Another method for programs to learn is *Reinforcement Learning* (RL). In Reinforcement Learning the program learns through its interaction with the environment. Agents or programs sense their environment in discrete time steps and they map those inputs to local state information. RL agents execute actions and observe the feedback

from the environment or a trainer in the form of positive or negative rewards. After performing an action and receiving a reward, the agent observes any change in the environment and it updates its policy in order to optimize the reward received for future actions [172].

In this chapter we introduce the mechanisms that make RL a suitable solution to problems where a model is complex to build, training examples are difficult to get or we require an *on line* learning approach. We have divided the chapter into two main parts, a background on Reinforcement Learning in general and a second part about the use of RL in distributed environments. The first part initially presents the foundations that supports RL and it continues with a brief description of two important algorithms commonly used. We conclude this part discussing the exploration vs. exploitation problem and the use of function approximation techniques. The second part first discusses the basis of Multi Agent Reinforcement Learning and the common challenges that it poses. We continue discussing in detail MARL related problems such as coordination, scalability and communication. We also introduce and discuss important research results on how to deal with these problems.

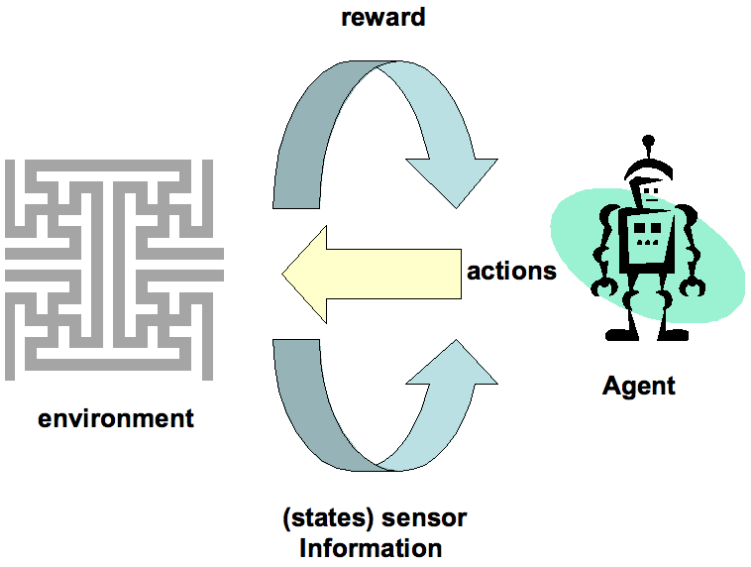
3.1 Reinforcement Learning

In *Reinforcement Learning (RL)* agents or programs receive sensor information from the environment and they map these inputs to states. Changes in the environment are represented in different sensor information to the agent and they are mapped into different local states. Agents then execute an action and they observe the feedback in the form of positive or negative rewards (Fig.3.1). The action selection is based on the maximization of a specific principle, examples of these principles are the *immediate reward*, *average reward per time step* and *total discontinued reward* among others. In the last steps of the RL process, agents map changes in the environment as new states and they update the learning policy to optimize future rewards.

3.1.1 Markov Decision Process

A reinforcement learning problem can be formally defined as a *Markov Decision Process* (MDP). Although RL is not constricted to MDPs the use of this theoretical framework

Figure 3.1: Reinforcement Learning



permits the study of its algorithms and its properties [21]. The MDP is defined by a 4-tuple $< S,A,R,T >$. S is a finite set of observable states by the agent. The states are mapping information from the agents sensor inputs. A is a finite set of actions available to the agent to perform. *R(s,a)* defines the reward obtained in state s after performing action *a*. *T(s,a,s')* $\rightarrow$ [0,1] is a probability transition function. It defines the probability to transit from state s to state *s'* after executing action *a*. In a MDP the *R(s,a)* and the *T(s,a,s')* functions only depend on the current state and actions. The probability distribution for a MDP is defined by Eq.(3.1) [172]:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \tag{3.1}$$

Eq.(3.1) is a special case of the more general scenario where the current state also depends of past states and actions (Eq.(3.2)) [172]

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{3.2}$$

When Eq.(3.1) is equal to Eq.(3.2) for all *s'*, r and the sequence s_t, a_t, r_t, s_1, a_1, r_1, s_0, a_0, r_0 we say that the state has the Markov property. The use of the Markov property

in reinforcement learning is very useful because it lets us predict the next state and the expected reward with only the current state and action. In practice however, agents have limited capabilities to sense every piece of environment information or they simply cannot have access to specific information (i.e. the opponent cards in a pocker game). In these cases we have hidden state information and they may not fully satisfy the Markov property; nevertheless it is useful and convenient to consider them as Markov. Those cases are referred as *Partially Observable Markov Decision Process* (POMDP).

3.1.2 Value Function

At every time step agents observe states and execute actions. The state-action mapping is called a *policy* π. Informally the probability of selecting an action a in a given state s under policy π is $\pi(s,a) \rightarrow [0,1]$ and it is defined by V^{π}. This is also known as the *state-value function for policy* π. Formally for MDP $V^{\pi}(s)$ is defined as Eq.(3.3) [172]:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \tag{3.3}$$

In this equation E_{π} is the expected value received if the agent follows policy π and it is in time state s_t. The equation can be rewriten as Eq.(3.4) and Eq.(3.5) [119] as the sum of rewards accumulated over time where $\gamma \in$ [0,1] is a discounted value that weights past against more recent rewards.

$$V\pi(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \tag{3.4}$$

$$V\pi(s_t) \equiv \sum_{k=0}^{\infty} \gamma^k r_{t+i} \tag{3.5}$$

Another important concept is Q^{π}, this is the *action-value function for policy* π. Q^{π} is the value or expected return of taking action a in state s and following policy π. It is defined as $Q^{\pi}(s,a)$ in Eq.(3.6) [172]:

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \tag{3.6}$$

The goal of the agent is to find the policy that maximizes the rewards obtained over the time. This is called the *optimal policy* and it is denoted by π^*. The value function that gives the maximum discounted cumulative reward to the agent from state s and following π^* is defined as $V^*(s)$ and called *optimal state-value function*. $Q^*(s,a)$ is the *optimal action-value function*, its value is the expected return by taking action a in state s and then following an optimal policy.

3.1.3 Q-learning

There are different approaches to calculate the optimal policy and to maximize the obtained reward over the time. One of the most widely used techniques is *Q-learning* [183]. Q-learning is a method that learns the value function Q^*. As in other *Temporal-Difference-Learning* (TD) methods, in Q-learning the agent iteratively tries to estimate the value function. It exploits the foundations of the Bellman Equation [24] that states the relation between $V^*(s)$ and $Q^*(s,a)$. The value function V^π is the unique solution to the Bellman equation. In terms of the optimal policy, we could considerer that the V^* is generated by executing action a in each state s and then following the optimal policy π. The *Bellman backup* is defined as:

$$V^*(s) \leftarrow max_a \sum_{s'} T(s,a,s') \ \left[\ r\left(s,a,s'\right) + \gamma V(s') \ \right] \tag{3.7}$$

In this equation T is the *transition function* that represents the probabilities to move to state s' after executing action a in state s, α is the learning rate with a value $0 < \alpha < 1$ and γ is a constant with value $0 < \gamma < 1$ that represents the relative value of delayed versus immediate rewards.

The Q-learning algorithm (Fig. 3.2) forms a table which rows are states and columns are actions, the initial values of the table are randomly chosen. The agent in each state s chooses an action a according to the policy derived from Q, it observes the reward r and the next state s'. Then it updates the estimated Q-value denoted by $\hat{Q}$ in (3.8) [172].

$$\hat{Q}(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left(r + \gamma \ max_a \ \hat{Q}(s',a') \right) \tag{3.8}$$

The requirement to converge to Q^* with probability of 1 in Q-learning is that the

Figure 3.2: Q-learning algorithm [172]

- Initialize $Q(s, a)$ arbitrarily
- Repeat (for each episode)
  - Initialize s
  - Repeat (for each step episode)
    - Chose a from s using policy derived from Q
    - Take action a, observe r, s'
    - $\hat{Q}(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \; max_a \; \hat{Q}(s', a')\right)$
    - $s \rightarrow s'$
  - until s is terminal

agent must visit all the states an infinite number of times. In practice this requirement is not possible and it raises other issues such as the exploration vs. exploitation strategy that we will discuss later.

3.1.4 SARSA

Another method to calculate the optimal policy is *SARSA*. SARSA is named after the letters from the tuples used to evaluate the optimal state value function $< s, a, r, s', a' >$. SARSA algorithm is very similar to Q-learning, but contrary to Q-learning that updates the Q estimate using the maximum Q-value independently of the action chosen, SARSA updates the Q-function using the Q-value of the selected action.

SARSA is not guaranteed to converge in every case to the optimal policy [172, 168] but some research work shows that this is possible under certain conditions . To converge to the optimal policy in SARSA the agent must visit infinite number or times all the state-action pairs using a greedy policy and with specific probabilities [139].

3.1.5 Exploration vs. Exploitation

The exploration vs. exploitation problem is a specific challenge that is present in reinforcement learning algorithms. To obtain the best reward, the agent will prefer actions that have been proved to provide higher returns. However, most of the times in order to discover the actions with better returns the agent needs to try actions that have

Figure 3.3: SARSA algorithm [172]

- Initialize $Q(s, a)$ arbitrarily
- Repeat (for each episode)
  - Initialize s
  - Choose a from s using policy derived from Q
  - Repeat (for each step episode)
    - Take action a, observe r, s'
    - Choose a from s using policy derived from Q
    - $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\left(r + \gamma Q(s', a')\right)$
    - $s \rightarrow s'; a \rightarrow a'$
  - until s is terminal

not been evaluated or that currently seen to yield a lower long-term reward. We say that the agent *exploits* actions that lead to better expected rewards but also it needs to *explore* other actions that may lead it to better rewards in the short or long-run [172]. Besides, some RL algorithms such as Q-Learning and SARSA require that the agent must visit all the states infinitely often in order to converge to the optimal policy. In order to find the actions that offer the better returns, agents need to explore among the set of available actions. However, it is unfeasible to explore all the time and they need to exploit the actions offering the better expectations. We must therefore balance the levels of exploration and exploitation. The methods to overcome the problem are referred as *exploration/exploitation strategies*.

One simple solution is to use a *random* strategy. While in theory this strategy guarantees convergence; in practice it is very slow. A more subtle alternative is to let the agent explore actions in the beginning of the learning and progressively start choosing those actions that lead to better expected rewards. *epsilon-greedy* and *Boltzmann* use this alternative. Epsilon-greedy is a semi-uniform random exploration strategy that uses a small value as a base probability to choose an action. The downside of epsilon-greedy is that it chooses among all the actions with the same probability. To address this problem, the Boltzmann strategy, also called *softmax action selection rules*, weights each action with a probability according to its expected value using the equation:

$$P(a) = \frac{e^{Q(s,a_n)/T}}{\sum_0^i e^{Q(s,a_i)/T}} \tag{3.9}$$

T is a positive number called *temperature*. Under high values of temperature the action selection tends to choose equally between all actions. Low values of temperature favour actions with high expected values. In practice, to speed up convergence the value of the temperature is decreased exponentially.

3.1.6 Function Approximation

When RL is used in real world applications it is neither practical nor feasible to map all sensor information to individual states because:

- RL algorithms using Q-tables require input information to be discrete. In many real world applications inputs and states are in a continuous form.
- In general the state space is represented by the cartesian product of n input variables. As the number of variables increases, the state space S increases exponentially. The result is that learning is commonly infeasible or extremely time and resource consuming [16]. This effect is commonly known as the *curse of dimensionality* and it means that computational requirements grow exponentially with the number of state variables [172]. For example one input variable with two values represents two states. Adding another input variable means four states, having 3 variables requires eight states and so on in an exponential growth. Adding more values has a similar effect, for example three variables with three values are represented with 27 states. Considering that some RL algorithms require the agent to explore many times all the states, the growth in state-space represents an increase in time to learn and in computing power.
- In large state spaces it is common that similar states exist.
- There may exist states that the agent will never visit and subsequently will not be trained on them.

To tackle these problems we use *function approximation* techniques. These techniques commonly used in supervised learning are adapted to RL in order to represent the

Q-function. To learn which actions may lead to higher rewards the agent needs to estimate a *value function* for each action. A simple method to represent this value function is by state-action tables. However as previously stated this limits scaling RL up to large state-action spaces. To approximate to the value function we can use a diversity of methods such as *sparse coarse coding* (also known as Tile Coding) [170], *multi-layer neural networks* [2], *random representations* (Kanerva Coding) [173] and *gradient-descent methods* among others. Specially successful in real world applications are sparse coarse coding in domains such as Robo Cup [167], Elevator Control [51] and Mechanical Control Systems [170] and Random Representations in Robo Cup [97].

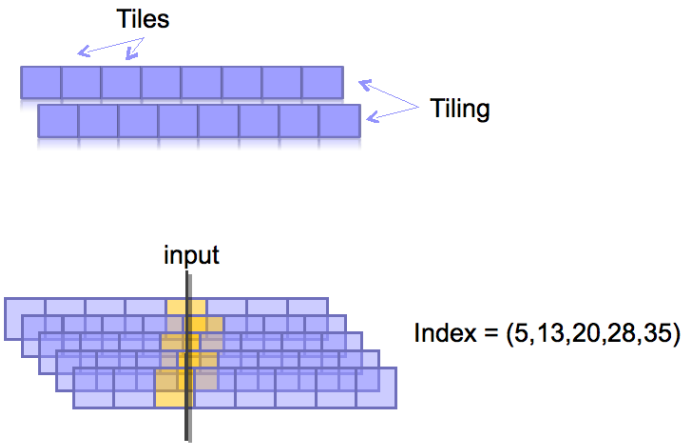
For a given policy the function approximator for the Q-function has the form of $\hat{Q}_w(s,a)$ where s is the state, a is the action and w is a set of adjustable parameters or weights. For TD methods the weights can be updated as shown in Equation 3.10. In this equation $(r_{t+1} + \gamma\hat{Q}_{t+1} - \hat{Q}_t)$ represents the Q-function update according to the estimates of Q $(\hat{Q})$, the current reward and the discontinued rewards:

$$\Delta w_t = \alpha \left(r_{t+1} + \gamma\hat{Q}_{t+1} - \hat{Q}_t \right) \sum_{k=0}^{t} (\lambda\gamma)^{t-k} \nabla_{wk}\hat{Q}_k \tag{3.10}$$

In order to be functional and efficient function approximator must comply with the following characteristics [148]:

- **Generalisation:** There are many states that agents will never visit and they will not be training on them. Generalisation is the ability to extrapolate observed state-action pairs values to the unobserved state-action pairs.
- **Resolution:** Resolution refers to the granularity with which continuous variables are represented as discrete values.
- **Storage:** This characteristic refers to how optimized is the function approximator in the use of memory resources. In general terms, the more resolution, the more accurate the Q-function representation is and the more storage required.
- **Computational efficiency:** This term refers to how complex are the algorithms to compute the Q-function representation. This characteristic is important because agents generally perform the evaluation operation several times and in many scenarios the agent processing resources are scarce.

Figure 3.4: Tile Coding



Tile coding [172] (also referred as *Cerebellar Model Articulation Controller* or CMAC [170, 148]) is a type of sparse coarse coding where the features or characteristics that we want to analyse are grouped in a set of partitions. These partitions form the input space and they are called *tilings*. Each tiling is divided into small pieces called *tiles* and each one has a corresponding weight. To produce the state-action feature representation several tilings are overlapped (See Fig. 3.4). The input features activate a set of tiles and their correspondent weights. The value of the state-action ($V\pi(s)$) is the arithmetic sum of all the weights ($\vec{w}$) of the tiles activated by the input. In general, the size and resolution of $\vec{w}$ depends on the number of tilings and the number of tiles per tiling.

Function approximation techniques are based on estimates of the real inputs. Because they can lose information in the estimate creation they cannot guarantee convergence of the RL algorithms; and even in some cases [28] they can diverge from the optimal value function. Despite this disadvantage tile coding has been successfully used theoretically and practically in diverse research works. Sutton [170] demonstrate its applicability along with the SARSA algorithm by showing successful results on the *2D gridworld*, *the puddle world* and *the mountain car problems*. Santa Maria *et al.* [148] expand the study of Tile Coding. They show how is it possible to modify tile coding

to obtain better resolutions in specific areas of the state-action space. The authors conclude that although non-uniform function approximators are harder to design they lead to better computational efficiency and memory storage utilisation. In more recent research, Sherstov and Stone [157] explore the use of Q-learning and tile coding to represent the value function on continuous-action domains. They use the gridworld to test discrete states along with a continuous action representation in the form of cardinal directions. They conclude that this method can be used to accelerate the learning rate of RL agents.

Tile coding and RL have been also used to solve real world problems. Stone *et al* [168] applied the SARSA algorithm and tile coding on the *Keepaway* problem of the Robocup Soccer domain. In this work a group of RL robots learn how to keep the ball away of a hand coded agent. Yagan *et al.* [190] apply tile coding and the *Semi-Markov Average Reward Technique (SMART)* to estimate the value function. They aim to dynamically allocate *Quality of Service (QoS)* based bandwidth in a network domain problem. In this problem agents are trained using a reward that depends on how they optimize specific environmental constraints whilst they allocate the correct bandwidth to applications. Andreasson *et al.* [7] analyze the use of SARSA and tile coding for garbage collection in the Java Virtual Machine. In this work the authors construct a prototype and propose a set of variables as features, the actions to execute and the possible rewards functions. They compare the prototype's performance with a hand-coded solution and although further investigation is required, they conclude that RL may achieve better performance.

3.2 MARL

Distributed Artificial Intelligence (DAI) is defined as *'the study, construction, and application of multiagent systems, that is, systems in which several interacting, intelligent agents pursue some sets of goals or perform some sets of tasks'* [153]. DAI can be also subdivided into *Distributed Problem Solving* (DPS) and *Multiagent Systems* (MAS) [169]. DPS deals with management information problems such as the processing of data through several entities or programs and how to reassemble the results in one solution. MAS researchers study principles and mechanisms for the construction of autonomous

agents that interact with each other and their environment. In Multiagent Systems the agents form a network of problem solvers that interact with each other and their environment to solve problems that are beyond their individual capabilities; furthermore as a coordinated team they can successfully reach their goal or task [174].

In Reinforcement Learning, agents do not require training examples or a predefined model to learn. They learn optimal behaviour from the actions that they execute and the reward that they received from the environment. The interaction between agents and environment and the possibility to use a *model free operation* makes reinforcement learning especially interesting to solve interactive and complex problems. Generally, in these scenarios the model creation is complicated or it is difficult and impractical to obtain examples of the desired behaviour. These advantages over other machine learning techniques also drew attention of researchers in MAS technologies.

Whatever the appealing characteristics are, the use of *Multi-Agent Reinforcement Learning* (MARL) poses difficult problems. The *curse of dimensionality* that affects standalone RL and other machine techniques has bigger effects in MARL. As the number of agents and states increases it becomes difficult to scale these systems to a large number of agents. Different approaches from function approximation techniques [96, 168] to hierarchical reinforcement learning [21, 62] have been proposed to scale MARL to large number of agents.

In summary, some of the main issues surrounding MARL are:

1. In single agent RL, agents need only to adapt their behaviour in accordance with their own actions and how they change the environment. In MARL agents also need to adapt to other agents' learning and actions. The effect is that agents can execute the same action on the same state and receive different rewards, not only because of its action, but also because of the action of other agents or by the effect of the combined actions of all the agents in the same state interacting with the environment. This creates the problem of a partial observability creating a non-markovian environment where the foundations of the RL algorithms do not longer apply.

2. MARL agents do not always have a full view of the environment and even if they have, they normally cannot predict the actions of other agents and the changes in

the environment [88]. This partial view of the of a dynamic environment forces us to seek forward new algorithms adaptable to these new problem domains.

3. The *credit assignment problem* [153] describes the difficulty of deciding which agent is responsible for successes or failures. This raises problems in how to split the reward signal among the agents and the trade-off between the use of local and global rewards to achieve fast learning or to guarantee to converge to a *global optimal policy*.

4. There are different levels of communication among agents in a MAS [169]. The decision to use communication and what type of information to share is one of the problems that face MAS researchers. In the case of communication MAS, the type and amount of information to share brings on challenges related to environmental constraints such as bandwidth.

Below we discuss in more detail the challenges faced when using MAS technology and Reinforcement Learning. We also present some research work related to solve these problems.

3.2.1 The Learning Problem

There are two categories of learning MAS: *team learning* and *concurrent learning* [132]. Team learning refers to a single agent learning the behaviour of a set of agents. The advantage of this type of learning is that it can use machine learning techniques with little or no modification. When a set of agents in a MAS are learning at the same time we call it concurrent learning. Concurrent learning is an interesting field of research because of its expected computational power and flexibility and the challenges that it presents. The most challenging problem is how to adapt machine learning techniques to a group of agents residing in a dynamic environment. Generally, concurrent learning may be preferable when the problem can be partitioned and divided. This could bring the opportunity to split the problem in several sub problems that could be solved independently.

RL is based on rewards and one of the first questions in MAS using RL techniques is which agent to reward. This problem is commonly know as the credit assignment problem [153]. A simple alternative is to give the same reward to every agent which is

referred as *global reward*. The problem with global reward is that it is difficult for the agents to know how to interpret the reward for their own computations. From the agent perspective, the same action executed in the same state could lead to different rewards, this situation definitely complicates the optimal value function computation. The *local reward* approach rewards agents based only in their own achievements. The downside of this approach are the complexity to identify the level of contribution that each agent has to the success or failure of the task and the possible generation of greedy agents [132]. Greedy agents are interested in maximizing their own reward and may not be interested in cooperating with other agents [96].

3.2.2 Coordination

Research on MARL has typically used stateless systems with a small number of agents capable of performing just two or three actions. Early work used zero-sum games on non-cooperative multi agent systems allowing researchers to adapt reinforcement learning techniques to multi agent system environments. Examples of these works are *minimax Q-learning* [106] and Nash-Q [77, 78]. MARL based on zero-zum games and *Nash equilibria* are based on the assumption that the agent's actions will converge to an equilibrium where agents will not have an incentive to change their strategy. However there is some critique [158, 138, 159] on the convergence of agents to a Nash-equilibria instead of a globally *team-optimal* solution [132].

The set of actions chosen by every agent in a certain time is call the *joint action*. When the agents do not know about other agents they are *independent learners* (IL). Independent learners perceive indirectly other agents only by the rewards of the joint action. Independent learners update their Q-values in the same way as single agent systems using solely their own actions. Agents that are aware of other agents are *joint action learners* (JALs). JALs update their Q-values using joint actions. In *cooperative MARL* the optimal joint action is the set of actions that lead the agents to the best expected reward. In these learning systems the agents try to coordinate their actions to achieve the optimal joint action.

Claus and Boutilier [47] studied cooperative MARL. In their research two agents in a single state game were trying to coordinate their actions to obtain a reward. They observed coordination problems in scenarios with multiple optimal joint actions or when

miscoordination is penalized with high negative rewards. To illustrate it, they introduced the *climbing game* and the *penalty game*. The climbing game presents problems in coordination due to high miscoordination penalty and single optimal joint action while the penalty game presents high miscoordination penalty and multiple optimal joint actions. Assuming an optimistic behaviour in which agents cooperate with each other, Lauer and Riedmiller [100] modify the Q-learning algorithm to overcome the coordination problem on deterministic environments. The algorithm allows the agents to remember the order of the actions that are proven to offer a maximum reward. Kapetanakis and Kudenko [91] proposed *Frequency Maximum Q Value* (FMQ) to solve the coordination problem on stochastic environments. In FMQ the authors encouraged the cooperation of agents by modifying the exploration strategy to carry information about the frequency that an action produces a maximum reward. Chang *et al.* [40] tackle the coordination problem generated by increasing the number of agents using *Kalman Filters* to separate the noisy global reward from a more informational signal. Using this signal the authors were able to approximate a local reward more useful to construct a near-optimal policy.

Some other works investigating how to improve the coordination capabilities of agents are *Hyper-Q* learning [179], *Exploring Selfish RL* [92], *Incremental Policy Learning* [67] and *Lenient Learners* [131]. An analysis of a variety of research investigating coordination, scalability, communication and other problems in MARL is given by Panait and Luke [132]. An updated survey of the research in these areas can be also found in Busoniu *et al.* [30].

3.2.3 Communication

There are some scenarios where agents can work independently of each other without communication, however for some problems the use of communication can improve the performance of the multi-agent system [178]. When communication is required, this is generally restricted in terms of latency and bandwidth [132]. Systems with unrestricted communication are generally infeasible by the problem domain and researchers have focused on MAS without or with minimal communication among agents. These systems may be further divided into *direct communication MAS* and *indirect communication MAS*. In direct communication agents explicitly inter-change information in

the form of signals, blackboards systems and messaging. In systems with indirect communication agents share information implicitly by means of traces. Generally indirect communication MAS are inspired in biological entities such as bees' colonies and ant systems.

In his research Tan [178] evaluated the performance of agents by sharing sensations, sharing episodes and sharing learned policies. He concludes that learning is improved when agents can communicate, however excessive sensory information can have negative effects. Hamid and Vengerov [70] used agents that update a common policy, they argue that this approach reduces the risk to converge to non-optimal policies. Nunez and Oliveira [128] use an approach where agents communicate by sharing sensations along with *advices exchanges*. In the work of Huang *et al.* [79] agents share their state-action space to improve the learning rate. Although not based on RL, Jim and Giles [89] present interesting works based on the predator-prey pursuit problem where agents learn communication signals posted in a blackboard system. In this research agents post binary signals to the blackboard and the rest of the agents learn the semantics of the message. In a similar fashion, Afsharchi *et al.* [3] present another research work where agents learn and teach each other ontologies and concepts.

3.2.4 Scalability

The vast majority of the research on MARL has been done using a small number or agents and its scalability to a large number of agents is still an area of active research. One approach commonly used is to discompose the learning problem into simpler tasks. One approach is *Hiearchical Reinforcement Learning* [58, 133, 21]. Hierarchical RL aims to reduce the effects of the curse of dimensionality by splitting complex high level tasks into simpler low level actions [185]. In Hierarchical RL agents first learn a primitive set of actions, once they have learned basic actions they start learning high level tasks. This reduces the state-action space and accelerates the learning rate. Most research is based on agents that stop learning before moving from a lower to a higher level, however for some cases is it possible to have concurrent learning [185]. Hierarchical RL has been successfully applied in MARL in the previously mentioned work by Stone *et al.* [168] in the Robocup keep-away problem, Elfwing *et al.* [62] using *macro actions*, and by Ghavamzadeh *et al.* [68]. The research of Elfwing *et al.* [62] uses macro actions. These

macro actions force the agent to execute the same primitive action several times. The result is that agents are able to improve their learning. Similarly, the agents in the research work of Ghavamzadeh *et al.* [68] rather than sharing information at the level of primitive actions, they share information at the level of cooperative subtasks.

In MARL using global rewards the miscoordination problem is more evident by increasing the number of agents. As previously mentioned Chang *et al.* [40] tackle this problem by estimating a local reward from a global one. Taking an opposite approach to global rewards, Schneider *et al.* [149] propose the use of only local sensor information and local reward to calculate a local policy. To coordinate actions the system tries to optimize the sum of all the local rewards. Another research pursuing this approach is proposed by Bagnel and Ng [15]. Although the use of local rewards or its approximation from global rewards seems promising to improve the learning it does not guarantee to converge to an optimal global policy.

Another approach to scale MARL to a large number of agents is to reduce the state-action space by means of function approximation techniques. Function approximation techniques such as *sparse coarse coding*, *multi-layer neural networks*, *random representations* and *gradient-descent methods* estimate the value of the Q-function. In the domain of the Robocup, applied to independent learners and using Tile Coding is the work of Stone *et al.* [167]. Similarly Kostadis *et al.* [97] have applied successfully Kanerva Coding to the same problem. Kanerva Coding is a type of Random Representation that uses an array of binary features that are compared with a set of random representations. Inputs and representations are compared to activate an output set of weights [90]. Takahashi and Asada [177, 176] presents a hierarchical architecture applied to a group of JAL. They use low level sensor agents to learn specific actions confined to they narrow vision of the environment. High level agents have a wider scope of the task and they communicate with the lower layers to obtain sensor information. The input from higher layers is the output generated by learning policies in lower layers. Specifically they use *Continuos Q-learning* [177] as the function approximation technique. Continuous Q-learning creates a multidimensional matrix with the sensor information and action space vectors. They arrange the vectors in *n-dimensional* hypercubes which vertices are used to calculate an output vector of weights. Abdul *et al.* [2] applied a function approximator based on *Multilayer Feed-Forward Perceptron* (MLP) to a group

of JAL. In ANN, the back-propagation algorithm updates weights in the direction of error-gradients. In MLP the weights are updated using *output gradients* of the current input.

3.3 Conclusions

MARL has been proved to be a challenging field to researchers. Even in environments with a reduced state-action space and a few agents, researchers face important challenges related to coordination, scalability and communication. We have reviewed how the simple transformation of single agent's RL algorithms is not enough to provide a feasible approach to tackle real world problems using MARL. Despite these facts, we believe that the use of MARL may help to address some problems where training data is difficult to obtain, a model is complex to build and a distributed architecture of autonomous agent is required.

Problems that require a high degree of adaptability to rapidly changing environmental information and a distributed processing of high amounts of information are especially interesting to address with MARL. In this research we present the problem of detecting abnormal network states generated by Denial of Service and Distributed Denial of Service attacks. Due to the distributed nature of these attacks, the diverse sources of information that are required to identify them and the adaptation capabilities that detection engines require to recognize new attacks, we propose the use of an adaptable distributed platform such as MARL to tackle this problem. In the next chapter we will analyze and discuss how other research has used RL in single agent architectures and in multi agent systems to address the problem of intrusion detection, and particularly to detect Denial of Service and Distributed Denial of Service Attacks.

Chapter 4

RL, IDS and DoS Attacks

The use of RL in the intrusion detection domain has not been widely studied and even less so in distributed intrusion detection. Nevertheless, RL has been applied experimentally to solve some other problems in computer network domains. Especially interesting for researchers have been the areas of routing protocols, admission control and quality of service enforcement. This interest may be due by the fact that RL is very suitable to be employed in control problems where there is a feedback from the environment. In all the cases described we can find some sort of feedback that can be used as a reward.

In this chapter we present a review of some research works that have been experimentally applied RL in single agent and in multi-agent systems to tackle problems related to computer networks and to intrusion detection. We first review how RL has been used in the domains of routing, call admission control and fault detection. Although not completely related to IDS, we consider that reviewing the mechanisms employed in those domains is essential for the development of an intrusion detection engine based on RL. The rest of the chapter reviews and discuss a variety of approaches employing RL in the intrusion detection domain. We review single agent implementations, distributed architectures and some cases where RL have been used to address the problem of Denial of Service attacks.

4.1 RL and Computer Networks

An application of MARL to networking environments is presented by Boyan and Littman [29]. They present a multi-agent system where cooperative agents learn how to route packets using optimal paths. The authors used Q-learning as a dynamic routing protocol in a network of several routers using delay times as metric. Routers send packets through their interfaces and receive a feedback or reward from the network in the form of delay times. Using this approach they learn the proper link to send traffic to a specific destination. Other routing protocols use the number of hops (routers) as the metric to select the best path to send a packet. A disadvantage of these protocols is that routers are unaware of network congestion. So, if the shortest path is congested, routers will be unaware of the problem and they will send the packets through this path even though it might not be the best. Boyan and Littman compare their results with a routing protocol using the number of hops as the metric to forward packets. They noticed that contrary to routers using the shortest path to forward packets, routers using the Q-learning approach were congestion aware and had better results under heavy traffic conditions. They also report that although Q-learning was very good under heavy conditions it took it a lot of time to find the best path when the load conditions changed from heavy to low. This delay to converge to the best action is consequence of the greedy policy followed by routers. Nowe *et al.* [127] addressed this problem by reducing the number of control packets and averaging the immediate reward in a period of time. They also added more exploration through the broadcasting of control packets at regular intervals. Both research works show how RL can be applied to dynamic environments that provide a feedback and require adaptive behaviour from the agents.

In networks supporting voice, data and video (converged or integrated networks) there is a need to provide different services and resources depending on the application. For example voice and video require minimize delays and jitter while data traffic requires maximum bandwidth. The problem to allocate resources such as bandwidth or available transmission channels is referred as *call admission control* (CAC). Although initial research works focused on voice calls, today it can be used for any type of application in converged networks. One of the first works to apply RL to the problem of call admission control to converged networks is Marbach *et al.* [113]. They state that even thouth the problem is naturally defined as a dynamic programming problem, it is to

complex to model with exact values. To address the problem they propose RL as means to approximate the dynamic programming value function. They decompose the problem in a way that each link had a value function that is updated with every event processed by the link (i.e. processed and terminated calls) and by a reward function. The reward is computed by the number of links along the path to destination. Under this technique each node learns which links have enough resources available to route the call or in the worst case they learn when to reject a call as result of no available paths. To evaluate their proposal they compare it against an Open Short Path First (OSPF) implementation. They modelled the call generation using a Poisson distribution. They show that the RL approach performs better than the OSPF implementation. The RL approach rejects fewer calls, the value of the total calls terminated is closer to the optimal and it is able to route calls to alternate paths when congestion occurred. Tong and Brown [180] applied the same principles to a similar research. Despite the similarities of this work, it presents interesting and important differences. The work of Tong and Brown applies CAC to calls and to data packets and it uses Q-learning as the value-function update. The use of Q-learning raises a problem. When a call is terminated, no action is required but a state transition is made. The solution is to update the value function only on states associated with call arrivals. The authors show that the optimal policy can be obtained by this method. These research works on CAC show how RL can be also applied to more complex problems to provide an adaptive mechanism to optimise resources.

An adaptive multiagent reinforcement learning method for solving congestion control problems on dynamic high-speed networks is presented by Hwang *et al.* [84]. Allocating resources on a high speed network domain presents similar challenges than other CAC problems using RL (i.e. the works of Marbach and Tong and Brown. For instance the authors state the difficulty of manually determining the threshold and sending rate required to provide applications with accurate QoS parameters. To address the problem, they propose an adaptive congestion control method deployed in edge nodes. Edge nodes or agents learn independently accordingly of their local information about queue lengths and data rates. They also learn to cooperate to optimize the global system resources by employing a computed global reward. The global reward is estimated by each agent using a fuzzy evaluator that uses low-bandwidth utilization, packet loss rate

and improper buffer utilisation for the estimation. State variables are composed of buffer length and sending rate and the estimates of the global reward signal. They are processed by a CMCC function approximator that generates hypercubes as the state representation. It is interesting how the authors used environment information for both, state information and to produce a global reward. Even though agents have total view of local information, at global scopes they only have partial observations. For example consider 2 nodes, A and B. B is the downstream node of A. When B transmits more traffic in order to get better rewards, A sees only B's actions part of the environment in the form of more traffic, but not as another agent. To model this non-stationary effect frequently found on MARL, the authors base their approach on stochastic games and Nash equilibria. The RL method used was TD with eligibility traces. They evaluate their architecture using the proposed method versus no-control and manual-control. They use throughput, buffer utilization, and packet loss as their metrics. They reporte that the no-control and manual-control methods showed low throughput and high packet loss whilst the RL approach was able to maximize bandwidth and buffer allocation with low packet loss. This shows how manual methods may not scale to complex problems and how RL solutions can be evaluated.

Chang *et al.* [40] applied RL to mobilized ad-hoc networking where agents are required to move to optimize the connectivity of the whole network. The problem with this scenario is that the agents have only partial observations of the environment. They, for example cannot see their absolute location in the grid. To address the coordination problem raised by the global reward they use Kalman Filters. The idea is to separate the noisy global reward from a more informational signal. In another area of research, Littman *et al.* [107] formulate the automatic network repair task as a RL problem. Using this method, the agent suffering a fault has the option to try a test action with a cost, or try a repair action. The repair action has a cost but it can restore the system. They model the actions required and the state transitions probabilities of the problem. Unfortunately the authors do not present experimental results evaluating this technique. RL mechanisms that aim to tackle CAC and fault detection problems are interesting for us because we think that they can be adapted to the intrusion detection domain. They also prove the feasibility of the use of RL in complex problems related to computer networks where it is also possible to get some kind of feedback as result of the agents'

actions.

4.2 RL and IDS

As we previously stated, the use of RL in intrusion detection has not been widely studied and even less so in distributed intrusion detection. Some of the earlier works were made by Cannady [34, 35]. Cannady indicates that neural networks are feasible solutions when they are trained for a specific problem domain with representative sets of training data. Still they are unable to adapt to new data until they are taken off line and retrained with the new sets of representative data. To address this problem he used a Cerebellar Model Articulation Controller (CMAC) Neural Network. This type of Neural Network has the capability for online-learning. They use a three-layer feedforward mechanism designed to produce a series of input-output mappings. In this research the single IDS-agent learns how to detect flood-based Denial of Service attack based on ICMP and UDP. The system initially learns how to detect ICMP attacks and through previous knowledge and continuous re-training it learns how to recognize new attacks based on the UDP protocol.

One approach used to find intrusions on HIDS is based on observing sequences of systems calls. These calls are issued by process running in the host and they are grouped in sets of traces. Each trace contains the list of systems calls generated by a processes from beginning to end. To apply machine learning techniques using sequences of systems calls researchers commonly construct a transition model using labelled examples of normal and attack activity. The states of the model are defined by short sequences of system calls in a single trace. Xu and Xie [189] applied *Hidden Markov Models* (HMM) and RL to detect host intrusion by learning the state transition probabilities. They argue that there are uncertainties in modelling the state transition on IDS and HMM are able to offer a suitable alternative to the problem. They used a linear function approximation technique and a temporal difference algorithm to update the value function. The system was trained off-line and they induced a reward of -1 for normal activity and +1 for attack activity. The authors reported positive results compared with other ML techniques that used the same training and evaluation set from the *The DARPA off-line intrusion detection evaluation data set* [105].

In another related work, Xu and Lou [187] applied temporal difference methods [172] to model dynamic behaviour in a HIDS approach. To approximate the value function and to extract features they used sparse kernel-based LS-TD(λ) algorithm. As described by the authors, the kernel-based LS-TD(λ) algorithm is a non-linear function estimation that uses a high-dimensional feature space and least-squares TD learning. To evaluate their approach the authors use system calls traces from the sendmail application. They showed positive and better results compared to a prior implementation using HMM-methods.

4.3 DIDS and RL

Miller and Inoue [116] use reinforcement learning to train a DIDS called Perceptual Intrusion Detection System with Reinforcement (SPIDeR). The system consists of heterogeneous agents performing intrusion detection and communicating through a blackboard system. All the agents have a three-layer architecture composed of a signature-based detection for well known intrusions, an array of SOM for anomaly detection, and a third layer to collect information to further analysis. The remote agents perform intrusion detection and they send their votes through the blackboard system about the local activity sensed. By means of a RL process the central blackboard system computes and weighs the votes and in turn it rewards the agents in accordance with it effectiveness. The authors evaluate SPIDeR using the KDDCup'99 data set [52] showing positive results.

By means of HMM, RL and the behavioural analysis of IP addresses Xiu *et al.* [188] propose a DIDS focused on detecting DoS/DDoS attacks. The architecture is composed of a group of sensors that have partial observability of the environment. Because of communication constraints, sensors are not able to send all their sensor information. Instead they learn to recognize local attacks and to communicate them to a central facility. Although the authors report high rates of detection, a possible drawback with this approach is the use of a single source of information (IP addresses) that can be easily forged. Furthermore, the authors reduced the problem of detecting DoS attacks to discriminate legal IP addresses from random IP addressees, which in our opinion is not enough information to accurate detect these types of attacks.

Awerbuch, Holmer and Rubens [12] applied RL to security and routing. They de-

veloped a secure routing architecture for wireless routing. The system is formed by a group of routers that share communication using secure channels. To route packets they use RL and they are capable of recognizing DoS attacks against the routing infrastructure. The authors used RL and game theory to train upstream agents to recognise DoS attacks. They used the feedback from downstream agents as the only reward signal. They assumed that under attack acknowledgement packets from receiving agents are lost which cause transmitting agents to look for alternate paths when they are under attack.

4.4 Conclusions

We have discussed the RL mechanisms used to tackle routing, CAC and fault detection problems. These are interesting for us because we consider that they can be adapted to detect and contain attacks. In this respect we reviewed how the routing mechanism was modified to detect DoS attacks on wireless routing infrastructure. In the area of RL and intrusion detection systems we discussed some mechanisms used to address adaptability and reliability problems in single IDS and DIDS implementations. These works have used RL in very different ways and some of them along with other ML approaches such as ANN or SOM, which give us an idea of the versatility of the RL mechanism. We think that even though the challenges that RL and MARL present related to their scalability and to agent-action coordination may be significant, they can be used to increase the reliability of IDSs and provide adaptation capabilities to zero-day and DoS/DDoS attacks.

In previous chapters we reviewed different mechanisms aimed at detect DoS attacks. Whether they are based on machine learning or not, all of them indicate that a feasible solution to recognize abnormal and malicious activity inside computer networks must be distributed and dynamic in nature. The ability to evolve and adapt is also important if new attacks are to be detected. This characteristic has been proved to be necessary against the continuous creation of attack tools from hackers that exploit unknown host, application and protocol vulnerabilities. To provide the adaptation capabilities ML seems to be a potential alternative, nevertheless it also brings other challenges. Some of them are the complexity of modelling these environments [64, 184] and how to get

accurate examples for training [8, 105]. In the next chapter we will propose an adaptive, heterogeneous and distributed architecture of agents capable of identifying and categorizing abnormal and malicious activity.

Chapter 5

Multi-Agent RL of Signaling

One method for attackers to launch Flood-Based Denial of Service (FBDoS) and Flood-Based Distributed Denial of Service (FBDDoS) attacks is to use several remote controlled sources trying to exhaust a target's key resource. Stand alone sensors or intrusion detection systems do not have all the information to accurately identify sources of such attacks. Yet, most of the time FBDoS and FBDDoS attacks are indistinguishable from normal activity. In this case a heterogeneous variety of sensors is required to collect and to process network and host data from multiple sources. Due to bandwidth restrictions it is infeasible to assume that agents are able to communicate their complete local state observations or that a single central repository is able to process all sensor information.

In summary, to effectively detect and categorize these attacks we require a distributed architecture. It is difficult to model the environment where these events happen and to get data to train adaptive approaches is not a simple task. Taking those factors into consideration we are inclined to propose RL as a feasible alternative to address this problem. Sensors must be capable of processing local information and of communicating a summarized set of data to coordinate the detection of complex or globally scoped attacks. To meet these requirements we propose *Multi-Agent RL of Signaling* (MARLS) for Distributed Intrusion Detection. In this chapter we will describe the design and the basis of the proposed approach to deal with FBDoS and FBDDoS attacks. The first section of the chapter presents a summary of the agent architecture. The rest of the chapter presents the structural design of each of the agents in the architecture, the basic agent structure defined as *Basic Collaborative Cell* and the hierarchical model

composed of several agents grouped in collaborative cells.

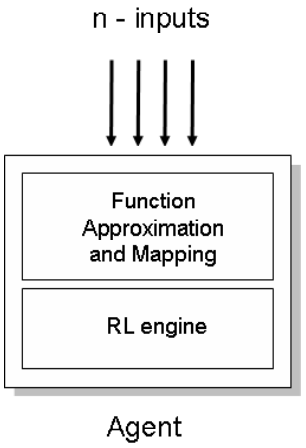
5.1 Multi-Agent RL of Signaling

MARLS-DIDS is a hierarchical agent architecture integrated by a heterogeneous remote sensors and reinforcement learning aiming to detect and categorize FBDoS and FBDDoS Attacks. Using this approach, distributed sensors process the local state information and pass on short signals up a hierarchy of RL-IDS agents. These signals are received by the RL-IDS and without any previous knowledge it learns their semantics and how to interpret their meaning. Through these signals and rewards from the environment, RL-IDS agents learn to distinguish abnormal activity using multiple information sources.

To expand the architecture to a large number of agents and to more global scopes, RL-IDS agents can in turn send signals up to a higher hierarchy of RL-IDS agents. These high level RL-IDS agent learn to detect and categorize attacks at regional or global scopes by means of the signals sent by low level agents. The lower the hierarchical level of the agent is, the more local information it processes. On the lowest level there are sensor agents directly collect local environment data and learn how to summarize it. The agent on top of the hierarchy learns whether or not to trigger an overall alarm to the network operator. The result is that with minimal communication and central processing, high-level hierarchical agents have a better overview of the current state of the whole network and local sensor agents are specialized in collecting specific sets of network information.

The proposed distributed architecture has a number of advantages over a single agent architecture and over a multi-agent system with full communication. In the case of the single agent it may be unfeasible or even impossible to collect a variety of environment state information in a single point. This is not only true for networking environments but it is also valid for other control systems. For the networking case for intrusion detection we have reviewed that we need to collect a variety of information dispersed in several entities within the network. Although the single agent architecture is technically possible for some network environments, we think it will not give as accurate results as a distributed approach [125, 22, 112, 117]. The distributed approach proposed has advantages over a full-comunicating MAS. In a full-communicating MAS distributed and

Figure 5.1: Agent Model



remote sensors are collecting network information but they do not process it locally. Instead they send all the collected data to a central repository for processing. This scenario may present important bottlenecks and constraints related to bandwidth and processing. Contrary to the full-communicating MAS our approach process data locally and it learns when and what information to provide to other agents. The outcome is that we are able to extract normal and abnormal states with low bandwidth utilisation, low central processing requirements and using a rich variety of source information needed for high accurate intrusion detection.

5.2 Agent Model

The proposed agent architecture is composed of two types of agents: *sensor agents* (SA) and *decision agents* (DA). These agents are grouped in small groups called *cells*. Each cell has a variable number of SA and one DA. SA collect and analyse local state information about the environment, this means that each SA has only partial observation about the global state of the environment. The local state is mapped to communications action-signals as depicted in Figure 5.1. The signals are received by the DA and, without any previous knowledge, DAs and SAs learn to assign them some semantics.

Using these communication signals, the DA tries to model the complete state of the cell environment. The semantics of the signals are not pre-defined, we cannot assign them a meaning because we do not know in advance what states are normal and which abnormal. It is only when SAs and DAs work together correlating information from multiple sources that intrusions are detected. Then agents can assign a meaning to the shared signals. In the last step of this process, the DA decides which signal-action to execute to a higher level agent outside the cell or in single cell environments the final action to trigger. In the intrusion detection domain, it triggers an alarm to the network operator.

Each agent uses a modified version of Q-learning (see Equation (5.1) below) to learn which action to execute in a specific state.

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha(r - \hat{Q}(s,a)) \tag{5.1}$$

The value of this function is the maximum discounted cumulative reward or the value of executing action a in the state s plus the value of following the optimal policy afterward. Nevertheless we have based our evaluation methodology using Q-learning, experiments using SARSA as an alternative update method did not show any significant difference in performance. The action selection strategy during learning is provided by Boltzmann exploration. We selected Boltzmann over ϵ-Greedy and random selection because of the control that we can have by executing random actions using a decay factor in the Temperature (T). We will show some empirical results in the use of these strategies in the next chapter.

To expand the number of agents we use multi-cell environments composed of cells of DAs. In multi-cell environments each DA inside the cells sends an action-signal to a central DA, which in turn can trigger a final action or send an action-signal to a higher level DA. When the top agent in the hierarchy triggers the action and this is appropriate to the goal pursued, all the agents in the cell receive a positive reward. If the action is not correct, all the agents receive a negative reward. The goal is to coordinate the signals sent by the SAs to the DA in order to represent the global state of the environment. After a certain number of iterations every agent must know the action that they need to execute in a specific state to obtain positive rewards.

In real life, SAs can be any network device that can collect network data. This

activity may be the primary goal of the sensor or it may be additional to its normal operation. Examples of devices are routers, Network Management Systems, switches, Network IDS, Host IDS, Firewalls, Netflow collectors, etc. This strategy provides heterogeneous input information in different forms such as SNMP data, Netflow traffic analysis data, Syslog messages, NIDS-HIDS messages, etc. The RL algorithm presented accepts these inputs as Q-tables or as more complex function approximation techniques such as tile-coding.

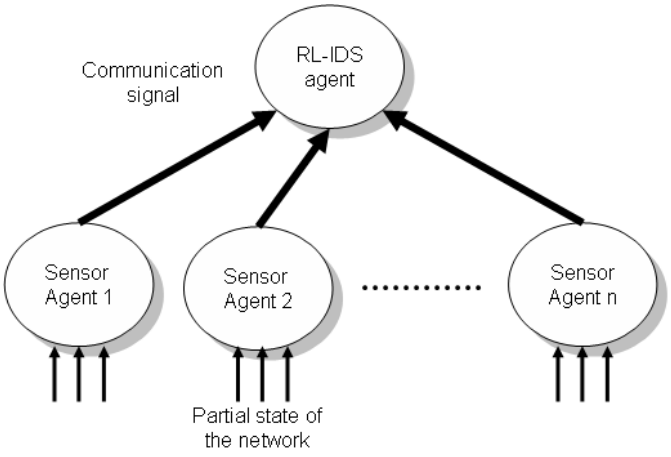
This is in brief how sensor agents collect local state data and how this information is summarized in communication-signals that are processed by agents with a higher level view of the environment. In the next following sections we will discuss in detail the operation of the agent architecture.

5.3 Basic Cell: The Basic Collaborative Model

SA and DA collaborate to learn how to detect malicious and non malicious network activity. The *Basic Collaborative Model* or *Basic Cell* (Fig.5.2) is composed of one central agent (DA) and n sensor agents (SA). Each sensor agent receives only partial information about the global state of the network. Then it maps this local state information to communication signals which they send to the DA of the cell (i.e. the signal constitutes the action of the sensor agent). Without any previous knowledge about the semantics of the signals, the DA agent tries to model the state of the monitored network. In turn it decides on an action that summarizes the global state of the environment, for example *normal state* or *attack*. If the signal or action is in accordance with the real state of the monitored network, all the agents receive a positive reward. If the action is inaccurate, all the agents receive a negative reward. The goal is that after a certain number of iterations of the algorithm, every agent would know for each state the action that they need to execute to obtain positive rewards.

Even though this is a simple scenario with few agents and states, it presents important challenges. Recall that stateless MARL with only two agents presents coordination problems under specific scenarios with multiple optimal joint actions or when miscoordination is penalized with high negative rewards [47]. Applied to the IDS domain

Figure 5.2: Basic Cell



we will not only have more than two agents, we will also deal with agents with several states which will complicate the coordination among agents. Related to this problem is the curse of dimensionality, in this case it is related to the exponential growth in the state-action space as the number of agents grows only linearly. In addition, as the number or states and agents increase so does the probability to converge to non-optimal joint actions. This pathology that makes agents converge to suboptimal joint actions is described by Panait, Tulys and Luke [131] and refered as *action shadowing* by Fulda and Ventura [67]. Besides, we have a special scenario here related to the joint actions. In all the previously analyzed MARL the joint action $a_{i,j,k}$ of agents i, j and k in state s is defined as the set of actions chosen by every agent in a certain time. In our case the joint action $a_{i,j,k}$ is the action a_l executed by agent l in reaction to the single actions a_i, a_j and a_k of agents i, j and k in any given state. Nevertheless, contrary to find these difficulties a barrier, we have found the IDS domain a very interesting case study to develop and propose some solutions to the coordination and scalability issues affecting MARL.

There is no defined number of agents per cell, however the maximum number of agents will be influenced by a number of factors that complicate their coordination. Among those factors are the number of variables that each sensor monitors and in con-

sequence the number of states, the value function estimation algorithm and possibly the exploration/exploitation strategy. In order to expand the total number of agents in the whole environment and to offer zone segmentation, we expanded the agent architecture to several hierarchical levels. The following section explains this architecture in detail.

5.4 Hierarchical Model

Before going into the details of the architecture it is important to clarify the distinction of our hierarchical architecture of agents using RL and *Hierarchical RL* [58, 133, 21]. Recall that Hierarchical RL aims to reduce the effects of the curse of dimensionality by splitting complex high level tasks into more simpler low level actions whist our approach uses low level hierarchical agents that collect local state information that is summarized and sent to higher agents in the hierarchy. As previously mentioned, to expand the sensor architecture to a large number of agents and to allow an administrative division of agents' groups we have created a hierarchical architecture. A schematic representation of this architecture composed of h levels is shown in Fig.5.3. This architecture is built of m cells with n agents per cell. The operation of the hierarchical model is stated as follows:

1. DAs are defined as $\mathrm{DA}_{i,j}$, where i is an unique identifier and j is the hierarchical level

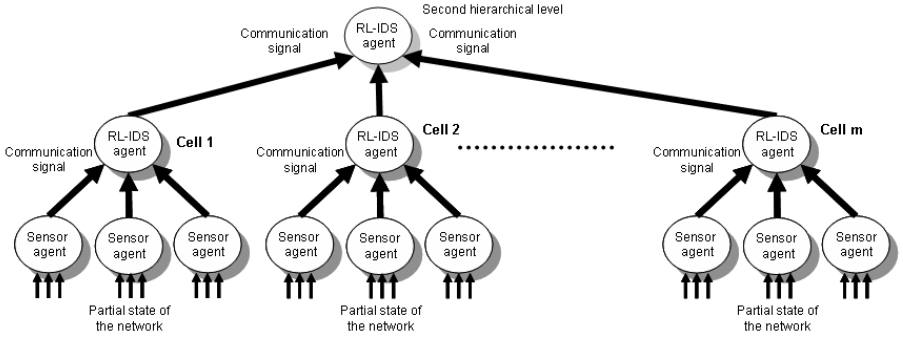
2. In this topology DAs in the first hierarchical level j=1 ($\mathrm{DA}_{i,1}$) receive local information from sensor agents and learn what signal to trigger to the next DA in the hierarchy ($\mathrm{DA}_{i,2}$).

3. Then, through the signals from the $\mathrm{DAs}_{i,1}$ and the reward function, the $\mathrm{DA}_{i,2}$ in the topology learns which signal to trigger to the network operator or to the next hierarchical level DA ($\mathrm{DA}_{i,3}$).

4. This procedure is repeated iteratively until it reaches the last DA in the top of the hierarchy (generally $\mathrm{DA}_{i,3}$).

5. This $\mathrm{DA}_{i,3}$ agent is responsible for determining the state of the whole system.

Figure 5.3: Hierarchical Architecture



There are some considerations that are important to state. The maximum number of agents per cell and the maximum number of cells connected to single $DA_{i,1}$ have been determined experimentally and we will detail this topic in Chapters 7 and 8. By using signals from local SAs to DAs to detect intrusion on a global scope we are creating a coarse state signal of all the states of the whole network. That means that a restriction of our proposal is that we cannot guarantee a converge to the optimal value function. The last agent in the DA hierarchy knows whether the state of the network is normal or not and it triggers the final action to the network administrator. Nevertheless, this top DA does not know about the local states in each cell. To provide local state visibility we have also proposed a local signaling to DAs in each cell. This capability is not mandatory for the operation of the algorithm, but it adds more practical functionality to the proposed architecture. This is so because the information provided to the network operator can now also include the location of local attacks and not just their existence.

5.5 RL of Signalling remarks

We have outlined in detail the functionality and operation of the RL for signalling algorithm, however there are some characteristics that are worth noting.

In the application of RL for signalling to the Intrusion Detection domain agents are not directly interacting or changing the environment. The only actions that agents execute are informative signals that do not directly change the state of the network.

Although the recipient of the signals (i.e. a network operator) can take an action using this information, changes in the environment are not immediate and the network operator could decide not to take further action even though there is an attack in progress. As result, in our implementation of Reinforcement Learning, the action selection of agents is based on the maximisation of the immediate rewards. We took this approach considering that agents are more interested in following a policy based on immediate rewards rather than one based on the value of a state.

The RL for signalling approach uses two types of agent architectures, the single cell and the hierarchical model. In both, input information is received and processed by sensor agents and then it is fed forward to the next hierarchical level. In the next level or layer, the information is processed and it continues to upper layers or an output is generated. One important characteristic in this architecture is the existence of a delay between the times when the input information is received and processed by sensor agents and when the output action is generated by the top agent in the hierarchy. Input information is processed by SAs at time *t*, this includes sending a signal to the DA. However, it is not until time *t+1* that the DA is capable of processing the information from every SA and generate its own signal (to the network operator or to the next DA in the hierarchy). It will be important to consider this characteristic when analysing the results using this agent architecture.

It can be observed that the hierarchical architecture and operational model resembles a Multi-layer Artificial Neural Network (ANN). Although the operation seems very similar as well there are important differences. In ANNs the output of each node has to be forwarded to the next layer in synchrony. In our practical case this it is not possible all the time. Nodes are dispersed and there is no guarantee that their outputs would reach their destinations, less that they would arrive in proper time to be processed. Besides the impracticalities that synchronisation process implies, the missed or outdated information from nodes may cause a negative impact in the learning or classification process of the ANN.

Contrary to ANNs, RL for signalling agents can process signals asynchronously. Agents are programmed to send signals at regular periodic intervals, however if a signal is lost, upper layer agents can use the last received signal from an agent whose communication with has been lost. We will show in our experimentation that the learn-

ing and classification processes are minimally affected by the lost of synchrony in the communication signals.

5.6 Conclusions

In this chapter we have illustrated our MARL signaling approach and how it can be applied in the intrusion detection domain under conditions of partial observability, restricted communication and global rewards. The approach is based on flat and hierarchical architectures of sensor and decision agents to detect normal and abnormal network states generated by Flood-Based DoS an DDoS. In previous chapters we have stated the need to use a variety of source information to accurately detect abnormal network states, however for some network scenarios this may be difficult to bandwidth and processing limitations. One advantage of our approach is that we are able to extract normal and abnormal states with low bandwidth utilisation, low central processing requirements and using a rich variety of source information required for high accurate intrusion detection.

We have described two basic architectures of agents: flat and hierarchical. The flat architecture is also referred as *Cell* and it is the basic group of agents. Hierarchical architectures are created by different levels of cells connected to DAs. By means of the hierarchical architecture we have the possibility of increasing the number of agents independent of the limitations imposed by MARL architectures. Moreover, the hierarchical approach permits the physical division of sensors into *administrative zones*. Although these zones are not important from the RL stand point, they are useful in practice to visualise the attacks' sources and targets.

MARL presents a diverse set of challenges, for example its scalability to large number of agents and large state-action spaces. As previously mentioned, because of these challenges, we think that the IDS domain is an interesting case study to apply Reinforcement Learning to a real world scenario and to propose new mechanisms to scale up solutions based on MARL. In the next chapters we will present the evaluation methodology applied, the evaluation of the presented agent's architectures in what we have defined as the abstract environment simulator and finally the set of tests and results in a more realistic environment supported by the network simulator NS-2.

Chapter 6

Research Methodology

One important problem when designing Intrusion Detection Systems or a new security mechanism against computer attacks is that there is no universally accepted methodology to test and to evaluate them [8]. This lack of a common methodology and test data to evaluate IDS complicates comparison between approaches. Although there are some resources publicly available [105, 52], it is common that researchers select their own customised methodology to evaluate IDS [8, 142].

For this research work we have selected a simulation methodology to develop and evaluate the hypothesis previously stated. Specifically we used two different simulation environments, one working as a high level abstraction of a network and another with more capabilities to emulate real network behaviour.

This chapter presents the description of these simulation methods, the proposed metrics and the criteria that we employ to evaluate our research work. The chapter is divided into three main sections. The first section explains the difficulties faced when evaluating intrusion detection systems and it describes the simulation environments that we use to evaluate our proposal. Section 6.2 proposes a set of metrics to measure how well our proposed agent architecture is capable of categorizing network activity. Finally Section 6.3 explains the evaluation criteria that we use to evaluate our proposal.

6.1 Simulation environments

Some authors describe the evaluation of intrusion detection systems as a complex process due mainly for the lack of a common methodology, tools, evaluation criteria and metrics [8, 142, 13]. The absence of a common methodology and test data to evaluate IDS complicates its comparison between approaches. One interesting effort, and probably the most important that has been made to evaluate IDS is *The DARPA off-line intrusion detection evaluation* [105]. This evaluation, founded by DARPA, evaluated several IDS projects and it was created by the Lincoln Laboratory at MIT. It produced a data set of background traffic (non malicious) interlaced with malicious activity that was used in the evaluation. This data-set is made up of synthetic traffic produced by hundreds of emulated hosts, IP address and automata process that generated traffic along with intrusions in the form of e-mail messages, telnet sessions, HTTP and FTP file transfers. Although this data set has been widely criticized it has been extensively used to test several IDS systems projects [93, 26, 53]. Further work of the Lincoln Laboratory resulted in LARIAT: *Lincoln adaptable real-time information assurance testbed* [147]. This system has the ability to generate different types of attacks at different data rates, to verify attack success or failure rates and score the IDS performance. Unfortunately this system is not available to the open community. A variation of the data set produced by the *The DARPA off-line intrusion detection evaluation* is the *KDD Cup 1999* [52] set. This data set is also publicly available and it was used in the 'The Third International Knowledge Discovery and Data Mining Tools Competition' and in the 'KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining'. The competition aimed to build a NIDS with a predictive model capable of distinguishing between *bad connections* or attacks and *good* or normal connections.

Despite the existence of several standarized data-sets, there is no a standardized methodology in how to evaluate an IDS. Today the *customised* methodology is prevalent among intrusion detection researchers. Examples of these methodologies are:

1. **Testing Laboratory:** IDS are evaluated using a closed network inside a controlled environment or laboratory. Background traffic is injected in the network using pre-compiled sets or by specific appliances such as traffic generators. Attacks are injected in the network using specific appliances such as traffic generators or us-

ing pre-compiled traffic from sources such as the *KDD Cup 1999* and IDSwakeup. [8].

2. **Testing Laboratory and customised attack tools:** Instead of using pre-compiled attack traffic, researchers create their own attack profiles using a variety of tools such as hping, iperf, Flame Thrower and Fragrouter. For some authors this is considerered the most accurate way to test IDS [8, 142]

3. **Simulation environments:** A simulation environment is generally used to evaluate IDS in complex environments involving large networks. Background traffic and attack patterns can be injected using pre-compiled sets or by creating specific patterns according to the evaluation requirements.

Although evaluating IDS using a test network seems to be the best method, it is not always possible due to time and resource constraints. Specifically in our case, we found it very difficult to use this method to evaluate DIDS. The research described here aims to develop an intrusion detection system capable of recognising and categorising DoS and DDoS attacks. As stated in previous chapters, in order to do so we require a distributed architecture of sensors. This architecture composes a large number of sensor agents located in a diversity of places within a computer network. To build this network we required costly resources such as network devices and links that were not available for this research. For this reason it was infeasible to construct an evaluation network composed of real devices. Instead we opted to evaluate our research in a simulation environment. In summary, we justify the use of a simulation environment instead of an evaluation network composed of real devices and real traffic because of the following reasons:

- **Speed:** Tests on a simulated environment will be faster than in a real network. Tests in a simulator can be performed in a fraction of the time length that it would be required in a test network. For example tests that simulate five minutes of network activity can be processed by the simulator only in a few seconds. Furthermore, launching a test in a DIDS can take several minutes or hours just because of the set up process. In the network simulator this process is done by scripts and it takes almost no time.

- **Resource Constraints:** To test scalability and the learning capabilities of the system to its maximum capacity it will be necessary to add large numbers of agents with different capabilities, several cells and hierarchical levels. Furthermore to test the interoperability of the DIDS in inter domain scopes (e.g. simulate inter domain Internet interconnections) we will need several SA and DA. In both cases we do not have all the hardware necessary to build these complex networks.

We used two simulated environments to develop this project. The first environment works as highly abstract simulation of a network. In this abstract simulation, agents, networks status and events are represented by abstractions of real behavior. Examples of these abstractions are the level of CPU utilization of a router agent in a specific time and the matching of a flow pattern in a flow collector. The second simulation environment is a network simulator with more capabilities to emulate real network behaviour and to process network connections. We will describe these environments and their use in the following sections.

6.1.1 Abstract Environment

The abstract environment worked as an initial test bed to evaluate the RL algorithms for coordinating agents. Under this environment we evaluated combinations of different agent topologies, numbers of agents per cell, hierarchical levels, exploration strategy and values of RL constants such as learning rate α. The base RL algorithm that we designed was evaluated with this simulation; this included the evaluation of different value function estimation algorithms (e.g. Q-Learning) and exploration/exploitation strategies. This simulated environment is an ad-hoc abstraction of a real network programmed in C#. It does not perform many of the tasks of the real networks but it is capable of performing the tasks that we need to initially evaluate our proposed agent architecture and the use of RL.

The simulation supports different types of agents and they can execute a diverse number of possible actions. Each agent is an object and can represent any network element including a router agent, a network IDS agent, a host IDS agent, an anomaly traffic agent (Netflow based) or a central agent. The simulation can hold several agents, each one of them can have different capabilities or they can be of the same type. We can arrange different agent architectures, how central agents can interact on different

hierarchical levels and we can also inject a range of attacks and faults into the network. Faults and attacks are emulated by changing the global state of the network. This simulates the different traffic and utilization patterns found in real networks under normal and abnormal conditions. The global network state is partially observed by each agent according to its capabilities.

Each test in the simulator is constructed with different types of elements. These elements are the number and type of agents, number of actions, agent topology and the fault or attack type. The simulation has the following parameters as inputs: α (learning rate), γ (relative value of delayed versus immediate rewards) and number of iterations to perform. The simulation also allows us to modify the exploration-exploitation strategy such as Boltzmann or Epsilon Greedy. In both cases we can modify the values of the temperature decreasing factor and epsilon (ϵ) respectively. The outputs of the simulation are the Q-tables of each agent and a final report. The final report holds information about the learning of the agents and it allows us to observe the behaviour of every agent in a specific iteration. Through this report it is possible to know the number of false positives and negatives in any step of the simulation. We will review further in this chapter some other metrics that are interesting for evaluating the performance of the intrusion detection capabilities of the agent architecture.

This abstract simulation environment has some flexibility to hold different numbers of agent per cell, hierarchical levels and it could generate different combination of synthetic attacks. Nevertheless it lacks capability to emulate real and complex attacks required to completely evaluate our research proposal. To overcome this problem we used a network simulation environment that provides a much more accurate model of real world networks.

6.1.2 Network Simulation Environment

As we neeeded to test a distributed IDS environment with several agents we opted to use a network simulator instead of a test network with real devices. To effectively evaluate our approach using a network simulator we require it to have specific characteristics that minimize the differences with an evaluation using a real network. The simulation environment at least must comply with the following criteria:

1. It must provide a realistic model of at least: end nodes (host computers), routers,

data links, queuing methods, delays, packet lost and the TCP/IP stack.

2. It must provide an emulation of network applications and protocols such as FTP, HTTP, VoIP, etc.

3. It must also provide interface to add source code to implement new capabilities to current objects

4. We must able to inject pre-recorded network traffic for future extensions to this research.

5. It must support a large number of agents and links on a diverse set of topologies such as bus, star, ring, etc.

We evaluated two network simulators OPNET [130] and NS-2 [182]. OPNET stood for Optimized Network Engineering Tools, the simulation product is a discrete event simulation engine that provides models of protocols and vendor device models. It is an object-oriented modeling and hierarchical environment. It provides an open interface for integrating external object files, libraries, and other simulators. It also included a GUI-based debugging and analysis module. NS-2 is an open-source discrete event simulator that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. NS-2 provides an accurate model of end nodes, routers, data links, queuing methods, the TCP/IP stack and packet loss and delay. Because it is open source it is possible to use it freely and we can modify it to include our own code to interact with the NS-2 engine, to create new agent types and to modify the behaviour of the already included agents in NS-2. Also, it is considered the de-facto standard in the academic world for network simulation and it has a large user base which makes it easier to learn how to use it and troubleshooting it.

After some evaluation of the described criteria that the network simulation needed to comply and combined by informal interviews with NS-2 and OPNET users, deep analysis of features available and use of both network simulators, we conclude that NS-2 was the best platform to use. There were different characteristics from NS-2 that persuade us to use it. The first one was the accurate model of network resources that it provides. Using these models our sensors are able to collect a diverse set of network information such as queue sizes and packets transmitted in each node's links, TCP windows sizes

and *Round Trip Times* (RTT) among others. Contrary to OPNET, NS-2 is open source. This allow us to modify the code to add our own agents and libraries. Finally, the large user base and the support that we could get through out online resources such as blogs, forums and mailing lists was another decisive factor.

In the network simulator we created different network scenarios varying the number of nodes, the network, topology, the traffic patterns and the type of attacks. To simulate the background traffic and the attacks we create our own profiles instead of using a pre-compiled set such as the KDD Cup 1999 The DARPA off-line intrusion detection evaluation. There are some important reasons that made us choose to design our own profiles over these sets:

1. The KDD Cup 1999 and the DARPA off-line intrusion detection evaluation sets only contain a few sets of attacks that pose the requirements that we need to evaluate FBDoS attacks.

2. To inject the traffic from the KDD Cup 1999 and the DARPA off-line intrusion detection evaluation to the NS-2 simulator and to simulate DDoS we required processing power that our workstation did not have. This argument is also valid for the use of traffic generators such as hping2 and iperf.

3. Transfer rates are critical to evaluate our system. According to Athanasiades *et. al* [8], in the KDD Cup 1999 and the DARPA off-line intrusion detection evaluation set these data rates were never set up and they can be in the order of kilobits per second.

4. We found that creating new attacks or background traffic in NS-2 through scripting was easier and faster than using the data sets or traffic generators (hping2 or iperf).

6.2 Evaluation Metrics

There are different ways to evaluate IDS, Ranum [142] states that in order to make a good evaluation the first step is to focus it on a quantitative or qualitative measures. Axelson [13] points out the complexity of evaluating IDS and comments on the use of the following criteria: *Effectiveness*, how well the IDS detects intrusions and avoids

producing false positives; *Efficiency*, that is how much computing resource and storage the system needs; *Ease of use*, this is related to the complexity to operate and implement an intrusion detection solution; *Security*, how good is the IDS to defend itself against attacks; *Interoperability*, this is related in how the IDS can work with other IDS as the architecture grows; *Transparency* or how the IDS can be deployed without disrupting the network environment and *Collaboration*, that is how the IDS can work with other devices to improve it the environment security. Another metric that is possible to use is *Cost* [155]. Cost is further divided into *Deployment* and *Monitoring* cost; deployment costs are related to the configuration of the devices, the design and tuning of the system, updating of rules and data storage among others. Monitoring costs are related to the alert analysis and the capacity of the device to produce low levels of FP and FN. High levels of FP require accommodate expensive human resources for incident analysis; cost related to FN are much harder to measure because of the number of unidentified security attacks leading to real security incidents is unknown and the cost of those incidents is difficult to quantify. From the list of metrics we can observe that effectiveness, efficiency and cost are quantitative measures while the rest are qualitative. In this research we are interested to see how an intrusion detection approach can be improved by adding more sources of information and an adaptive algorithm capable to detect new attacks from past experience. To evaluate this improvement we will use a quantitative approach using effectiveness metrics. From a holistic point of view we are also interested how fast our approach can learn to recognize attacks and how scalable it is. We will comment more about these criteria further in this section after introducing the intrusion detection metrics that will be used to evaluate our proposed IDS approach.

The most common metrics used to measure the detection performance of IDS are the *False Positives* (FP), the *attack detection rate* (IDR) and ROC curves relating the FP and IDR (Table 6.1). The FP rate is the fraction of the total alarms that do not represent an intrusion. The attack detection rate is the fraction of the total number of alarms that were identified as intrusions, this rate is also referenced as *precision*. Another metric is *False Negatives* (FN), FN are the fraction of the total number of intrusions that were not categorized as intrusion, however this metric is not very common to measure IDS in real networks. The *receiver operating characteristic* (ROC) curve is used to intuitively show the relation and trade-off between the false positive rate and the attack detection

Table 6.1: Performance Metrics 1

Measure	*Formula*	*Meaning*
False Positive Rate	FP / (TP + FP)	The fraction of alarms that do not represent an intrusion
False Negative Rate	FN / (TP + FP)	The fraction of the total number of attacks that were not categorized as intrusion
Intrusion Detection Rate	TP / (TP + FP)	The percentage of total number of alarms that were identified as intrusions
Events	TP + TN + FP FN	The total number of events

rate. ROC curves show the intrusion detection rate as a function of the false-alarm rate, points are plotted using the number of false alarms in the x-axis and the intrusion detection rate in the y-axis. Although it is widely adopted, ROC fails to provide accurate measurement in scenarios where the probability of an intrusion is very low [13].

To assist us in the design and evaluation of this proposal, we have also introduced the use of the prediction metrics shown on Table (6.2). These metrics are commonly used in bioinformatics and machine learning [61]:

1. **Precision:** This metric has been already discussed and it is also referred as IDR.
2. **Recall:** We specifically introduce this metric because ROC variables do not show the number of malicious events that the IDS fail to categorise as negative instances (TN) (no-attacks). To verify that the IDS is learning how to detect attacks this variable is important to observe.
3. **Accuracy:** In a similar fashion than recall, accuracy relates all the variables together to offer an intuitive idea of the performance of the IDS system in relation to the number of correct events categorised.
4. **Specificity:** This metric relates the number of no attacks events with the number of false alarms.

It is important to mention that all the described metrics will produce unclear results where the probability of intrusion is very low.

Table 6.2: Performance Metrics 2

Measure	*Formula*	*Meaning*
Accuracy	(TP + TN) / (TP + TN + FP + FN)	The percentage of positive predictions that is correct
Precision (IDR)	TP / (TP + FP)	The percentage of negative labeled instances that was predicted as negative
Recall	TP / (TP + FN)	The percentage of negative labeled instances that was predicted as negative
Specificity	TN / (TN + FP)	The percentage of predictions that is correct

6.3 Evaluation Criteria

To verify that our IDS approach is properly capable of detecting and categorising Distributed Denial of Service Attacks at inter and intra domain scopes we evaluated it using a specific set of criteria and we also compared the results to other IDS approaches that we emulated. The set of criteria is composed of *Intrusion Detection Capabilities*, *Adaptability* and *Scalability*.

The Intrusion Detection Capabilities criterion evaluates the detection of known attacks and it uses intrusion detection metrics. We intend to maximize accuracy, precision (IDR) and recall while minimizing the false positive rate. This criterion also measures the coordination capabilities under the assumption that a well coordinated group of agents detects known attacks with low rates of false positives and high levels of accuracy, precision and recall. By means of adaptability and re-training mechanisms our approach will be capable to identify unknown attacks. The criterion Adaptability is related with the detection of new or previously unobserved attacks. This criterion evaluates how the function approximation and re-training techniques correctly generalise or classify new events. Finally the Scalability criterion evaluates how well the approach is scaling up in the number of agents. To evaluate scalability targets we use a rate relating the number of agents in a cell and the intrusion detection metrics.

A direct comparison to existing IDS was very difficult because the lack of common

criteria to evaluate them. Other works that were addressing similar problems to ours used their own evaluation methodology and they did not provide enough details to reproduce the evaluation tests. We chose to compare our approach with two hand-coded approaches, in each case using the network simulator as the evaluation environment. With these approaches we tried to mimic the functionality generic IDS and to compare them with our evaluation conditions and criteria. The first hand-coded approach emulates a misuse IDS. In this case the IDS is looking for the patterns that match an attack in the same way that some commercial misuse IDS do in real world networks. Examples of these devices are Snort [162] and Checkpoint [41]. Our hand-coded approach is a simplified version of these devices that emulates the process of checking for a signature of an attack and triggering an alarm if there is a match. For this case only a reduced set of signatures related to packet flow analysis is used. To evaluate a more complex IDS implementation we also developed a hybrid approach that uses misuse IDS and behavioral traffic analysis to detect intrusions. This second hand-coded approach integrates the same variety of input information as our learning algorithm and it is similar to the mechanisms employed in some commercial Intrusion Protection System (IPS) such as the Cisco Intrusion Prevention System Sensor [46] that search for intrusions through signature and anomaly detection methods. In the hybrid approach, the signature module uses the same mechanism than the first hand-coded approach described. The anomaly detection is provided by a set of thresholds that monitors the same variety of information. Details of these thresholds and the information monitored will be addressed in Chapter 8.

6.4 Conclusions

In this chapter we have discussed the evaluation methodology followed in our research. This includes the simulation environments that we use to test our proposed agent architecture, the difficulties to compare and evaluate the performance of intrusion detection systems, and finally the metrics and the evaluation criteria used.

Due to a diverse set of constraints that prevent us from evaluating of our system using real network devices in real environments we opted to use two simulation environments. The first simulation environment is the abstract environment. It works as a test bed

to test the principles of our MARL approach and we evaluate combinations of different agent topologies, number of agents per cell, hierarchical levels, exploration strategies and values of RL constants such as α and γ. After finalizing our RL evaluation, we continued testing our learning approach using the network simulation environment. We opted to used NS-2 as the network simulator. NS-2 is an open-source discrete event simulator that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. NS-2 provides an accurate model of end nodes, routers, data links, queuing methods, TCP/IP stack, packet loss and delay. Using this environment we evaluate our proposed intrusion detection against denial of service attacks under more realistic conditions.

The evaluation and comparison of intrusion detection systems is known to be a problem. The evaluation of IDS may be based on qualitative or quantitative criteria. For this research our evaluation strategy is based on quantitative parameters denoted by the set of metrics chosen to evaluate our system. This set of metrics is composed of Accuracy, Intrusion Detection Rate, Recall and Specificity. The intrusion detection evaluation criteria were defined as Intrusion Detection Capabilities, Adaptability and Scalability. The Intrusion Detection Capabilities criterion evaluates the detection of known attacks and it is related to the maximization of the intrusion detection metrics. The criterion Adaptability is related with the detection of new or previously unobserved attacks and it evaluates how the function approximation and re-training techniques correctly generalize or learn new events as positive or negative. Finally the Scalability criterion evaluates how well the approach is scaling in the number of agents per cell and in the number of states per agent. Using these metrics and criteria we will be able to measure the intrusion detection capabilities, the adaptability and the scalability of the intrusion detection presented.

Chapter 7

Abstract Environment

The abstract environment is the initial test bed for evaluation our ideas in how to develop an intrusion detection engine based on RL. Further we use this environment to apply our RL approach to different agent architectures varying the number of agents per cell, the exploration-exploitation strategy, the different values of RL parameters, the number of states per sensor agent and the distribution of attacks as input information. In these initial experiments, we fall back on an idealised model of a network that posseses the principle learning and coordination challenges of the real-world case.

This chapter describes the different tests performed, the obtained results and how these results were used to provide better and more realistic experiments in the network simulator. The chapter is divided into four sections. The first three sections report the results using a single cell architecture. The last section shows the evaluation results of a hierarchical architecture. Section 7.1 presents the first results using a simple architecture of two agents in one cell. Section 7.2 presents the results and methodology followed to achieve coordination among the agents, Section 7.3 explains the problems and the evaluation tests performed to scale the agent architecture. The last section presents the methodology and the evaluation tests of our proposed agent architecture using two and three hierarchical levels.

Table 7.1: State Matrix: Two Agents

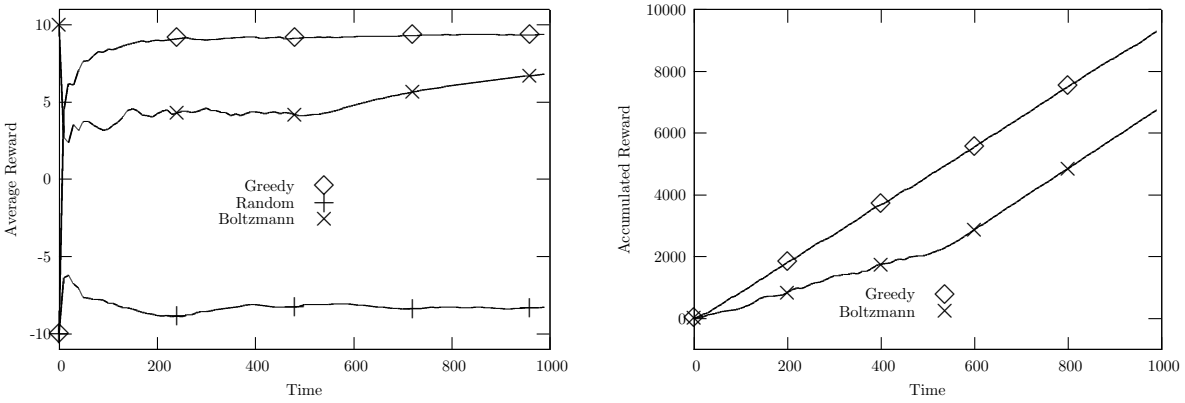
State	Agent 1	Agent 2
1	A	A
2	A	NA
3	NA	A
4	NA	NA

Table 7.2: Game Matrix for joint actions

		A2	
		A	NA
A1	A	10	-10
	NA	-10	-10

7.1 Initial Tests

The first test is a proof of concept with two agents. Each agent is capable of executing two actions and there is no communication between agents. The goal of this test was to verify the capabilities of the simulation and some basic concepts of learning in multi-agent systems. The agents are a router agent and a network IDS agent. The actions are *alarm* (A) and *not-alarm* (NA). The simulation generates four global states for the network. State 1 represents a DoS attack, in state 2 the simulation has normal heavy traffic. State 3 denotes a minor scanning of the network and it does not affect the performance of the network. State 4 of the simulation represents normal traffic load without scanning. State 1 is perceived by the router as a high CPU utilisation use and by the NIDS as a scanning in the network. Both agents must execute the action alarm. For the remaining states, the agents are trained to execute the action NA. As shown in Table 7.1 the result should be that just one agent will alarm on states 2 and 3, and none in state 4. The game matrix of Table 7.2 shows the rewards of the joint actions of the state 1 of the simulation. Agent 1 (A1) is the router agent and Agent 2 (A2) is the NIDS. Other states show similar matrixes but with different rewards.



(a) Average Reward (b) Accumulated Reward

Figure 7.1: Cell two sensor agents

Each agent uses a modified version of Q-learning (see Equation (5.1) in Chapter 5) to learn which action to execute in a specific state. The value of this function is the maximum discounted cumulative reward or the value of executing action a in state s plus the value of following the optimal policy afterwards. We test two agents under three action selection strategies: completely random, Epsilon-greedy and Boltzmann. A comparison of these three strategies is shown in figures 7.1(a) and 7.1(b). With a random selection of actions, the agents are unable to learn when to execute the proper action. This behaviour can be easily observed in the average reward graph on Figure 7.1(a). The Q-tables and the final report generated by the abstract environment also show this behaviour and the high rate of false alarms. Epsilon-greedy seems to reach convergence faster and it seems to outperform the Boltzmann exploration-exploitation strategy in terms of learning rate. In this case ϵ is constant and the strategy never converges to zero false alarms. On the other hand Boltzmann has more false alarms in the beginning due to high exploration, but when the temperature drops the false alarms fall to zero. Recall that Boltzmann exploration uses a decreasing factor (T) known as temperature to slowly decrease exploration over time. A similar method can be applied to ϵ-greedy by letting ϵ to drop to zero over time. As observed, in both cases agents are able to converge to the optimal joint actions.

Next, we tested whether the agents would perform well as the training data changes

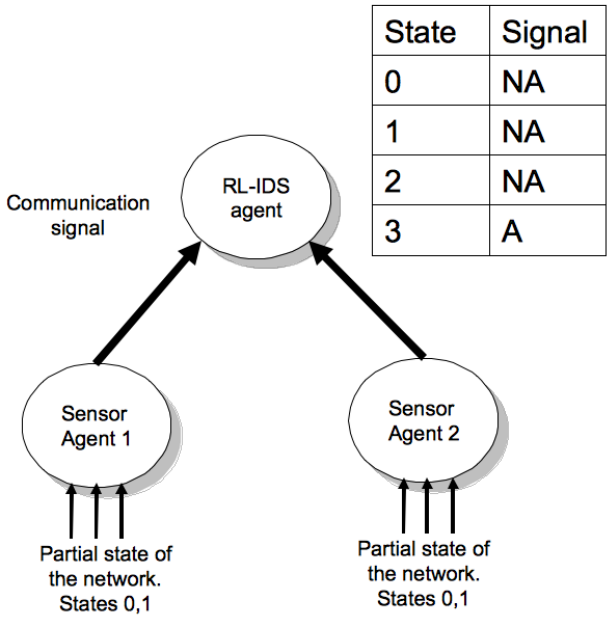
Table 7.3: Game Matrix for joint actions in Cell with two Agents

State	SA-1	SA-2	DA	R	DA*	R*
0	X	X'	A	10	NA	-10
1	Y	Y'	NA	10	A	-10
2	X	Y'	NA	10	A	-10
3	Y	X'	NA	10	A	-10

and the number of agents or states grow. The simple test using the cell environment (Figure 7.2) included two SA and one DA. We also set up the SA to have two states (0 and 1). Again, note that each SA cannot observe the states of other sensor agents and that the combination of all sensor agents' states represents the global state of the network. In this simple cell scenario we have four global states where state 1 represents an abnormal network state that would require an alarm signal from the DA. The sensor agents have to learn to produce the right signal action to the DA, while this agent needs to learn to interpret these signals. As shown in Table 7.3, the action-signals generated by the SA are unknown in advance (shown as X, X', Y and Y'), the goal of the DA is to learn which signals from the sensor agents represent a normal state of the network or a warning state. When the DA takes the right action (i.e. A in state 0 or NA in state 1-3) it receives a positive reward, when it does not (A in states 1-3), it receives a negative reward. SA-1 and SA-2 are rewarded with the same reward as the DA that takes their signals.

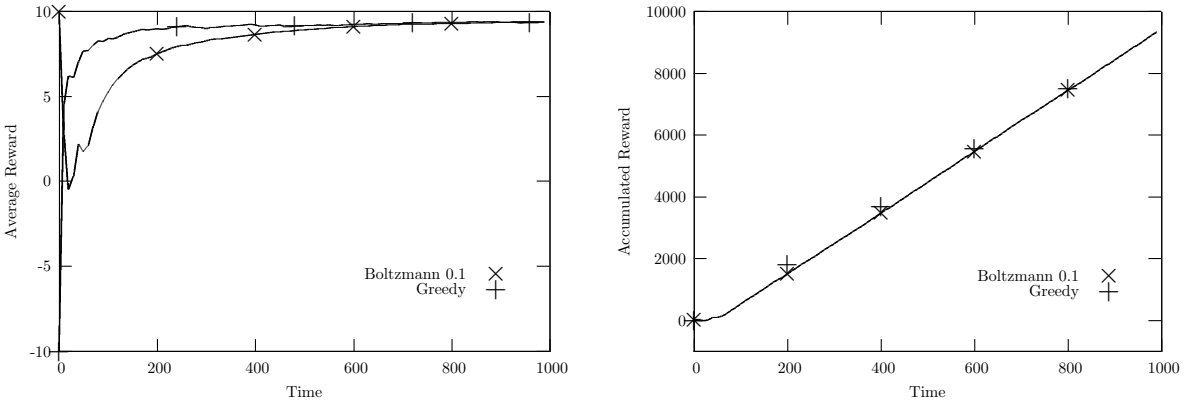
The result of the test is shown in Figures 7.3(a) and 7.3(b). The figures depict the average and accumulated reward respectively of the DA during learning using Boltzmann and Epsilon-greedy as the action selection strategy. As can be seen the negative decay factor β in the Temperature parameter (Eq.7.1) plays an important role in how fast the DA is able to converge to high reward values. The reason for this is the high exploration compared to exploitation for low values of β during most part of the simulation. For Epsilon-greedy we used a similar strategy to encourage exploration in the beginning of the simulation and exploitation at the end.

Figure 7.2: State Description: Two SA and one DA



$$P(a) = \frac{e^{\beta Q(s,a_n)/T}}{\sum_0^i e^{\beta Q(s,a_i)/T}} \tag{7.1}$$

As we will discuss later, the amount of exploration versus of exploitation will be important to bias the agents to coordinate their actions towards better expected rewards. The graph of figures 7.4(b) and 7.5(a) show how the intrusion detection metrics are affected by the number of agent and the number of states. As noted, the more the agents advance in the simulation the better the values get. In the last steps of the simulation the SA agents know what action-signals to employ to signal the DA to trigger or not an alarm. However as we increase the number of states per sensor agent from two to four states, the value of the intrusion detection is affected negatively. Further in this chapter we will analyse the negative effect in performance as the number of agents or states increases.



(a) Average Reward (b) Accumulated Reward

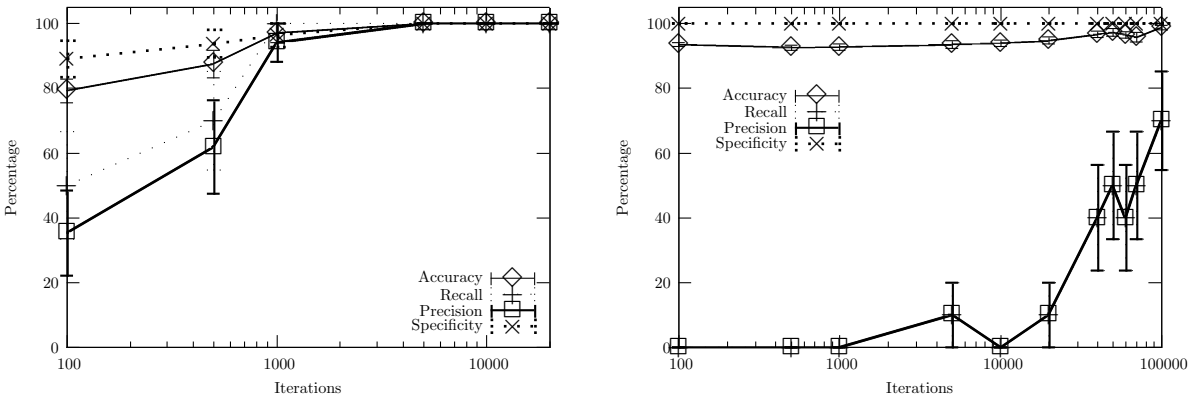
Figure 7.3: Two Agents, Boltzmann

7.2 Coordination tests

Further tests using the abstract environment are mostly related to solve scalability issues and the coordination problems with a large state-action space and many agents. In the next section we will detail our tests aimed at evaluating the maximum number of agents and states that can generate acceptable performance values in terms of intrusion detection metrics. We will show our results of evaluating the singe cell scenarios and the hierarchical architecture with different number of agents and different exploration/exploitation strategies.

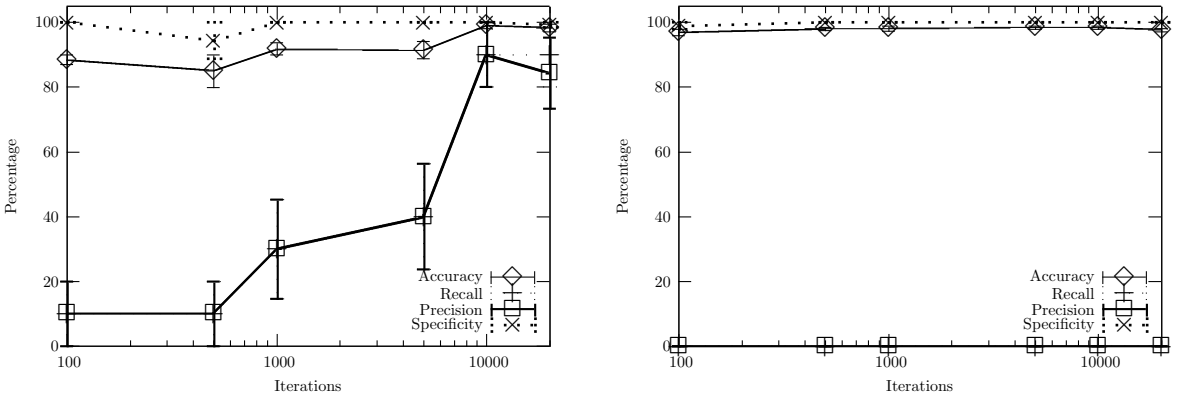
7.2.1 Attack Bias

To evaluate the agent architecture with more agents inside a basic cell, the first step was to increase the number of SA to three. Each SA was configured with two states. The additional sensors implicitly increase the number of states of the decision agent; in general this state space is the product of number of sensor agents and the number of action-signals of each sensor. Figure 7.5(a) shows the intrusion detection performance of one cell with three sensor agents and two states per agent. As observed, the agents require more learning time to achieve similar performance than the cell with two agents. The metrics that are more affected are precision (IDR) and recall. The graph of figure



(a) Two States (b) Four States

Figure 7.4: Intrusion Detection Metrics, Two agents

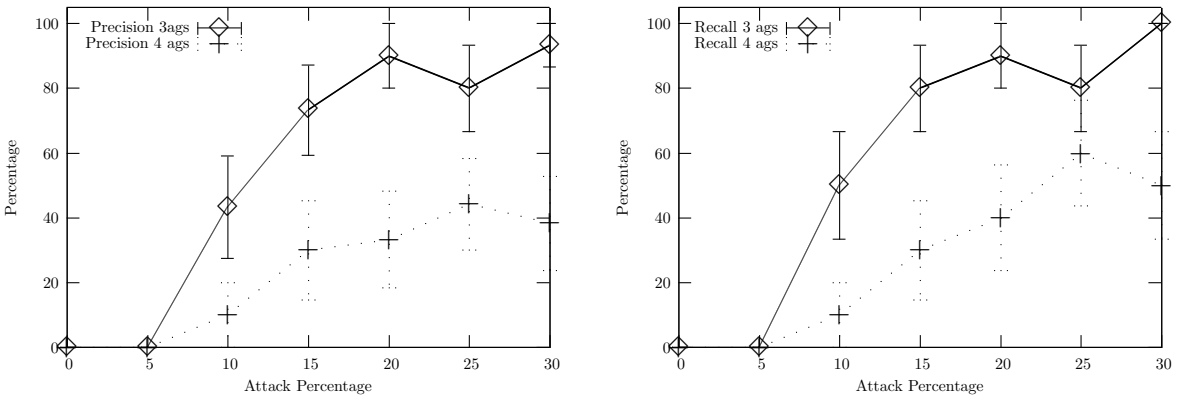


(a) Two States (b) Four States

Figure 7.5: Intrusion Detection Metrics, Three agents

7.5(b) shows the same metrics but applied to a cell with three sensor agents with four states each. As can be seen, the agents are completely unable to detect attacks.

In the abstract environment the global state of the network is simulated by randomly choosing between normal and abnormal states. As the number of agents increases, the number of abnormal states is relatively small compared with the number of normal ones. This distribution of training data biases the agents to learn that the safer action is not



(a) Precision (b) Recall

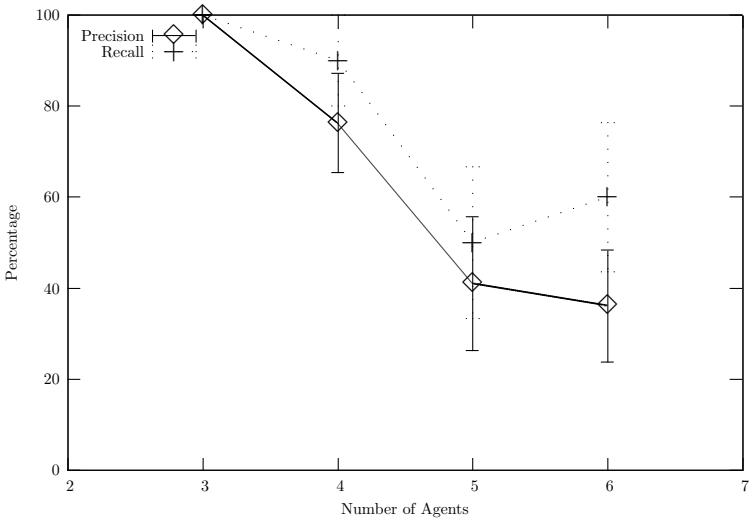
Figure 7.6: Total Attack Percentage

to generate any alarm action at all, that is the joint action with the best discounted reward over time is to signal a no-attack even though there is one. To address this issue we provided more attack training examples to the agents. We varied the percentage of attacks examples provided in the training set from 0% to 30% using cells of 3 and 4 sensor agents. We fixed the number of iterations to 20,000. Figures 7.6(a) and 7.6(b) show the precision and recall values of these tests. As it can be observed, as we increased the percentage of attacks, the overall value of the metrics increased. However depending the number of agents the optimal value seems to vary from 30% to cells with 3 agents and 25% of attacks for cells with 4 sensor agents. Other metrics not shown are not affected.

7.2.2 Exploration

The sole effect of adding more attacks examples was an improvement on precision and recall for the cell of three agents. A similar effect was found for the cell with four sensor agents, however the improvement in the value of the metrics was lower. To evaluate how precision and recall are affected as we increase the number of agents we evaluate our architecture using three, four, five and six sensor agents. Each agent has 2 states and we provided a minimum of 25% of attacks in the training set. As it is shown in Figure 7.7, as the number of agents increases, precision and recall decrease.

Figure 7.7: Precision and Recall by number of agents

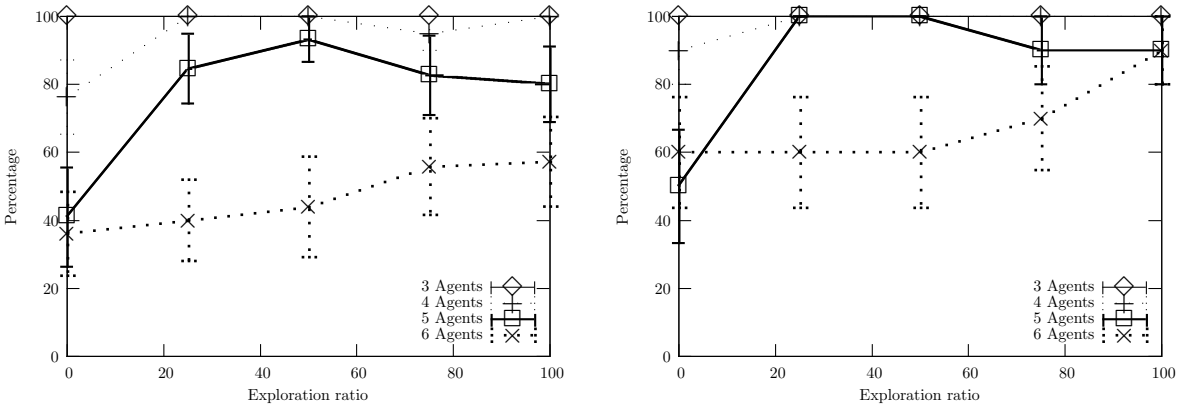


We supposed that the agents had little time to explore all the no-attack states and to *fix* the Q-values that were miscalculated as result of the credit assignment problem and the partial observation of the environment. To tackle this problem we extended the exploitation phase of the exploration/exploitation strategy to allow the agents to exploit actions and to modify the values of their Q-tables. To carry out this task we divided the exploration/exploitation strategy into two sections.

The first part was the initial Boltzmann strategy where agents slowly decrease exploration over time accordingly to a decreasing factor (T). The second part was a total exploitative strategy where agents do not explore actions any more. In this second stage agents execute the action with the best expected outcome without any random selection. We denoted this as a *pure exploitive strategy.* The level of *pure exploitation* or *exploration ratio* is given by Equation 7.2.

$$Exploration\ ratio = \frac{Number\ of\ pure\ exploitive\ iterations}{total\ of\ iterations} \tag{7.2}$$

To evaluate how different values of exploration ratio affects the learning performance we run some test varying the number of agents. Figures 7.8(a) and 7.8(b) demonstrate



(a) Precision (b) Recall

Figure 7.8: Exploration ratio

how the exploration ratio has more impact in the intrusion detection metrics as the number of agents increases. For the cell with three agents the exploration ratio has no impact at all; however for cell of four, five and six agents the intrusion detection metrics perform better as the exploration ratio increases to some optimal value. The exploration ratio that produces optimal intrusion detection metrics values varies depending on the number of agents. For example for cells with four agents an optimal value is reached at around 20% of exploration ratio, lower values represent lower performance and higher values produces no change. For cells with five agents a 50% value seems to represent the optimal point, lower or higher values produces lower values in the measured metrics. Finally for cells with 6 agents the optimal point is reached with an approximate value of 80% in the exploration ratio, lower values represent lower values and higher values represent no change. Contrary to cells with four agents, cells with five and six agents never reach 100% in any of the intrusion detection metrics. This issue is discussed in the next section.

7.3 Scalability tests

One important constraint in MARL is the complexity to scale these systems to large numbers of agents or state-action spaces. To evaluate how these factors affect the

Table 7.4: Three States

State	CPU Utilisation
0	low
1	medium
2	high

intrusion detection metrics and in general how fast and well the agents learn to detect and categorise normal and abnormal network states we ran tests varying the number of agents per cell and the state-action space per agent. We have shown how the number of agents affects the coordination of action among agents, next we will show how a similar behaviour appears as we increase the number of states of each agent.

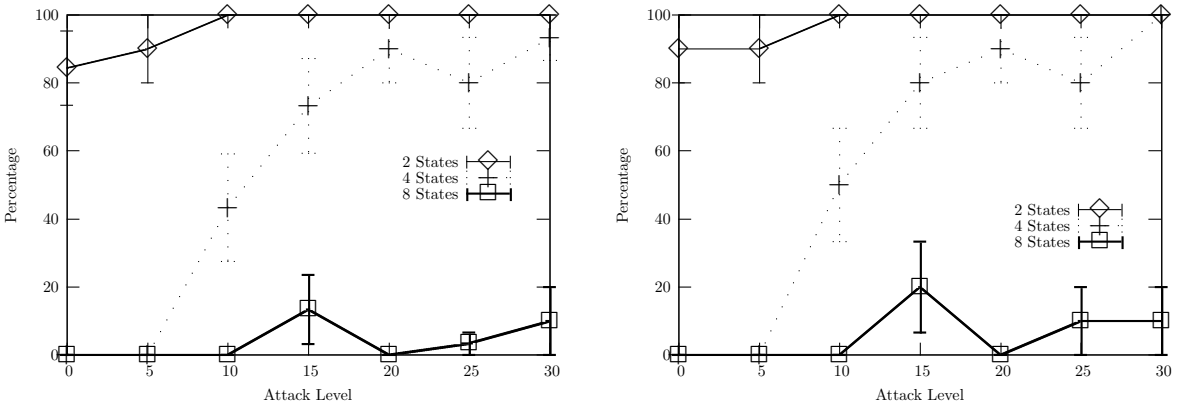
The state space is an important part of RL because it represents the environment observations of the agents. In this case we are using simple tables to represent the value function and each state represents a different environment observation. For example, using only two states in a router agent that monitors its CPU's utilisation we can map the variable as $lowUtilisation = 0$ and $highUtilisation = 1$. The number of states is related to the resolution of the observed variable or feature. Increasing the number of states can give us more resolution of a single variable (Table 7.4) or it can relate multiples variables (Table 7.5). Consequently, it is possible that more states generate better learning results.

Using several features with high resolution is an important issue in RL. In principle it may represent better the environment, but the state space growth is exponential. The enormous state space could make the learning converge very slowly or not to converge at all. To evaluate how the agent architecture copes with the growth of the state-action space we ran a set of tests varying the number of states in each sensor agent. We have already shown the impact to increase the state space in tests with two and three agents (See figures 7.4(a), 7.4(b), 7.5(a) and 7.5(b)). As observed, the number of states impact was more conclusive in the tests using three agents. In this test, agents with 4 states never learn how to detect and categorise abnormal behaviour.

To increase the performance of the intrusion detection metrics in cell with few agents but large state spaces we followed the same approach used when we increased the number of agents per cell. First we increased the attack rate until reaching 30% of minimum

Table 7.5: Four States, two variables

State	CPU Utilisation	Memory Utilisation
0	low	low
1	high	low
2	low	high
3	high	high

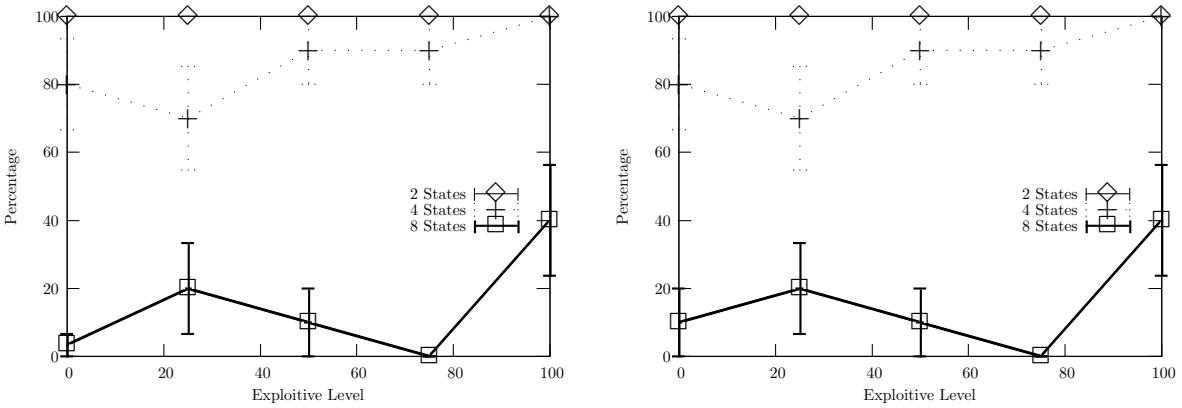


(a) Precision (b) Recall

Figure 7.9: Attack Level

attack examples in training. As shown in Figures 7.9(a) and 7.9(b) as we increased the attack rate the performance of the metrics was even better. For cells using 2 and 4 states it was possible to get close to the 100% of performance. For the case of 8 states it was just a small improvement. The next step was to fix the attack rate to a value and then modify the exploration ratio to encourage exploitation at the end of the learning process. We selected a value of 25% of minimum attack examples using cells with three agents with 2, 4 and 8 states. As can be observed in figures 7.10(a) and 7.10(b) the case with 2 states per agents has reached its maximum performance and the exploitation level represents no change. However for the cases with 4 and 8 states there is an improvement. The improvement was more evident for the case of 4 states.

The achievement of higher values on the intrusion detection metrics and learning rates is negatively affected as the state-action space and number of agents and the state-



(a) Precision (b) Recall

Figure 7.10: Attack Level

action spaces increase. This negative impact is the effect of the previously mentioned problems often found in MARL such as credit assignment, partial observation, curse of dimensionality and mis-coordination penalised with high negative rewards discussed in Chapter 3. To effectively apply MARL to solve practical problems it is important to address these scalability related problems. To address the the problem of scaling to a high resolution and high dimensional state space we will use function approximation techniques. We will show related tests in the next chapter using the network simulator. Aiming to address the misscordination problem as the number of agents increases we introduce a hierarchical approach. In this architecture sensor agents learn to coordinate their actions inside a group called cell. Consequently and in a hierarchical fashion groups of cells learn to coordinate their actions to detect and categorise abnormal activity on a global scope. In the following section we will present the results of the evaluation of this approach.

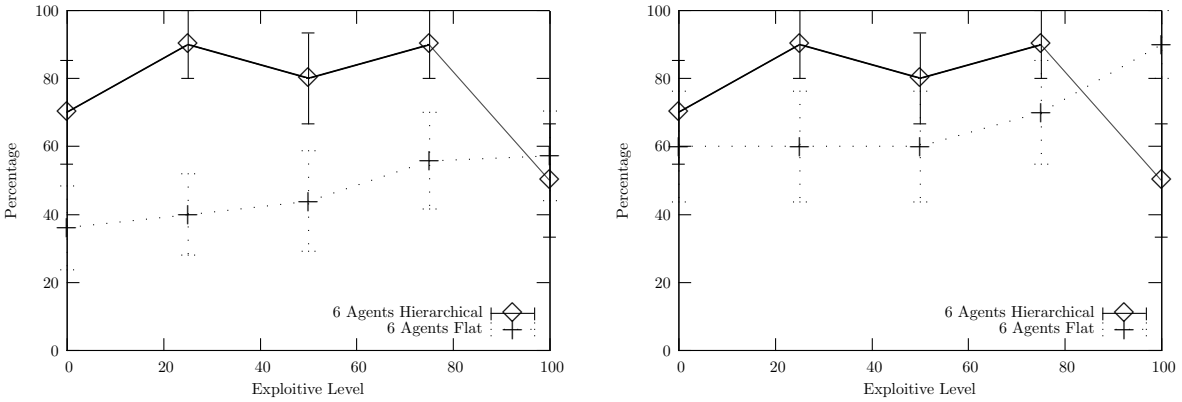
7.4 Hierarchical Architecture

In Chapter 5 we introduced a hierarchical agent architecture (See Figure 5.3). This approach has two purposes. Its primary goal is to increase the number of sensor agents that can be distributed along the network helping to address the scalability problem

regarding the number of sensor agents. As a secondary aim, the hierarchical approach lets us use our architecture to detect abnormal activity on inter-domain networks or in intra-domain networks with geographical zones. Recall, in this new architecture Sensor Agents (SA) and Decision Agents (DA) inside a cell learn how to identify local normal and abnormal activity. The DA learns the semantics of the action-signals sent by the sensor agents (SA). Once they have learned to coordinate their actions, the DA inside the cells send communication signals to the next DA in the hierarchy. This agent in turns learn the semantics of the signals sent by the DAs in the lower level. This procedure is repeated iteratively until it reaches the last DA in the topology, i.e. the agent responsible for determining the state of the whole system.

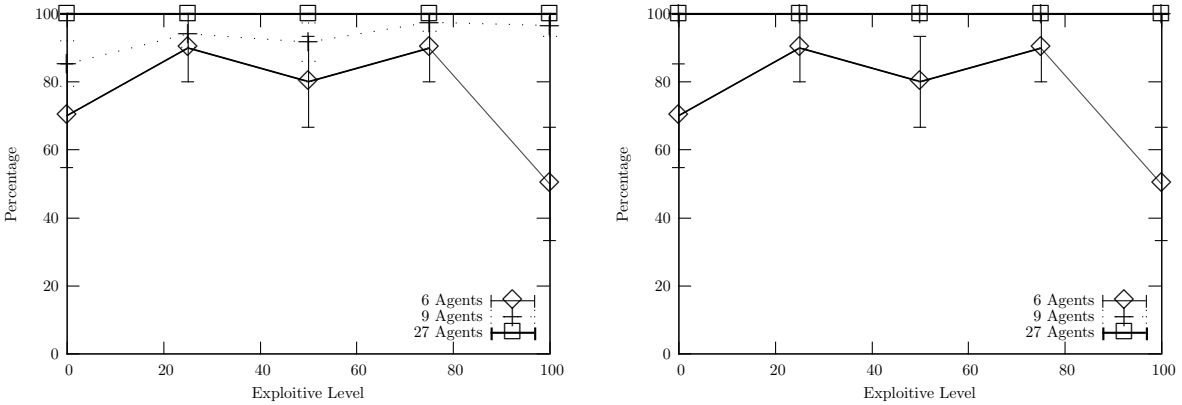
The basic experiment was to compare the performance of 6 agents using this hierarchical topology with the flat approach of 6 agents in one cell. The hierarchical architecture is composed of two cells with three SA and one DA each. This architecture has two hierarchical levels, this means that it has one DA on top of the cell architecture and one top level DA which is in charge of triggering the last action to the human manager of the network. For these experiments we ran 20,000 iterations, a 25% of minimum attack training examples and we varied the level the exploration ratio. The resulting comparison between the six agents flat and hierarchical architecture is shown in Figures 7.11(a) and 7.11(b). For both intrusion detection metrics precision and recall, the hierarchical architecture displays better performance. The only exception is when the agents follow a 100% exploitive strategy where the flat architecture show better results. This may be the effect of an over-exploitation by the agents; a similar case is found for the five agents architecture shown in figures 7.8(a) and 7.8(a) .

Figures 7.12(a) and 7.12(b) shows the intrusion detection metrics for a cell with nine SA, three DA and one global DA. Similarly to the case with six agents, the architecture shows acceptable levels in all the intrusion detection metrics. The graphs also show our expansion of the architecture to three hierarchical levels in a larger architecture. This large architecture is composed of 27 SA and 13 DA (40 agents in total). For all the tests shown in these figures we used at least 25% of attack training examples as we did previously. The results presented support our proposed architecture to expand the number of sensor agents without losing performance in the learning rate or in the intrusion detection metrics.



(a) Precision (b) Recall

Figure 7.11: Flat vs. Hierarchical



(a) Precision (b) Recall

Figure 7.12: Hierchical Architecture

7.5 Conclusions

In this chapter we have presented our experiments and results of evaluating the proposed agent architecture using an abstract network environment as the basis of our simulation. We have shown results using single cells architectures varying the number of agents per cell and the state-action space of each agent. Analysing these experiments we have found that the number of attack examples and the exploration/exploitation strategy play important roles in increasing the learning rate and the performance of the intrusion detection metrics. In other words, these characteristics in the training data and in the exploration/exploitation strategy let the agent learn which actions lead to better rewards.

We also found that an increase of the number of agents per cell or the agent's state-space reduces the performance of the agents. To increase the performance under these circumstances it is necessary to apply a minimum of attack states and an exploration/exploitation strategy focused on more exploitation in the last steps of the simulation. Although these mechanism raise the agents' performance, they are not enough when the cells hold a large number of agents or when the agents require a large state-action space.

To address the scalability problem related to the number of agents per cell, we introduced a hierarchical architecture of agents. In this architecture, after a DA learns to interpret the signals from SAs, they start to coordinate their actions with a higher layer of DA. The last DA in the hierarchy is in charge of triggering the action signal to the network operator. Using this approach we were able to scale up to 27 SA and 13 DA in three hierarchical levels without any sacrifice in the agent's coordination capabilities. These sets of experiments showed how our proposed architecture addresses the scalability problems related to the number of agents.

Despite the positive results shown in our evaluation there are still some unanswered questions about how to move the architecture forward to a more realistic environment. The first question is how to represent the value function with the high feature resolution required to accurately represent the environmental information received by sensor agents. The second question is related to how the agents are able to use the value function representation and the learning mechanisms to recognise unseen states and correctly categorise them. The third unanswered question is related to the learning approach.

So far the learning stages of the proposed architecture are learning and exploiting. In the learning phase the agents are presented with training data. Once they learn to communicate their actions, the agents stop learning and the exploitation phase starts. In the exploitation phase the agents do not learn any more. To this respect, we need to investigate how the proposed architecture deals with an online learning approach where agents never stop learning. Last but not least, the final open question is about the ability of the architecture to signal local states of cells. The current architecture produces only summarised information about the whole state of the network but it does not provide information about local states and attacks.

To answer these questions we evaluate our proposed agent's architecture using a more realistic simulation environment. Within this new simulation environment we evaluate not only coordination problems as the number of agents increases, but also the adaptation and online learning capabilities of the architecture to recognise and categorise new attacks. The adaptation is mainly provided by the function approximation layer of the learning process and the online learning capability is one of the key advantages of reinforcement learning over other machine learning techniques. The results using this new simulation environment will be explained in detail in the next chapter.

Chapter 8

Network Environment

In the previous chapter we used a highly abstract IDS scenario to test how a group of agents learn to interpret and coordinate their action signals to detect normal and abnormal activity. We proposed a hierarchical architecture of agents composed of groups of agents or *cells*. This environment gave us the opportunity to test the basic feasibility of the agent learning architecture using an abstract simulation containing simple network agents. Nevertheless, the question of how the approach would work with more complex and realistic network topologies, traffic patterns and connections remains open. To evaluate our learning architecture of agents and to add elements and the complexity of real applications, in this chapter we use the network simulator NS-2, a specifically designed library for NS-2 and the *Tile Coding Software* [171].

Within this new simulation environment we evaluate how agents are coordinate their actions whilst their number in the system increases, we also evaluate the adaptation and online learning capabilities of the architecture for recognizing and categorizing new attacks. The adaptation capabilities needed to recognise unseen normal and abnormal states. The online learning capability is one important feature of reinforcement learning.

This chapter presents the evaluation experiments of our agent architecture under realistic conditions. The chapter is divided in four sections. Section 8.1 presents the basic parameters required to run our simulation tests in the network environment. These initial parameters were based on our evaluation tests performed in the abstract environment. The next three sections present our findings using tile coding and RL in the network simulator. Each section includes findings related to tests evaluating basic learn-

ing features such as traffic changes and adaptability. Also, in each section we outline the limitations of the presented approach. Section 8.2 reports results using only flow information in a single agent architecture. Section 8.3 describes the evaluation tests using multiple sources of information in a single cell. The last section presents our findings in the use of a hierarchical architecture to expand the number of sensor agents. In this section we analyze the cases for online learning where agents never stop learning and the generation of local information to enhance the reporting capabilities of the architecture.

8.1 Initial Parameters

We have described in previous chapters the operation of the single cell and the hierarchical architectures; also we have shown how we used the abstract environment to initially evaluate the proposed MARL Signaling algorithm. The evaluation includes the Reinforcement Learning related parameters such as the learning rate, the discount factor, the exploration/exploitation strategy and finally the scalability of our learning approach. We now use there results as the foundation for more complex testing in the network simulator. We summarize these findings as follows:

- We evaluated a variety of scenarios using ϵ-greedy and Boltzmann as exploration/exploitation strategies. We found that in general Boltzmann yielded better results and faster learning in our agents.

- We found that following a uniform distribution of possible states in tests with more than two agents creates a very small number of abnormal states compared with the number of normal states. This distribution of training data caused the agents to learn that the safer action was not to generate any alarm-action at all. To solve this problem we provide a minimum of 25% of abnormal states in the training data.

- To increase the Intrusion Detection Rate we modified the exploration/exploitation strategy to add more exploitation. A percentage of the exploration/exploitation was Boltzmann and the rest was a complete exploitation strategy. We found that exploitive levels above 0.5 and lower than 0.8 produced on average the best results.

- When we increased the number of agents the levels of precision went down due to high rates of false negatives. In order to keep the number of false negatives low, we determined that maximum number of agents per cell should be less than 6.
- We developed a hierarchical architecture of agents to increase the number of sensor agents that we can use. Another advantage is that the architecture is adaptable to detect abnormal activity on inter-domain networks or in intra-domain networks with geographical zones. This led to good results with a large number of agents using up to three agents per cell and up to three hierarchical levels.

We used these pre-defined parameters in the initial set of experiments using the network simulator. In some experiments we modified some parameters such as the exploitation level and the RL parameter *alpha*, in all these cases we will explicitly point out the modifications.

8.2 RL and Flow Information

To initially evaluate our learning approach and architecture using a function approximator, we chose the use flow data as a single source of information to detect Flood-Based Denial of Service Attacks. As we previously pointed out in Chapter 2, Flood-Based Denial of Service Attacks change the normal data flow of data in the attacked network. Analysing the changes on the features that are affected by these events can lead us to identify when the network is under attack. To detect those anomalies, data flow information has been used by different approaches to detect Flood-Based DoS and DDoS. Methods using data flow information can accurately detect the data patterns of abusive behaviour as generated by these attacks [98, 160]. The main drawback of all these approaches is that they usually require complex fine-tuning to provide the correct values of the monitored features and thresholds. This characteristic makes the use of flow information an interesting problem to solve using our proposed learning architecture. We want to evaluate how our learning algorithm using a function approximation technique is capable of interacting with the environment and learning when the flow data corresponds to a normal transmission of data or when it corresponds to an attack. To make this evaluation we developed a single agent architecture that analyses flow data to

categorize flows belonging to normal and abnormal network states using Reinforcement Learning and tile coding as a function approximator.

To process the flow information we followed the algorithm described in Figure 8.1. In the initial state, the agent initialises the weight array $\vec{w}$ of each action a to 0. At time t the agent analyzes each flow; for each flow the agent takes the values of the n monitored features, i.e. protocol, port, average packet size. To process the flows, the agents use a subroutine (as part of the tile coding library [171]) that uses as input the selected features, the number of tiles, and the divisions per tile (tilings). The output of the subroutine is a vector $\vec{w}$ containing the index of the weights activated by the flow. Using this information the agent sums the activated weights in each action and following a *greedy policy* (no exploration, only exploitation) the agent executes the action with the highest sum. In our experiments we used only two actions, the first action is to send a communication signal tagged as *Alarm* when a flow identified as anomalous is detected. The second action is to send a communication signal tagged as *No-Alarm* when the flow is from normal traffic. Additionally, the greedy action selection has been replaced by a Boltzmann strategy where agents initially explore actions and they start to exploit (greedy policy) them using a decay factor T.

After triggering the communication signal, the agent receives a reward r_{t-1}, that is the reward for the action for the same flow executed in the previous iteration (on $t-1$). The reward is positive if the action was correct, it is negative otherwise. Finally the agent updates the activated weights for the current flow according to Eq. 8.1 and proceeds with the next one. After all the flows are processed the agent waits *timestep* seconds and samples flows again. To finish it stores the values of the $\vec{w}$ and the $V\pi(s)$ of each flow and action for future use. To process each input with tile coding we used 32 tilings, each tiling was divided into 16 tiles giving as result an array of 512 tiles. We used this set up for all the tests evaluated using the network environment. The total number of weights varied according to the number of inputs processed, in the majority of the tests we used at least three input variables (i.e. drops per queue, packets in a queue, TCP window size, etc.) in each sensor agent.

Figure 8.1: Flow Processing with Tile Coding and RL

- At time t=0 initialize for each action a the vector $\vec{w}$ and the value function $V\pi(s)$
- While *time* != *end* do;
  - for i=0 to *lastflow* do;
    - Get tiles according to number of features and m divisions per tile
    - Sum weights $\vec{w}$(t) to obtain the value function of the action
    - Follow greedy policy to select action
    - Receive reward r_{t-1} Update $\vec{w}$ according to:

$$w_i = w_i + \alpha * (r_{t-1} - Target) \tag{8.1}$$

      where:

$$Target = \sum \vec{w}_{(Bestaction)} \tag{8.2}$$

  - Wait *timestep* seconds
- Store $V\pi(s)$ of each action and $\vec{w}$

8.2.1 Basic Experiments

We ran a series of tests to find out whether the use of flow information along with RL and tile coding could enable agents to learn to categorise normal and abnormal activity in the network using flow information. The network simulator was configured to use a simple network topology of 4 nodes as shown in Figure 8.2. Node 0 generates normal FTP-alike traffic while node 1 produces normal UDP traffic. Node 4 is an attacker producing a flood of UDP traffic. Node 2 is the *Reinforcement Learning Agent (RLA)* that must learn to differentiate the normal traffic from node 0 and 1 from the abnormal behaviour of node 4. Node 3 is the node under attack and receives valid data from nodes 0 and 1 and it as well. The node parameters set up by this test are shown in Table 8.1

To train the RLA we generated normal traffic flows from nodes 0 and 1. To simulate a real network we randmly started and stopped the flows. The attack activity was emulated by node 4 at specific times during the training. After the agent had learnt we performed a diverse set of tests under different traffic conditions, different

Figure 8.2: Network Topology on Testing

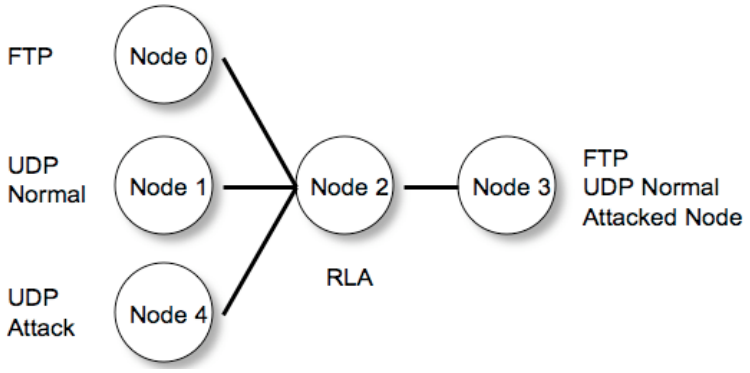


Table 8.1: Node Parameters

Node	*Protocol*	*Port*	*PacketSize*	*Rate*
Node 0	6 TCP	21 FTP	variable	variable
Node 1	17 UDP	5080	512 Bytes	16 kbps
Node 4	17 UDP	1580	110 Bytes	1 Mbps

flow information and using a different number of features to analyse. We applied the learning algorithm to the tests using 2 and 3 features. As shown in Table 8.2 we used the numerical value of protocol and port as features 1 and 2. Packet size was feature number 3.

For the first test, we added more sources of FTP, UDP no-attack and UDP attack

Table 8.2: Features

Number of Features	Features
2	Protocol number and Port number
3	Protocol number, Port number and Packet Size

Table 8.3: Tests Results

Test	Features	Precision	Recall
Traffic Pattern	2	100%	100%
Traffic Pattern	3	100%	100%
Attack Port Changed	2	0%	0%
Attack Port Changed	3	100%	100%
Protocol and Port Mimic	2	0%	0%
Protocol and Port Mimic	3	100%	100%
Protocol, Port and Packet-size Mimic	2	0%	0%
Protocol, Port and Packet-size Mimic	3	0%	0%

combined with changes in the sending patterns of the applications and attackers. We referred this test as *Traffic Pattern*. Using two and three features the RLA had no problem at identifying and categorising the flows as shown in Table 8.3. Although the agent was trained using only the flows generated by nodes 1, 2 and 4 it was capable of recognising the new flows generating by the new sources of traffic.

8.2.2 Adaptability Tests

To simulate how a real world attacker will try to bypass firewalls by changing information in their attacks, we created a new test where we changed the port used by the attacker. We called this test *Attack Port Changed*. For this test we changed the attack port from the original 5080 to 6665. As well as in the previous test, the RLA had no problem in recognising the attack using three features. Still, using only two features the agent was not capable to recognise the abnormal activity. In a more complex masquerade attack test we tried to simulate how an attacker would try to hide its attack by using the same protocol and port of a valid application, we identified this test as *Protocol and Port Mimic*. The two feature approach was again unable to distinguish between the attack and the no-attack flows. Nevertheless, the use of three features allowed the RLA to identify the attack, in this case throughout the packet size. This raises the question: What happens if the attacker has the ability to modify the application to use the same packet size, protocol and port as a valid application?

To answer this question we developed the tests referred as *Protocol, Port and Packet-size Mimic*. Similarly to the previous test we set up the protocol and port numbers to be the same as a valid application. Furthermore simulating a complex masquerade attack we modified the packet size of the attacker traffic. In this case both, the two and three features approaches were not able to identify the attacker flows. It is important to point out that this does not mean that flow information is not useful for detecting attacks or that tile coding and RL are not suitable solutions either. It just exemplifies that as any other approach, these techniques may not be one hundred percent dependable for all sort of scenarios. Using the last test as example we can see that the agent does not have a way to differentiate normal and attack flows because all the features are the same. In order to detect these hidden flows it is necessary to explore more complex detection implementations, possibly using more than one source of information.

8.2.3 Findings and Limitations

We performed tests using a function approximation in the form of tile coding to detect anomalous flows of traffic. Tile coding provides a high resolution and multivariable data without state-space explosion. The use of flow information, RL and tile coding has yielded some positive results along with some limitations in our specific domain of traffic anomaly detection. We showed that the RLA learned how to categorise flows of known applications and how it has learned how to generalise some features to identify unknown activity. On the other hand, it cannot categorise flows under complex attack scenarios.

In order to enhance the detection capabilities of our RL agents and to achieve our final goal of detecting flooding-based DoS and DDoS we plan to add more capabilities to our approach. Interesting information to follow is the *Round Rrip Rime* (RTT) of the flow, connection creation/termination (e.g. TCP three way handshake, SIP signalling) and the analysis of aggregated flow information. The next section will present our work working to improve the intrusion detection of relative complex attacks using a variety of information sources in a multi-agent system environment.

8.3 Single Cell with Multi-Agent architecture

The use of a hierarchical agent architecture lets us identify and categorize simulated Flood-Based DoS and DDoS under an abstract simulation environment as described in the previous chapter. Nevertheless we observed that the system was able to detect attacks and normal states using a large number of agents, we were unable to evaluate this architecture using a large state space, which is an important requirement to address real world problems. To access a large number of features without exploding the size of the state space we propose the use of tile coding as a function approximator and we evaluate it using a single agent architecture and flow information to detect and categorize attacks. We discover that the approach performs well for simple attacks but it did not for complex scenarios where the attacker mimics valid flows to hide its attack.

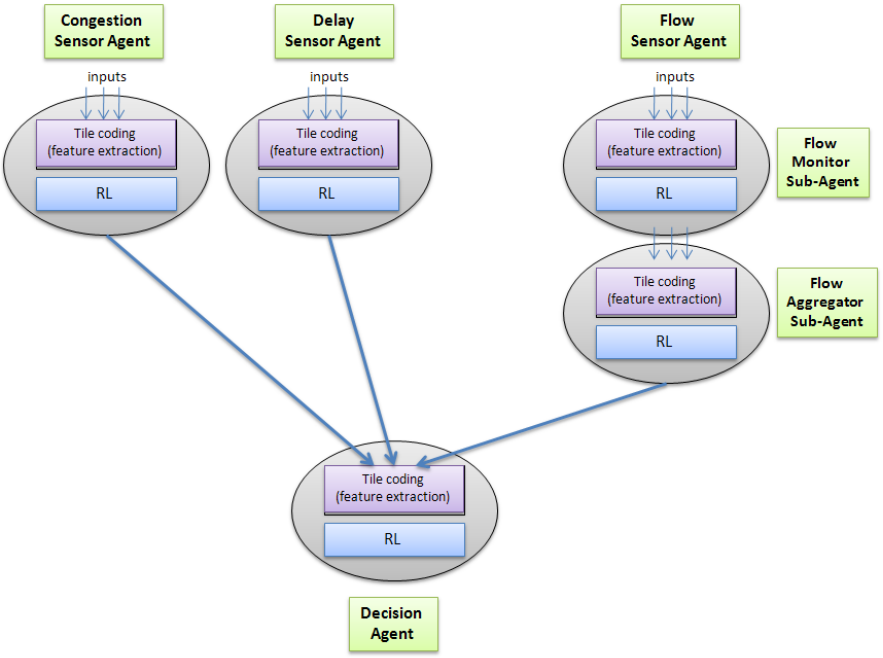
8.3.1 Agent Architecture in the Network Environment

After we have reviewed the performance of a single agent architecture in the past section, we concluded that the multiple source of information had significant potential for improving the ability of IDSs to detect complex attacks. To address this situation we proposed an agent architecture similar to the one previously used in the abstract environment but enhanced with the ability to use a large input space. This new ability to use several inputs with high resolution will allow us to adapt this architecture to the requirements that a real world application needs. To accomplish this task we have slightly modified the original agent architecture used in the abstract environment (See Figure 8.3).

The basic architecture is composed of a single cell with a Congestion Sensor Agent (CSA), a Delay Sensor Agent (DSA), a Flow Sensor Agent (FSA) and the Decision Agent (DA). The variety of information sources is required to improve the detection capabilities [22, 112, 117]. The idea here is that each sensor agent perceives different information depending on their capabilities, their operative task, and where they are deployed in the network [154].

One important reason to require a MAS approach instead of a single agent with multiple sources is that not all the features are available at a single point in the network. Flow and congestion information may be measured in a border router between the

Figure 8.3: Agent Architecture



Internet and the Intranet whilst delay information may be only available from an internal router. Besides, Flood-Based DDoS attacks are launched from several remote controlled sources trying to exhaust a target's key resource. A stand-alone IDS does not have all the information to accurately identify sources and destinations of DDoS attacks when the sources use address spoofing to hide their identity.

The CSA analyses link information on a particular node in the network. Specifically this agent samples link utilisation in bytes per second, the size of the queue in packets, and the number of packets dropped by the queue. This set of monitored information (link utilisation, queue size and packets dropped) will be referred as *feature domain*. A good placement for this agent to collect data is in the path between the protected service and the untrusted network. The closest this agent is to the protected service, the more data it could gather. The DSA monitors TCP connections between nodes. DoS and DDoS attacks modify the normal behaviour of the network in many ways. Some of these changes can be spotted by analysing TCP information from connections in the path of the attack. This agent has the same internal structure as the CSA but

its feature domain is different. The features analysed for the TCP connections are: the average number of ACK packets received, the average window size, and the average *Round Trip Time* (RTT).

The task of the FSA is to analyse and summarise flow information. To perform this task the agent is divided into two logical sub-agents: the Flow Monitor (FM) and the Flow Aggregator (FA). The FM analyses the traffic flows that pass through the FSA. This sub-agent can be hand-coded or a learning agent. If it were hand-coded, it would use a table with the flow information that represents an attack in a similar fashion to a misuse IDS. For the tests that we perform we use a learning agent using RL to learn which flows are abnormal. The feature domain of the agent is composed of protocol number, port number, and the average packet size of the flow. Using this information the learning agent acting as FM learns which flows are normal traffic and which ones may lead to an attack. The second sub-agent is the FA. It aggregates flow information by keeping a flow table with the signals reported by the FM. The basic feature domain of the FA is the number of attack flows reported by the FM. It is possible to hold more information such as the total number of flows, the number of no attack flows, the rate of attack flows vs. no attack flows, etc.

In the network environment we use the same Decision Agent (DA) that we used in the abstract environment. The DA did not undergo any modification in its structure, functionality or operation. The diagram of the Figure 8.3 shows a module called *feature extraction*, this module uses tile coding as function approximation technique to map input data to state information. The tile coding parameters used in tests based on this agent architecture were 32 tilings and 16 tiles per tiling. We used three input variables in each sensor agent.

The same RL of Signalling algorithm explained in Chapter 5 was developed using this agent architecture. Sensor agents (CSA, DSA and FA) send signals to their correspondent DA within the cell. The DA in turn generates an alarm or no-alarm signal to the network operator or an action-signal to the next level in the hierarchy. Again, it is important to mention that agents are not directly interacting or changing the environment, thus the action selection is based in the maximisation of the immediate rewards. The DA in top of the hierarchy obtains its reward from the network operator. If the categorisation of the network state was correct, the operator rewards positively

and negatively otherwise. This same reward is passed down to other DAs lower in the hierarchy and eventually to the SAs.

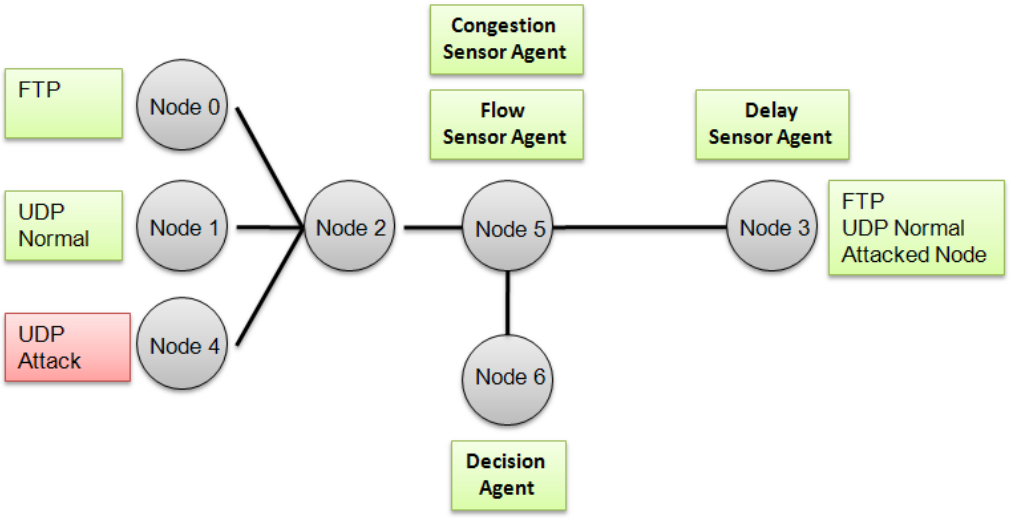
Both Figure 8.3 in this chapter and Figure 5.2 in Chapter 5 show an agent architecture that resembles a Multi-layer Artificial Neural Network. Also the operation looks very similar, input information is received and processed by perceptrons (sensor agents) and then fed forward to the next layer. In the next layer (DA) an output is generated. would imagine that this problem would be better addressed using a simple ANN. That would be true if all the inputs would be in the same location and the outputs of all the sensor agents (or perceptrons) to the DA were synchronised. In practice this is not possible. Sensor agents are located away from each other, they are not just gathering different information, but also they may be located in different places within the network. To deploy an ANN we need to synchronise the outputs of each perceptron to arrive at specific intervals to the next layer in the neural network. Such synchronising of signals may be very difficult to accomplish in noisy and congested environments such as a network under a DoS attack. In contrast, the approach using RL works with asynchronous signals and it is easier to deploy and more reliable under the loss of information from sensor agents. However it is important to consider the delay that exists between when the input information is processed by sensor agents and when the final action-signal is generated by the top DA in the hierarchy.

8.3.2 Basic Tests

To find out whether the agent architecture along with the proposed learning process are capable of detecting abnormal states of the network we performed a series of tests. We generated the network topology of figure 8.4 composed of 7 agents or nodes. Node 0 generates normal FTP-like traffic while node 1 produces normal UDP traffic. Node 4 is an attacker producing a flood of UDP traffic. Node 5 is logically divided in two RL sensor agents, one CSA and one FSA. Their tasks are to forward traffic and collect data about the network. Node 6 is the DA and it solely works as a RL agent. Finally Node 3 is the DSA, it receives valid data from nodes 0 and 1 and it is the node under attack as well.

We first train our agents to categorize basic normal and abnormal activity in the network. Following the same strategy as in the previous tests with the single agent

Figure 8.4: Tested Network



architecture we randomly start and stop connections from node 0 (TCP/FTP) and node 1 (UDP stream) to simulate normal traffic. Using another random pattern of connections we used node 4 to simulate the attacks to the network characterised by a flood of UDP traffic. At time $t = 0$ each one of the agents starts gathering information from the network and learning as previously explained. At time t_{final} we stop the learning process and store the values of the weight array $\vec{w}$ in order to use them in the evaluation tests. During these tests the agents are not learning anymore and are exploiting the knowledge acquired during the training. We have performed the entire set of tests using the feature domains for sensor and decision agents previously described and to measure the performance of the architecture we used the intrusion detection metrics already used to evaluate the agent architecture in the abstract environment. As a remainder, a low false positive (FP) rate indicates that alarm raised by our agents will not overwhelm the network operator. Low rates of false positives are reflected as well as high levels of precision and recall. A high value in recall also indicates that the agents are able to identify attacks while they maintain a low number of false negatives (FN) or unidentified attacks. Finally a high level of accuracy indicates that the system is capable of identifying attacks while generating few false positives. The intrusion detection rate (IDR) or precision is an important metric but it can be misleading given a certain type

of traffic (e.g. the IDR can be high when the system recognises few attacks but the number of FP is low).

The test *Traffic Pattern* aims to evaluate the adaptability of our architecture for changes in the traffic patterns. It considers an identical network topology as in training but with different traffic patterns. In this test we modify the start-stop times of the data traffic from the no-attack and attack nodes. This test is important because the changes in the traffic patterns could create false positives and false negatives. TCP connections have a flow control mechanism that tries to use as much bandwidth as is available without congesting the network. This behaviour causes a high link utilisation whilst the TCP connection is up, and sometimes generates packet dropping in the router queues during the bandwidth adaptation phase. Likewise, FBDoS and FBDDoS attacks tried to use as much as bandwidth as possible and can generate packet dropping in the router's queues. This similarity may make the categorisation of this traffic in different groups a complex task. Despite almost identical behaviour, FBDoS and FBDDoS attacks starve link bandwidth and generate more drops in queues during longer periods of time. Although in simple environments these differences can be spotted easily by humans, in complex environments the identification of the best variables to monitor and the correct thresholds values to reach in order to infer an attack is a complex task that requires a trained eye and hours of meticulous observation. In general, the resulting values of the metrics' performance of the test *Traffic Pattern* were interesting (See Table 8.4). The results show that the agents were able to coordinate their actions to recognize normal and abnormal activity with low levels of FP and FN.

8.3.3 Adaptability Tests

The tests *Attack Port Changed*, *Protocol and Port Mimic* and *Protocol, Port and Packet-size Mimic* were analogous to the previous tests on the single agent architecture using flow information as input. They were designed to create a more complex scenario where the attacker changes its attack to mimic authorised or normal traffic. *Attack Port Changed* simulates when the attacker changes the attack port to any other given port while in *Protocol and Port Mimic* we change the attack port to be the same as the authorised application. In test *Protocol, Port and Packet-size Mimic* we simulate when the attacker goes further and changes the attack port and the packet size to mimic the

Table 8.4: Tests Results

Test	Precision	Recall	Accuracy	FP Rate	FN Rate
Traffic Pattern	100%	99%	99%	0%	3%
Attack Port Changed	89%	97%	93%	11%	3%
Protocol and Port Mimic	96%	73%	86%	0%	20%
Protocol, Port and Packet-size Mimic	100%	29%	66%	0%	40%
Multiple UDP sources	100%	97%	93%	9%	4%
Multiple FTP sources	91%	97%	93%	9%	4%
Multiple UDP attack sources	100%	99%	93%	0%	4%

no-attack application. As can be seen in Table 8.4, all the tests show that the system is capable of categorise normal and abnormal activity. However in particular for the test *Protocol, Port and Packet-size Mimic*, the levels of intrusion detection, accuracy and recall metrics are lower than the rest. We explain these results below.

For the test *Attack Port Changed* the system seems to be a little bit noisy by generating some FP and it also missed some attack events. The precision or IDR for test *Protocol and Port Mimic* is surprisingly better, however this metric alone might be a little misleading as in this case. Even though the IDR is higher in the *Protocol and Port Mimic* than in the simpler test *Attack Port Changed*, the other metrics have poorer performance due to the high level of FN. In other words, the architecture is able to successfully distinguish valid and invalid traffic, but is not good at detecting abnormal traffic when the attacker hides the attack.

The results for the test *Protocol, Port and Packet-size Mimic* follow a similar pattern as the previously described test. The system can detect abnormal states without generating FP giving high values of IDR even though the system misses many attacks. The high number of FN however is reflected in the values of recall and accuracy. Remember that test *Protocol, Port and Packet-size Mimic* simulates when the attacker changes the information of the IP packet (protocol, port and packet size) of its attack to mimic a valid connection. When there is an attack, the FA has no way to differentiate the attacks flows from the non-attack ones, as was the case on the single agent

evaluation with a similar test. The FA interprets the flow information as a no-attack. Contrary to the single agent architecture using only flow information, in this architecture we have more information sources. Link information and TCP data are also collected and analyzed by the CSA and the DSA respectively. These agents help to recognize if not all, at least some of the anomalous states when the attacker creates a complex mimic attack. When the system does recognize an attack, even though the FA is reporting a no-attack, the signals for the CSA and DSA activate the DA weights that trigger an alarm-action. When it does not, the signals for the CSA and DSA are not strong enough to activate the alarm-action and the DA triggers a no-alarm-action. We observe that this last situation occurs in the states related to the initial launch of the attack. In these states, the link utilisation and the negative impact in performance are still low, which without accurate flow information make the abnormal states difficult to differentiate from the normal ones. When the attacks start, the congestion and delay value measured by sensor agents are similar to the no-attack states. This causes the DA to trigger an incorrect action generating a FN. As the attack progresses and gets to its peak, the signals from the DSA and CSA make the value of the alarm action better than the no-alarm and the DA triggers the alarm. Once more, this scenario also could be seen as the emulation of a broken or compromised sensor forced to send misleading signals. In this case the FA could be seen as a compromised sensor forced to send misleading information such as labelling an attack flow as normal. Under this assumption we can argue that our architecture also offers certain level of resiliency against broken or compromised elements.

Tests *Multiple UDP sources*, *Multiple FTP sources* and *Multiple UDP attack sources* modify the network topology adding more sources of traffic. These tests are important because they modify some of the features that the learning process uses to detect intrusions such as link information, number of flows, packets transmitted per flow type, etc. As the name implies *Multiple UDP sources* adds multiple UDP sources and *Multiple FTP sources* adds multiple FTP sources, both of them are valid applications. Finally in test *Multiple UDP attack sources* we add multiple UDP attack sources to simulate a DDoS attack. In all the tests we modify the traffic patterns but we do not modify flow information.

For the tests with more valid traffic sources (FTP and UDP) the system adapts well

to the new traffic trends resulted from an extra and more sustained bandwidth utilisation, more congestion and more flows. Both the FTP and UDP tests show some FP and FN. The FP is the effect that sustained bandwidth utilisation and extra congestion creates during very short periods of time. The FN is the effect of the delay between the action-signal generated by the SA and the signal processing and action generation in the DA.

8.3.4 Comparison with Hand-Coded solutions

In order to compare our learning approach with other IDS alternatives, we implement two common hand-coded (i.e., non-adaptive) IDS techniques. We use this emulated IDS approaches instead of direct comparison because of the technical difficulties to embed commercial or open source IDS within NS-2 (or any network simulator) and the lack of common evaluation criteria in the IDS field. The use of real IDSs such as Snort, Cisco IDS or CheckPoint was also not possible because of scalabilty constraints. Comparing our evaluation results with other research work was not possible in many cases, or it could produce only subjective conclusions. Some of the works that addresed similar problems to ours used their own evaluation tests. Details at how to reproduce those tests were not available. Other works used the KDDCup 99 Data set, however there were no details available about the specific attacks tested or the data rate employed.

The first hand-coded approach, which we shall refer as *Hand-Coded 1*, emulates a misuse IDS. For this case the IDS is looking for patterns or signatures that match an attack in the same way that some commercial misuse IDS do in real world networks. To emulate this approach we use flow information and a flow-table. Each row in the flow-table correspond to a signature of what is considered a malicious flow. This signature is analogous to the signatures that misuse IDS such as the open-source Snort [162] or the commercial products by Checkpoint [41] use to identify attacks. Each flow that passes through the IDS is compared with the flow-table for a match. If a match is found, the Hand-Coded 1 IDS triggers an alarm.

Besides the Hand-Coded 1, we developed another more complex IDS implementation defined as the *Hand-Coded 2*. This approach is an emulation of the mechanisms employed in some commercial Intrusion Prevention System (IPS) such as the Cisco Intrusion Prevention Systems Sensor [46]. Some of these devices search for intrusions

through signature and anomaly detection methods. The Hand-Coded 2 approach uses a hybrid model by emulating a combination of misuse and anomaly IDS. For the misuse IDS model we used the same mechanism as in the Hand-Coded 1 approach. The anomaly module uses the following input information to identify intrusions in the form of traffic anomalies:

- **Queue information in CSA:** Size of output queue, byte drop rate in queue, output rate of bytes on queue.
- **TCP information in DSA:** ACK rate, TCP window size, average RTT

The IDS agent of the Hand-Coded 2 approach categorises network states as abnormal when an attack flow is detected or when some thresholds related to the inputs analysed (queue information, TCP information of the monitored connections) are reached. The operation design of the Hand-Coded 2 approach, the input information to use and the threshold values of the variables used are the result of a manual, long and tedious observation of the network under normal and abnormal states. We first analysed as much input information as possible, that included not only the six variables already mentioned relating queue and TCP information, but also included the input rate of queue in bytes and packets, output rate of queue in packets, packet drop rate in queue, number of duplicate TCP ACKs, RTT mean deviation estimate, smoothed RTT estimate and others. We observed all these variables under normal and attack states and we registered how they changed in each state. We then selected the variables with the most noticeable changes between states. For each input feature (monitored variable) we registered an estimate of the threshold reached when the network state changed from normal to attack and vice-versa. Table 8.5 shows a set of tests aimed at tuning the values and the types of variables to be used in order to correctly categorise network states. For each test we either changed the value of one of more variables or we changed the monitored variables. Table 8.5 shows only a subset of all the tests that we performed trying to minimise the number of False Positives or Negatives using the hand-coded approaches.

After we obtained the set of variables to be used and their estimated values under normal and abnormal states, we evaluated the learning and hard-coded approaches using the simulated network created for the tests *Traffic Pattern* and *Protocol, Port and Packet-size Mimic* test. We used the test *Traffic Pattern* because it only changes

Table 8.5: Evaluationg Thresholds in Hand-Coded IDS

Test	FP	FN	Events
Test 1	2	114	199
Test 2	2	114	199
Test 3	1	49	199
Test 4	1	49	199
Test 5	1	49	199
Test 6	1	48	199
Test 7	2	4	199

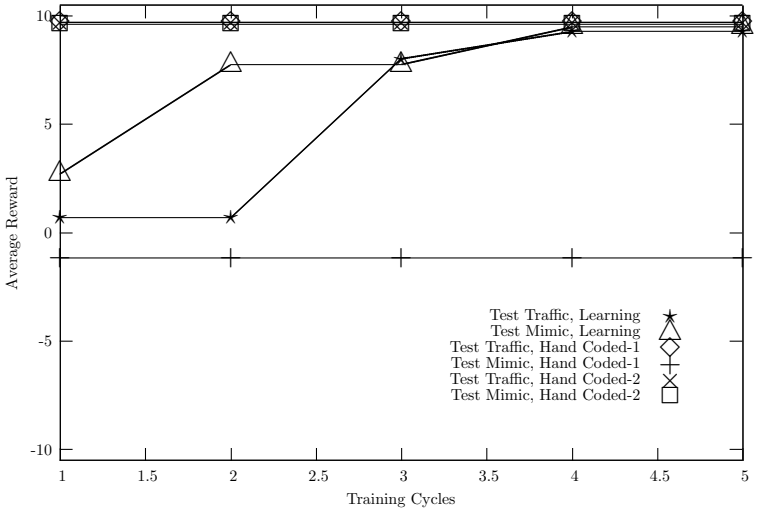
Table 8.6: Tests Results Hand-coded

Test	Precision	Recall	Accuracy	FP Rate	FN Rate
Traffic Pattern Learning	100%	99%	99%	0%	3%
Traffic Pattern Hand-coded 1	98%	100%	99%	0%	2%
Traffic Pattern Hand-coded 2	98%	100%	99%	2%	0%
Protocol, Port and Packet-size Mimic Learning	100%	29%	66%	0%	40%
Protocol, Port and Packet-size Mimic Hand-Coded 1	86%	10%	44%	14%	58%
Protocol, Port and Packet-size Mimic Hand-Coded 2	98%	49%	70%	2%	42%

the traffic pattern of the attack and it ought to be very simple to detect. On the other hand, the attacks in the tests *Protocol, Port and Packet-size Mimic* are the hardest to detect because they emulate some of the signatures of normal traffic.

The performance using the intrusion detection metrics is shown in Table 8.6 and the learning curves of the tests are shown in Figure 8.5. The Hand-Coded 1 approach had no problem to identify attacks and it had low FN for the *Traffic Pattern* test. Nevertheless, it completely failed to detect the mimic attacks. This is the same problem that misuse IDS have when the signature of the attack changes or when they face unknown attacks (Altough the curve does not show it, the result was the same for

Figure 8.5: Learning Curves



the test *Attack Port Changed*). Contrary to the results of the The Hand-Coded 1 approach to the mimic attack, the results for Hand-Coded 2 and our learning approach using MARL are positive. This confirms our argument that for more reliable intrusion detection we need a variety of information sources.

The learning and the Hand-coded 2 approaches were capable of detecting the mimic attacks even though one of the sensors was reporting incorrect information. Both approaches give very good results regarding the identification of normal and abnormal states in the network. Hand-coded 2 reaches maximum performance from the beginning of the simulation. Nevertheless it has a major drawback; it requires in-depth knowledge from the policy programmer about the network traffic and patterns in order to detect intrusions. While the learning algorithm requires some time to learn to recognise normal and abnormal activity, it does not require any previous knowledge about the behaviour of the network or exactly which features to observe. Another advantage of the learning approach is its flexibility to use any (large enough) set of features to achieve some reasonable level of detection. The learning approach automatically will use the interesting features to detect attacks and it will ignore the ones that do not represent different states. Finally there is another quantitative advantage of the learning approach over the hand-coded, that advantage is *cost*. In Chapter 6 we deliberately left out from our

evaluation criteria some quantitative and qualitative metrics, among them cost. We did so because of the difficulty in evaluating those metrics, however we found important to comment about the advantages that our approach could bring to this criterion. As commented, the cost of IDS are in *deployment* and *monitoring* [154]. Related to deployment, our approach could be easier to configure and maintain, as result it would require less time from technical staff, less technical knowledge from them and possibly less cost. In terms of monitoring, both approaches have similar results, so the impact because due to high FP rate or the loss due to FN leading to further damages would be similar.

8.3.5 Findings and Limitations

We have shown how a group of agents can coordinate their actions to reach the common goal of network intrusion detection. Each SA is specialized to process a specific type of data such as link utilisation, link congestion or TCP connections while DA agents learn how to interpret the action-signals sent by the sensor agents without any previously assigned semantics. These action-signals aggregate the partial information received by sensor agents and they are used by the decision agents to reconstruct the global state of the cell. In our case study, we evaluated our learning approach by identifying normal and abnormal states of a realistic network subjected to various DoS attacks. Overall, we can conclude the following:

- We have successfully applied RL in a group of network agents under conditions of partial observability, restricted communication and global rewards in a realistic network simulation.
- The use of a variety of network data has generated good results to identify the state of the network. The system presents high reliability even in cases when some sensor information is missing or compromised.
- We have successfully used a high resolution state space by applying tile coding as a function approximator to a small number of agents.
- The learning approach yields better results than a simple hand-coded alternative. It also yields similar results to a more complex hand-coded alternative using a

variety of sensor information. The main advantage of the learning approach is that it does not require an expert with prior knowledge of the network environment.

- The hand-coded alternatives may yield better results but they require a fine tuning process that is complex, slow and tiresome. Although our simulation is not very long and it has few agents, as a personal experience we found the tuning process very complicated.
- One important limitation of the single cell architecture is its scalability to a large number of agents. Due to the coordination issues found in the previous chapter we have a maximun number of agents per cell.

The next step is to scale up our agent architecture to a larger number of agents using the hierarchical approach from our previous work on abstract networks. This will allow us to create more complex network topologies emulating geographical cells of agents, security domains composed of cells or groups of cells, complex DDoS attacks and eventually the emulation of real packet streams inside the network environment.

8.4 Hierarchical Architecture

To expand our architecture to larger numbers of agents we propos the Hierarchical Model. It allows us to increment the total number of sensor agents to a larger number than in a single cell architecture as we have shown in our experimental results described in the previous chapter. These experimental results show that a hierarchical architecture of six agents and two cells have better performance than a single cell containing the same six agents. Furthermore using the abstract environment we were able to build a large simulation composed of 3 hierarchical levels, 9 cells, 27 SA and 13 DA. To evaluate this same large architecture of agents but now under more realistic conditions we have developed a series of tests using the network simulator. These tests aim to evaluate the performance and scalability of the agent architecture, its ability to recognize new attacks and the use of a function approximation technique.

Table 8.7: Tests Results Hierarchical Architecture

Test	Precision	Recall	Accuracy	FP Rate	FN Rate
Traffic Pattern	98%	96%	95%	3%	0%
Attack Port Changed	95%	93%	91%	5%	13%
Protocol and Port Mimic	95%	92%	91%	4%	15%
Protocol, Port and Packet-size Mimic	96%	65%	75%	4%	42%

8.4.1 Basic Tests

We follow the same strategy as our previous evaluation. First we train our agents using some normal and abnormal traffic patterns. Next, we evaluate the performance and learning of the architecture using a set of tests changing traffic patterns and flow information. The basic test is "Traffic Pattern". On it we change the normal and abnormal traffic patterns and we measure the performance of the architecture based on the intrusion detection metrics. The results of this test are shown in Table 8.7. The results are very similar to the same test using a single cell. We can see a low level of FP and FN which in turn generates high levels in precision, recall and accuracy.

8.4.2 Adaptability Tests

To verify how the hierarchical architecture will work under more complex attacks we evaluate it using a set of mimic attacks. These attack were very similar to the ones performed to the single cell architecture and they change traffic and flow information. Test *Attack Port Changed* changes the port used by the attacker to any given port, we used again port 6665 as the new attack port. Test *Protocol and Port Mimic* changes the attack port to be the same as other valid applications and in the test *Protocol, Port and Packet-size Mimic* we also change the packet size. The performance of the intrusion detection metrics for these tests are presented in Table 8.7. Although they have some FP and FN the system is still able to have good results for the tests *Attack Port Changed* and *Protocol and Port Mimic*. For the hardest test to perform the system still recognizes attacks but it misses 42% of them. We are not surprised by the 42% of

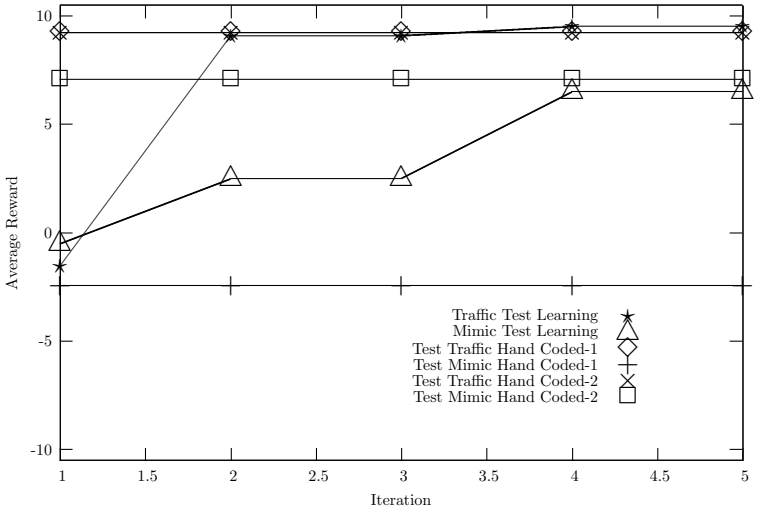
misses, on the contrary the 65% of recall and 75% of accuracy show that the systems is capable to cope with incorrect, misinterpreted or false information from one of the sensors.

8.4.3 Comparison with Hand-Coded solutions

For the evaluation of our proposal versus other IDS implementations we used again the hand-coded approaches developed for evaluating the single cell architecture. The operational model of the hand-coded IDS is the same that the model described in the previous section but enhanced to work as a DIDS architecture. In this way the Hand-coded 1 approach uses only signatures to detect intrusions while the Hand-code 2 is a hybrid model using misuse and anomaly intrusion detection techniques. Regarding to the hybrid intrusion detection method used by the Hand-coded 2 approach, we used the same operational model and the same input features that we defined for the Single Cell tests. However, we needed to use different thresholds values. To act as a DIDS each sensor on the Hand-coded 1 and Hand-coded 2 approaches generates an alarm or log each time they detect an alarm. A central DIDS agent receives or collects the information generated by the sensor agents and when a threshold on the number and type of alarms is reached it generates an alarm reporting a global attack in a similar manner to our approach.

We evaluate the three approaches (learning, hand-coded misuse and hand-coded hybrid) using the tests *Traffic Change* and *Protocol, Port and Packet-size Mimic* for the same reasons formerly stated. In a similar way to the evaluation with the single cell, the learning approach needed some time to learn how to identify attacks as it can be observed in the Figure 8.6. For the test *Protocol, Port and Packet-size Mimic* the system achieves a good level of performance, nevertheless the results in the intrusion detection metrics are never as good as the hybrid hand-coded approach as presented in Table 8.8. As observed, the hand-coded 1 (misuse only) presents very good performance for the *Traffic Change*. Nevertheless for the *Protocol, Port and Packet-size Mimic* test it completely fails to detect the abnormal network states. Contrary, the hand-coded 2 (hybrid) approach performs well in both tests as it did in the single cell architecture's tests. As it can be observed on Table 8.8 and in Figure 8.6 it performs even better than the learning approach.

Figure 8.6: Learning Curves Hierarchical



Even though the learning approaches take more time to reach their optimal states and they identify and categorise less abnormal network states than a hybrid IDS approach, it has the advantage that network operators do not need to know in advance which features are needed to be monitored to detect those abnormal states. We think that this advantage is more evident than in the single-cell architecture due to the complexities that a large network can have. As a practical experience, it was very difficult for us to adjust the parameters used by the hand-coded approach to improve its results. This task was even more difficult than in the tests involving Single Cell architectures. The process was more complex due to the large number of agents and data to analyse. Again, in order to improve results we needed to evaluate several times the same test changing the features and threshold required to identify the anomalies in the network. Using the same threshold values as used for the Single Cell architecture resulted in very high levels of FP and FN as shown in the last row of Table 8.8.

8.4.4 Online Learning

So far, the evaluation of the system's architecture has been done in two steps, learning and exploiting. The first step (learning) was to train our agents with a predefined set of examples. After they have learned to recognize and categorize abnormal network states

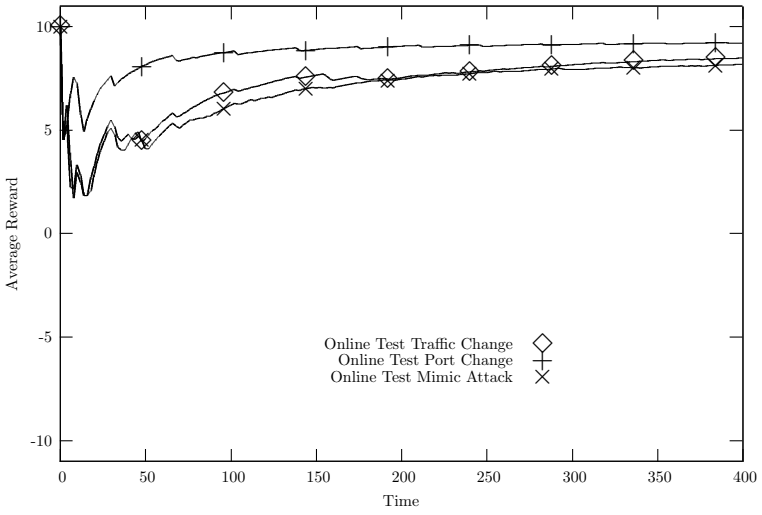
Table 8.8: Tests Results Hierarchical Hand-Coded

Test	Precision	Recall	Accuracy	FP Rate	FN Rate
Traffic Pattern Learning	98%	96%	95%	3%	0%
Traffic Pattern Hand-coded 1	100%	98%	98%	0%	6%
Traffic Pattern Hand-coded 2	100%	98%	98%	0%	6%
Protocol, Port and Packet-size Mimic Learning	96%	65%	75%	4%	42%
Protocol, Port and Packet-size Mimic Hand-Coded 1	86%	10%	38%	14%	66%
Protocol, Port and Packet-size Mimic Hand-Coded 2	98%	79%	85%	2%	30%
Protocol, Port and Packet-size Mimic Hand-Coded 2 with thresholds values from Single Cell Tests	86%	10%	44%	14%	59%

we stop the learning. The second step (exploiting) evaluates how well the agents can perform using a different set of traffic patterns and network information. In this second step the agents are not learning anymore. One open question that we could not answer using with the abstract environment was to investigate how the proposed architecture deals with an online learning approach where agents never stop learning.

To investigate how the system can improve results using an online learning approach we performed a series of tests aimed to evaluate our proposal capabilities under continuous learning. Contrary to previous evaluation tests performed in the network environment, on this online learning approach agents never stop learning. First, in the learning phase we trained the agents as normal applying the baseline traffic patterns and network information. Next, without stopping the learning process we changed the attack patterns to verify how long it would take to the architecture to stabilize positive results in its decisions about the state of the network. At this point, we also re-encourage our agents to start exploring again. We repeated this evaluation with the attack patterns used in the tests *Traffic Change*, *Port Change* and *Protocol, Port and Packet-size Mimic*. The resulting learning curves are shown in Fig. 8.7. The more similar the evaluated test related to the baseline training set, the less spikes the graph

Figure 8.7: Online Learning



shows and higher values of average reward will be reached faster.

Using the previous learning patterns and online learning, the agents are able to learn new types of attacks with very few iterations. These tests show that the system can either recognize new attacks with previously learned patterns or within an online learning process. This opens new research avenues to evaluate in more depth how to combine both approaches to improve the system capabilities. Important paths to follow are the exploration/exploitation strategy to use and the security risks involved on re-evaluation the value function of the agent's actions. The exploration/exploitaitation strategy is related to action selection by agents. Generally at the beginning of learning agents try to explore actions whilst in the end they tend to exploit, that is to execute actions with the best expected future rewards. One question that raises is: After the agent has learnt and in order to discover the actions with the best future rewards and to avoid to be stale executing sub-optimal actions, when and for how long we need to influence agents to explore instead to exploit? In our evaluation we did know when the attack started and we could influence the action selection by increasing the exploration (this is also denoted by the peaks in the learning curve). Nevertheless in a practical implementation when an attack is not known in advance and we will need to provide an answer to this issue in future research.

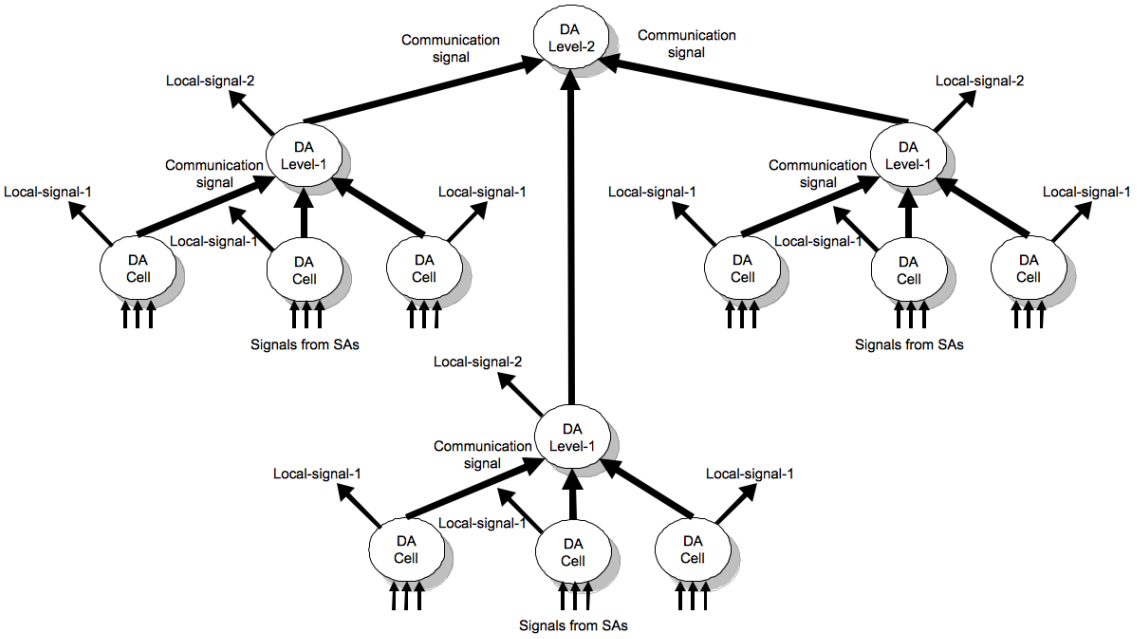
The use of an online learning approach brings up another important aspect to solve related to the security of the system. Anomaly IDS and in general, IDS using machine learning techniques has been questioned [57, 20] about their resiliency to be maliciously manipulated by the attacker. In their work, Barreno *et al.* discuss some techniques that can be used to deceive security systems based on machine learning. They also propose some research lines to address this problem. One Interesting and related research path is what they called *learn-adapt-relearn* where the learner learns, adapts and re-learns based on game-theory concepts. Another possible approach is to use control systems which are very related in how RL agents use the feedback from the environment to learn.

8.4.5 Local Information

The last chapter opened some questions that we tried to answer using the network environment. One of these questions was about the ability of the architecture to signal local states of cells. All the evaluation test showed until now have been generated by an agent architecture that only produces summarized information about the whole state of the network. This architecture does not provide information about local states and attacks, using the hierarchical architecture the top level DA triggers an action indicating the state of the whole network. This approach works very well to summarize network information; however in practice it is also useful to receive more information on the locations of the attack. To address this problem we need that the DA inside cells are able to report some network state. Recall that the cell's DA already sends action-signals up the hierarchy. Nevertheless the individual signal sent is meaningless on its own. Our solution to include local state information was to create a second set of signals indicating the explicit attack status of the cell, for example the local DA is explicitly signaling whether it believes the local cell to be under attack.

Using the local information method, the DA inside a cell generate two signals actions (*DA cell* in Figure 8.8). The first signal (communication signal) is the already explained in previous tests. The semantic value of this signal is unknown except for the DA that receives it, that is the DA in the next hierarchical level (DA Level-1). The second signal (Local-signal-1) is sent by the *DA cell* to the network operator. The value of this signal is the local state of the cell, this is *normal* or *abnormal*. All the DA agents inside a

Figure 8.8: Agent Architecture



cell generates these two actions. DA agents (DA Level-1) that collect information from cells also follow this mechanism. They generate two signals, one only meaningful for the DA in top of the hierarchy (DA Level-2) and a second signal (Local-signal-2) to the network operator. This second signal indicates the state of the group of cells. The state is considered normal if all the cells are normal, or abnormal if only one cells presents abnormal activity.

The test results using this approach showed acceptable intrusion detection metric values even for complex tests as the *Protocol, Port and Packet-size Mimic* (See Table 8.9). As observed the values of all the intrusion detection metrics for the global and the local signals fall in similar ranges. It is interesting to see how the global signalling seems to have an overall better performance than the local one. We assume that this behaviour is a reflex of the architecture's resiliency to ignore incorrect information from sensors. In these cases there are cells (no single sensor agents) triggering the incorrect signal according to the learnt state, nevertheless as the signal flows up to the hierarchy the effect of the miscategorised activity is reduced.

Table 8.9: Tests Results Local Signaling

Test	Precision	Recall	Accuracy	FP Rate	FN Rate
Traffic Pattern Global	98%	96%	95%	3%	0%
Traffic Pattern Local	97%	99%	98%	3%	2%
Port Change Global	95%	93%	92%	5%	13%
Port Change Local	96%	98%	96%	4%	4%
Protocol, Port and Packet-size Mimic Global	96%	65%	75%	4%	42%
Protocol, Port and Packet-size Mimic Local	95%	57%	71%	5%	43%

8.4.6 Findings and Limitations

One important limitation of the single cell architecture was its inability to be expanded to a large number of agents. To overcome this problem we used a hierarchical architecture of agents. Using this architecture we were able to extend the number of sensor agents with good performance compared to similar DIDS architectures based on hand-coded techniques. This hierarchical approach was also able to work under an online learning approach where agents never stop learning. However, the evaluation of the network state using this approach hid the local state of each cell. To overcome the problem we implemented a double signal approach. Employing this technique, sensor agents send hidden signals to decision agents along with significant local signals to network operators.

Regardless the positive results obtained, using the hierarchical approach still have some limitations. The first limitation is their comparative results with a hand-coded approach performing misuse and anomaly intrusion detection. For all the tests performed the hybrid hand-coded approach outperformed our learning architecture. Although our approach presents other advantages compared to hand-coded solutions, the performance issue is a factor to review in future research. Rather than limitations, there are still some open questions not yet answered. The most important question is how the hierarchical approach would perform under a larger input space represented with more resolution in the current features or simply with more features. Another related open

question is how the single cell and hierarchical approach will work using real data on a real network, perhaps analyzing DoS and DDoS to web services infrastructures. Finally our approaches have been trained using predefined data sets. To fully exploit the RL capabilities it would be important in the future to extract a reward function related to some sort of feedback from the environment. This feedback could be computed by collecting a diverse set of information about delay and latency times of specific applications. We think that this process can be achieved by collection that information through specialised probes, from sensor agents or from the monitored application layer. We will review these methods in detail in the conclusions of this thesis.

8.5 Conclusions

This chapter presented our findings in the evaluation under realistic conditions of an intrusion detection system based on single and multi-agent architectures. In the previous chapter we left some unanswered questions that we could not answer using the abstract environment. These questions were related to the portability of our proposed agents' architecture to a more realistic scenario, to the coordination, adaptation and online learning capabilities of our approach as well as to the scalability of the architecture to a large number of agents and state-action space. To answer these questions we ran a set of tests using a single agent architecture, a multi-agent system architecture represented by a single cell of agents and a hierarchical architecture of agents represented by a system of cells. The tests have provided us with some general results as well as with some specific findings related to the agent architecture used.

Important general results provided by evaluation are the evidence of the adaptability of our architectures to recognise new and unseen suspicious activity as well as the utility of a function approximation technique to represent the state space with high resolution and dimensionality. Regardless of the type of architecture used, we first trained the agents with an initial set of traffic patterns and network information. Then, after the system has learnt the initial set of information we modified it to different degrees. For simple scenarios with small modifications the architectures performed well and with high values in the intrusion detection metrics. As the attacks became more complex the performance was lower. Nevertheless, even for complex tests the architectures showed

acceptable levels of adaptability. In relation to the tile coding as function approximation technique, for most of the tests performed we have used at least three features as input. If we had used Q-tables to represent the environment state, we had ended up with a huge state-action space that probably would have required thousands or even millions of iterations to learn, while with tile coding we only required a few hundred iterations.

In addition to the general results obtained, we collected some interesting information regarding each particular type of architecture. The first set of tests in the network simulator presented results of a single RL agent using tile coding and flow information as input. An important finding yielded by the use of this architecture was the feasibility on the use of RL in a simple real world application and the weakness protection that a single source of information provides to complex attacks. The next set of experiments were done using the single cell architecture and they provided evidence in the following aspects: We found the first indicators that our proposed MARL architecture plus a function approximation generated acceptable performance in a realistic network environment and that multiple sources of information provide better adaptability and resiliency characteristics than a single source. The experiments performed with the hierarchical approach reinforced our previous observations regarding to the open questions about the portability of our proposed architecture to a more realistic environment and its adaptability to recognise unseen normal and abnormal states and correctly categorised them.

Other unanswered questions from Chapter 7 were related to the online learning ability of our MARL approach and the problem on how to extract local information of the cells composing the hierarchical structure of agents. To review how the hierarchical agent architecture worked using an online learning approach we modified our initial tests so the agents never stop learning. The results on this area were promising and we could verify that the system keeps maintaining a good performance related to the intrusion detection metrics. Nevertheless, the capability to re-learn new traffic patterns raises new concerns in how the system can protect itself against malicious activity aimed to use this ability as an attack vector. With respect to the local state of cells, recall that the basic operation of hierarchical architecture is to trigger a signal that summarizes the state of the whole network. Even though it is not an indispensable capability, we believe that adding the ability to report local states of cells would increase the practicability

of our approach. Furthermore, in the future it would allow tracking the origin of an attack. In order to do so, we added a second signal in each cell that only has local meaning. Using this signal a human can easily spot which are the cells receiving or generating an attack. The results of the test using this approach presented acceptable levels of performance in both the global and the local signals.

To standardize the evaluation of our proposal with other intrusion detection methods we emulated two common approaches. The first one was a misuse IDS represented by hand-coded approach using flow information as signatures. The second approach was a hybrid hand-coded intrusion detection using misuse and anomaly methods. We evaluated the performance of our proposal with these hand-coded solutions and we found that our learning approach offers positive results under the intrusion detection evaluation criteria. Specifically the learning approach yields better results than the simple misuse IDS alternative. It also yields similar results to hybrid alternative using a variety of sensor information. Although the evaluation suggested that we need more investigation in how to increase the performance of our learning approach, we think that the main advantage of the learning approach is that it does not need a designer with prior and in-deep knowledge about the network environment.

In summary, we have applied and evaluated our MARL Signaling approach in the intrusion detection domain under conditions of partial observability, restricted communication and global rewards in a realistic network simulation. Using a single agent, a multi-agent architecture confined to single cell and a hierarchical architectures of sensor and decision agents we obtained positive performance results on the categorisation and detection of network states normal and abnormal network states generated by Flood-Based DoS an DDoS. Finally we detected that future research related to our approach includes the improvement of the performance or our approach compared to hand-coded methods and the investigation of negative outcomes consequence of the online learning approach.

Chapter 9

Conclusions

Throughout this thesis we have described and proposed a feasible alternative to detect and categorise Flood-Based Denial of Service and Flood-Base Distributed Denial of Service attacks by means of a hierarchical architecture of sensor and decision agents. Through the use of Reinforcement Leaning, a signalling method and special heuristics, the agents learn to categorise normal and abnormal network activity. Under the evaluation criteria defined in Chapter 6 we have shown that the agent architecture performs wellon both, the abstract and the network simulation environment. In this chapter we state our final conclusions about these results and the future paths that our research can follow.

The chapter is divided into four main sections. Section 9.1 outlines the problems generated by FBDoS and FBDDoS attacks, our motivation to conduct this research, and the mechanism that we employed. Section 9.2 presents a summary of the experiments and the results obtained in both the abstract and network simulations. The third section discusses what we think are the relevant aspects of this research and how we think it offers a novel approach to manage some of the problems generated by DoS and DDoS attacks. The section also establishes the limitations of this study. Section 9.4 is the last of the chapter. It explores how our research could evolve by improving its detection mechanisms or by adding to it new capabilities, such as defensive actions. This section also describes how the proposed agent architecture could be used in related domains.

9.1 Overview

Denial of Service and Distributed Denial of Service Attacks are important attacks on today's Internet infrastructure. According to some reports this activity is not just common but also the aggregated bandwidth generated today by FBDoS and FBDDoS is around 40 Gbps and predicted to reach 100 Gbps by 2009 [115]. These attacks are characterised by starvation of a specific resource of the target. The resource may be CPU, memory, or available bandwidth. A major problem in detecting the type of attacks is that their abnormal activity may be completely indistinguishable from normal utilisation of the resource. This is especially true for simple detection mechanisms such as basic misuse and anomaly Intrusion Detection Systems, packet analysers and Network Management Systems.

To address this problem some researchers have used [125, 117, 19, 194, 98] and advocated [22, 112] the use of multiple types of source information to increase the accuracy of current intrusion detection engines. The rationale behind this principle is that complex attacks must be analysed using a high dimensionality of information that is not possible with a single source of data. This, in turn, brings out the problem of how to collect, merge and analyse this rich mix of information under practical constraints in processing power and communications channels. In Chapter 2 we discussed some of the approaches that have been taken to solve this problem. These include the use of analysis tools in the fields of statistical methods and machine learning; the deployment of central and distribute processing; and the integration of heterogeneous source data generated by SNMP devices, Netflow collectors and IDS engines among others.

To provide an alternative solution to this problem we have proposed a distributed architecture of learning and autonomous sensor agents. By means of a Reinforcement Learning algorithm, sensor agents collect a partially observable network state and generate communication action-signals to higher-level agents decision agents in a hierarchy. Without any previous knowledge about the semantics of the signals, decision agents learn how to interpret them to reconstruct a more complete network state. Using this new state information generated by partial observations and communications signals from sensor agents, decision agents are capable of triggering a signal indicating if the network is under attack or not. With the aim of expanding the solution to a large number of agents we used a hierarchical approach. The hierarchical architecture is composed

of groups of agents called cells. In each cell a decision agent is in charge of summarising the cell state and signaling this information to higher levels in the hierarchy. This iterative process can expand several hierarchical levels where the last agent summarises the state of the whole network. The outcome is that under conditions of partial observability, with low bandwidth utilisation and low central processing, we have the ability to extract and reconstruct normal and abnormal network states from a rich variety of source information required for reliable intrusion detection.

9.2 Summary of Experimentation

To evaluate the hypothesis defined in Chapter 1 we performed a variety of tests. The tests and their results have been described in Chapter 7 and Chapter 8. In Chapter 7 we used the abstract simulation environment. The abstract environment was the initial test bed used to evaluate our ideas in how to develop an intrusion detection engine based on RL. We have referred to this simulation framework as *abstract environment* due to the nature of the simulated objects. Even though we resorted to an idealised model of a network, the abstract environment and the evaluation tests performed on it pose the principal learning and coordination challenges of the real-world case. The network simulator and its tests were described in Chapter 8. We used this simulation environment to emulate more real and complex attacks. This environment provides an accurate model of end nodes, routers, data links, queuing methods, the TCP/IP stack, packet loss and delay. Although the best way to evaluate IDS is to use real networks with real traffic [8, 142], in Chapter 6 we concluded that the network simulator was a feasible alternative. As stated before, there is no standardised methodology to evaluate IDS. We defined a more rigourous evaluation methodology, intrusion detection metrics (See Table 6.1 and 6.2) and evaluation criteria controlling the testing of the different scenarios defined.

The hypothesis that governed the evaluation tests was:

> *A hierarchical architecture of Distributed Intrusion Detection Systems using RL-IDS agents is capable of detecting and categorising Flood-Based Denial of Service and Flood-Based Distributed Denial of Service Attacks at inter and intra domain scopes. This architecture is capable of detecting and*

categorising these attacks with high values of intrusion detection metrics similar to hand-coded approaches, but with the ability to adapt to new attack patterns.

The initial tests were focused on the difficulties in coordinating MARL agents, specifically on how the degree of complexity increases as the number of agents and states increase. After some evaluation we discovered that agents were not coordinating their actions. To correct this undesirable behaviour we applied a series of strategies. The first strategy to coordinate actions inside cells with three to six agents or with agents having a state space greater than 2 states was to include at least 25% percent of attacks. The second strategy was to change the exploration/exploitation strategy from a common *Boltzmann* to a combination of Boltzmann and a total exploitive strategy. Under this new combined strategy, agents initially explore and exploit actions during a period of time defined by the *exploitive level* (See eq. 7.2). After the period of time ends, agents start to only exploit actions. We discovered that this method was very useful to achieve coordination in cells with more than 4 agents or when regarding the number of agents, each had more than 2 states.

Using the applied mechanisms and the values of the RL parameters that we learned in the abstract environment, we then moved to the network simulation. We used the network environment to add more realistic characteristics to our evaluation and to answer the questions left unanswered in the abstract environment. Specific matters relating to these questions are the portability of our proposed agent architecture to a more realistic scenario; the coordination, adaptation and online learning capabilities of our approach as well as the scalability of the architecture to a large number of agents and state-action space. The network environment was based on the NS-2 network simulator and the tile-coding software. We used the NS-2 software as the network simulator for several reasons including: it provides an accurate model of different network object such as routers, end nodes, links and protocols; it is open source, therefore free to be used and modified accordingly to our aims as for instance, adding the RL capabilities to the agent's behaviour.

We used a simpler model of our agent architecture in the first test of the network simulator. To detect normal and abnormal network states we used one RL agent applying tile coding as function approximation technique and flow data as a single source

of information. After performing a variety of tests with different network scenarios and topologies, we established the validity of the use of RL as a feasible approach for use in a simple real world application and the weak protection that single source information provides to complex attacks. The next set of tests used the multi-agent architecture proposed. We expanded the architecture from simple scenarios with one hierarchical level and one cell to complex architectures by adding three hierarchical levels and several agents. For the majority of tests, agents were able to detect many of the attacks. Although the performance was lower for scenarios emulating complex mimic attacks, we demonstrated that the use of multi-source information improves the metrics compared with approaches using only one source. In addition to what has been said, the use of multi-source data increases the reliability of the agent architecture with sensors that fail or are compromised by the attacker.

As previously stated in Chapter 6, the evaluation or benchmarking of different intrusion detection systems is a complex process. To simplify it a little and to provide a base line against which our approach could be compared, we developed two hand-coded IDS approaches. The first approach emulated a misuse IDS looking for attack signatures based on flow information, a second approach was based on misuse and anomaly detection. The misuse-only approach worked well for simple tests, but failed completely for the mimic attacks. On the other hand, the hybrid approach performed well in all the tests including the mimic attacks. These results reinforce our argument that a variety of sources of information increases the detection capabilities. Relating to performance, the hybrid approach performed slightly better than our learning solution. Even though the results show certain advantages in performance of the hybrid hand-coded solution over the learning approach, we consider that the main disadvantage of the hybrid approach is that it does require a designer with prior and in-deep knowledge of network protocols, variables to watch and the particular network environment analysed.

In all the experiments outlined, the agents first are trained and then the learning process is stopped. Finally agents are exposed to the new traffic and network information to detect intrusions. This method is classified as off-line learning and it is used by many machine learning techniques. However, one advantage of RL over some other machine learning methods is its online learning ability. We obtained interesting results where agents were able to re-adapt their learning quickly to the new traffic and network

information. Even though the agent architecture showed the ability to detect complex attacks and to adapt to recognise new ones, the investigation opened more questions regarding the best exploration and exploitation strategies to follow and about the security of an online learning process. We will discuss this topic in more detail further in these conclusions. The final set of tests dealt with the capability of the multi-hierarchical architecture to generate global and local signals. We found that the performance results for both the local and global signals were very close. We also pointed out that this capability is not mandatory, nevertheless it adds a practical capability to analyse local information and in further research work using defensive actions it may be useful to track the source of an attack.

9.3 Research Relevance

Following the evaluation of our proposed architectures under various scenarios we can draw a number of conclusions related to the relevance and limitations of this study. We will examine these in the next section.

9.3.1 Contribution of the Study

The research presented has proposed a novel approach using an adaptable and feasible mechanism to accurately detect and categorise FBDoS and FBDDoS attacks. The solution employs a hierarchical architecture of Distributed Intrusion Detection Systems and RL agents. In this thesis we show the application and evaluation of a MARL architecture applied to the intrusion detection domain in a realistic network simulation. The intrusion detection domain poses important challenges due its conditions of partial observability, restricted communication, global rewards and distributed control. We identify the contributions of this study in two main areas: Adaptability of Intrusion Detection Systems and Multi-Agent Reinforcement Learning.

Related to the area of Adaptability of Intrusion Detection Systems, we summarise our contributions below:

- **Adaptability to recognise new attacks with minimal modelling requirements:** Research papers about the detection of DDoS are numerous, however the majority of them use statistical [42, 103] or hard coded methods to identify the

attacks [87, 99]. These methods often require complex models and in-depth knowledge about the network environment. We are proposing an adaptable method by means of machine learning that requires little modelling or previous knowledge about the network behaviour.

- **Semi-descentralised DIDS:** In the initial chapter of this thesis we identified the use of heterogeneous agents to detect distributed denial of service without central processing or management as an area with little extant research [87, 109, 192] compared with the research done using DIDS based on central processing. The key advantages of these mechanisms over the central processing approach are the better resource utilisation (i.e. CPU, bandwidth) and resiliency (i.e. if the central processor fails). In this thesis we have presented our approach related to a distributed architecture of autonomous agents coordinating their actions with limited communication and central processing. Although our approach makes some use of a central processing facility, the requirements are minimal because the main processing is done autonomously by each sensor agent. This approach saves bandwidth as a network resource and requires little processing in the decision agent. Our system also has some level of resiliency against broken or compromised sensors as we showed in the *Protocol, Port and Packet-size Mimic* attacks tests of Chapter 8. In those experiments one sensor agent sent wrong signals as result of complex activity hiding the attack. The system was capable of recognising if not all, then at least some of the attacks.

- **Use of multiple sources of information:** In the same way that there is little research work using adaptable platforms to identify FBDoS and FBDDoS attacks, the majority of the work has used a single source of information such as SNMP [33], DNS [186] or Routing information [191, 39, 75]. In this thesis we are presenting an approach that efficiently uses multiple sources of network data to identify and categorise anomalous behaviour with high levels of performance. We have shown that solutions using a variety of input information outperform approaches using single sources of data.

- **Automatic feature selection:** Hand-coded solutions present one important disadvantage compared to the learning approach proposed. In hand-coded ap-

proaches, the designer must have in-depth knowledge of the network environment to know the important features to be analysed and which are the values that indicate intrusions. On the other hand, in the learning approach, the RL process automatically selects the important features and ignores the irrelevant information. In this research we only used nine features and although manual analysis was feasible, it was hard and tiresome. In further research we could add features not available in the NS-2 simulator or related to specific protocols (i.e. URLs in HTTP, SNMP MIB-2, etc.). Then it would be possible to determine if adding more features improves detection abilities. If so, RL is better than the manual analysis considering than the amount of data to process may become unmanageable for a human operator.

- **Selected feature set:** As stated in Chapter 8, RL and tile coding transparently use or ignore certain features. For this reason it is difficult to identify which features are the most relevant without performing some sort of statistical analysis. Nevertheless, through manual testing we found what we consider are the most relevant features. The set includes the average number of TCP acknowledgement packets (ACK) received, the average TCP window size, the average RTT of packets, the link utilisation in bytes per second, the size of the link queue in packets and the number of packets dropped by the link queue.

The research make the following contributions in machine learning:

- **MARL coordination by means of learning to signal:** We think that a major contribution in the area of RL is the methodology of training high level agents through signals from low level entities. To the best of our knowledge, there is very little research in this area. We consider that this research extends past research in order to handle larger numbers of agents and more complex state-action space. We also think that we can adapt the proposed mechanism to other domains. Some scenarios related to computer networks where this methodology could be adapted are the summarization of monitoring information in Network Management Systems (NMS), the fine tuning of Quality of Service (QoS) parameters in large Voice over IP networks, resource allocation in cloud computing, or distributed sensor management.

- **Practical use of MARL:** Panait *et al* [132] in their MAS Survey propose some challenging real-world environments to test MARL. Among these environments are Hierarchical Multi-Agent Systems problems, Network Management, and Routing. The use of MARL in real world problems forces us to address important problems related to the scalability of these systems to large numbers of agents and the use of function approximation techniques to scale to high dimensional state-action spaces. In this thesis we have presented the application of MARL to a real world problem in the area of network security. The results presented have shown that applying specific mechanisms, a group of RL agents can coordinate their actions to detect and categorise network activity.

During the evaluation of this research we focused on *Efficiency* as our principal metric and we left out other quantitative and qualitative metrics such as *Cost*, *Easy of Use*, *Interoperability* and *Collaboration* among others. We did so because these are difficult to measure and consequently, it is hard to use them as a strong argument relating to the relevance of this research. Nevertheless, we feel important to point out some other qualitative and quantitative advantages that our system has besides Efficiency.

Concerning *Easy of Use*, our approach is easier to use and to implement than a hard-coded method if we consider all the expertise that a network operator needs to have in order to set up the appropriate parameters. Additionally, for large deployments the manual analysis and deployment of a hand-coded approach results in high costs, mainly due to the use of experienced human resources.

The system proposed is still in the development stage and it has not been ported to network devices (i.e. routers or linux hosts). Nevertheless, in the design, we envisioned it to interoperate and work collaboratively with other device such as IDS, firewalls and hosts. The metric *Interoperability* is related on how the IDS agents work with other agents as the architecture grows, clearly the system is capable off holding several RL-IDS agents working together to improve their IDS capabilities. *Collaboration* is the ability of the IDS to work along with other devices to improve the overall security of the system. In practice, RL-IDS agents work in parallel with a variety of devices such as routers, DNS servers, NMSs, firewalls, etc. This is done to collect information aiming to improve the security of the whole network.

Even though we cannot make strong claims yet, we also think that our research has the potential to evolve to a more sophisticated work in the areas below:

- **Expected future rewards:** One important advantage of Reinforcement Learning over other machine learning techniques is its ability to perform actions not just in relation to the current state, but according to the expected total discontinued reward (or any other principle defined). This allows RL algorithms to foresee beyond the current state. In the experiments described in this thesis we did not use this ability of RL, however we think that it can be used to expand this work to execute defensive actions and to counter sophisticated intrusions using slow and subtle attacks [156].
- **No labels in training data:** We performed our training using labelled examples of normal and abnormal activity. In real networks these training examples are difficult to create or select. This imposes a practical constraint on our approach and in general to any supervised learning method. As opposed to many supervised learning methods, it is possible for us to overcome this problem by incorporating a feedback signal from the environment, for example a network manager analysing if the output of the system was accurate.
- **Corrective and defensive actions:** In the work performed so far we have not considered the incorporation of corrective or defensive actions. However, we believe that the framework can be extended to address such issues. These new sets of actions can be used in the current IDS domain or they can be translated to other domains, i.e. Network Management and Quality of Service enforcement.

9.3.2 Limitations of the Study

As any other research work, this work has some limitations as result of our design decisions and environmental constraints. Next, we present those limitations:

- **Convergence is not guaranteed**. One restriction of our learning mechanisms is that we cannot guarantee convergence to the optimal value function.
- **Online learning risks**. Some researchers have questioned the resiliency of intrusion detection systems based on machine leaning [57, 20] from malicious manip-

ulated by the attacker. This problem is exemplified by Chang and Mok [43] by attacking IDS that automatically generate signatures by means of online malicious activity. This security issue is not only important for our research, but also it is an interesting problem in the trade-off that IDS based on machine learning has to deal with. On one hand, zero-day exploits and the spreading of computer worms require intelligent mechanisms capable of learning and adapting to new attacks patterns. On the other, these adaptable systems may be susceptible to malicious manipulation to bypass their security. In order to take our proposal to a practical deployment special care must be taken on this area.

- **Current dependency on training data:** One important characteristic of RL is that it does not really need previously collected training data. This characteristic makes RL suitable for environments such as the IDS domain where is difficult to create a model and to gather labelled training examples. In this research we are not exploiting this characteristic yet. The only action that our agents perform is to translate states into signals and, in the case of the agent on top of the hierarchy, to alarm in case of abnormal activity. The behaviour of our agents alone does not change the environment and does not offer a reward in the short term. Although we could have adapted our algorithm to include long term and delayed rewards as result to the corrective action performed by a human, the action does not depend on the agents. The resulting environment would be a more complex POMDP. Instead, we have decided to include a reward by means of labelling examples to evaluate our work. Nevertheless, we plan to eliminate the labelling of examples by either extending our work to the domain of long delayed rewards and POMDP, or to include corrective actions from the agents to modify the environment.

9.4 Further Work

It has been explored how our research could evolve to improve its detection performance or to add capabilities to perform defensive actions. Next, we analyse those mechanisms in detail and we include some others that we have not mentioned before.

Recent trends in the research of intrusion detection take a more active role not only to detect intrusions, but also preventing and stopping them by taking defensive

actions. Following this direction, one further step in our research is to adapt our agent architecture to detect and respond to abnormal states of the network. The response mechanisms might be in a variety of forms. Initially the rate-limiting and filtering of offensive flows seems to be a good start. These new actions would modify the state of the network as a result of the action performed by the agents. The new actions would bring out the opportunity to use a feedback signal from the environment. One example of this signal is a computed reward generated by collecting information about delays and latency of specific applications. There are different possibilities to compute a reward signal in this way:

1. **From sensors:** Besides their current roles, a sensor could gather and compute information to be used as rewards. This is advantageous because sensors are already in place and the overhead for collecting and computing a reward could be minimal. Also, a central facility to process and distribute rewards is not necessary. One disadvantage that we foresee is that in many cases, because of its location in the network, sensors might not be able to get the required information to process the reward signal.

2. **From probes:** Probes are specialized sensors that collect specific information about the network. The advantage of this method is that they can be located and configured to collect an optimal reward signal. We said optimal because contrary to sensors, probes can be placed where the reward signal is more representative according to the changes in the network state. An obvious disadvantage is the extra infrastructure required and a centralized or distributed facility to process the reward signal.

3. **From the application layer:** Some applications measure statistics related to the connection status between entities. These applications can provide feedback about the communications channel. Example of these applications are the Transport Protocol for Real-Time Applications (RTP) [150] and the Session Initiation Protocol (SIP) [111]. Both protocols designed to transmit real-time information are equipped with a set of metrics to keep track of the communication channel reliability. These metrics include jitter, packet loss, round-trip-time, etc. The advantage of this method is that we do not require new infrastructure and that

the reward information is measured where the DoS attack would affect more, that is the end user application. A clear disadvantage is that this solution is restricted only to applications that use protocols that measure statistics from the communication channel.

4. **From the application layer and by collaborative methods:** In order to bring the advantages that protocols such as RTP and SIP bring to applications that do not have access to communication channel statistics, we envision an alternate method. The method requires including special code in the monitored service, for example JavaScript and AJAX code within a web page on HTTP servers. When the user loads the webpage it also loads special code that executes data transfer from one of the monitored services/applications to the user. The code measures TCP and other data that will compute the efficiency of the communication channel and the data transfer. By computing data from several users, it would be possible to track a health measure of all the communications channels in global and local scopes. Similarly to the other methods described, in case of a DoS attack the measures will denote a decrease in the communication channel's performance. The advantages of this method to compute a reward signal are similar to the method based on the application layer. The disadvantages are that it requires the loading and execution of remote code.

Another line of research that can be further exploited is the analysis of features. So far we have identified a set of features required to identify intrusions but we have not ranked them according to their relevance. Currently it is uncertain if there are more features that can be used to improve the detection rate and which is the optimal number of features. Further investigation in this area could bring evidence in how to improve the detection rate for specific applications (i.e. VoIP, HTTP, etc.) by using specific sets of features.

Finally, we considerer that we can export our proposed MARL of signalling approach to other domains related to computer networks. There are some other domains involving conditions of partial observability and restricted communication where we would like to evaluate our agent architecture. One of these domains is Network Management. On TCP/IP networks, SNMP is used to monitor and to manage networks. The protocol is based on a centralized client-server architecture composed of a central Network

Management Station (NMS) and Network Elements (NE). The NMS sends and receives management information from a NE. This information is used to identify faults, traffic and utilization patterns, configure equipment, etc. This centralized model has proven to be very useful to manage computer and telecommunication networks but according to Boutaba and Xiao [27] it faces important challenges related to scalability. We think that applying our RL-signalling method and the hierarchical architecture approach we could address the scalability problems and the information overhead that central NMS pose. The architecture would be very similar to the IDS domain. In this new architecture, NMS collectors will take the role of sensor agents. NMS collectors will know which network information to pass to higher managements layers by means of the RL-signalling algorithm and the interaction with other NMS collectors and Decision-NMS. After learning, the approach would automatically select high-level events to trigger, thus reducing the overhead that irrelevant events generate to human network managers.

Another domain where we can evaluate our learning approach is the Quality of Service enforcement in converged networks. As mentioned previously in Chapter 4, converged networks supports voice, data and video within the same communications channels and to provide different services and resources depending on the application. Voice and video require minimise delays, jitter and packet loss while data traffic requires maximum bandwidth. We consider that our RL approach could be applied to provide call admission control methods. In this architecture, router and voice gateways would act as sensor agents. Contrary to the NMS case, in this domain we need to include an action to rate-limit connections and voice calls. This action would introduce an environmental feedback that could be used as reward.

Appendix A

Related Publications

Following there is the list of publications that I have made under this subject:

- **Multi-Agent Reinforcement Learning for Intrusion Detection**. Arturo Servin and Daniel Kudenko. In Proceedings Adaptive Learning Agents and Multi Agent Systems. Universiteit Maastricht, Maastricht, The Netherlands. April 2007.
- **Towards Traffic Anomaly via Reinforcement Learning and Data Flow**. Arturo Servin. In Proceedings 1st York Doctoral Symposium on Computing. University of York, York, United Kingdom. October 2007.
- **Multi-Agent Reinforcement Learning for Intrusion Detection**. Arturo Servin and Daniel Kudenko. Lecture Notes in Computer Science Series. Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning. Volume 4865. Springer 2008.
- **Multi-AgentReinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. In Proceedings ALAMAS+ALAg Workshop. Estoril, Portugal. March 2008.
- **Multi-AgentReinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. Frontiers in Artificial Intelligence and Applications; Vol. 178 archive. Short paper in Proceedings

of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence. Pages 873-874. University of Patras, Patras, Greece. July 2008

- **Multi-Agent Reinforcement Learning for Intrusion Detection: A case study and evaluation**. Arturo Servin and Daniel Kudenko. Lecture Notes in Computer Science Series. Proceedings of the 6th German conference on Multi-agent System Technologies. Kaiserslautern, Germany. Volume 5244. Springer 2008

- **Automatic Signature Generation via Reinforcement Learning and Data Flows**. Arturo Servin. In Proceedings CIC 2008: 17th International Conference on Computing. Mexico City. December 2008

List of References

[1] A. Abraham, R. Jain, J. Thomas, and S.Y. Han. D-SCIDS: Distributed soft computing intrusion detection system. *Journal of Network and Computer Applications*, 30(1):81–98, 2007.

[2] O. Abul, F. Polat, and R. Alhajj. Multiagent reinforcement learning using function approximation. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 30(4):485–497, 2000.

[3] M. Afsharchi, B.H. Far, and J. Denzinger. Ontology-guided learning to improve communication between groups of agents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 923–930. ACM New York, NY, USA, 2006.

[4] M.D. Aime, G. Calandriello, A. Lioy, and D.A. e Informatica. A Wireless Distributed Intrusion Detection System and a New Attack Model. In *Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 35–40. IEEE Computer Society Washington, DC, USA, 2006.

[5] D. Anderson, T. Frivold, and A. Valdes. *Next-generation Intrusion Detection Expert System (NIDES): A Summary*. SRI International, Computer Science Laboratory, 1995.

[6] J.P. Anderson. Computer security threat monitoring and surveillance, 1980.

[7] E. Andreasson, F. Hoffmann, and O. Lindholm. Machine Learning for Memory Management. *Proceedings of the 2nd Java Virtual Machine Research and Technology Symposium, USENIX*, 2002.

[8] Nicholas Athanasiades, Randal Abler, John Levine, Henry Owen, and George Riley. Intrusion detection testing and benchmarking methodologies. In *IEEE-IWIA '03: Proceedings of the First IEEE International Workshop on Information Assurance (IWIA'03)*, page 63, Washington, DC, USA, 2003. IEEE Computer Society.

[9] M. Atighetchi, P. Pal, F. Webber, C. Jones, and B.B.N.T. LLC. Adaptive use of network-centric mechanisms in cyber-defense. In *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, pages 183–192, 2003.

[10] Michael Atighetchi, Partha Pal, Franklin Webber, Richard Schantz, Christopher Jones, and Joseph Loyall. Adaptive cyberdefense for survival and intrusion tolerance. *IEEE Internet Computing*, 8(6):25–33, 2004.

[11] The Active Threat Level Analysis System (ATLAS). `http://atlas.arbor.net`, 2008.

[12] B. Awerbuch, D. Holmer, and H. Rubens. Provably Secure Competitive Routing against Proactive Byzantine Adversaries via Reinforcement Learning. *John Hopkins University, Tech. Rep., May*, 2003.

[13] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.

[14] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[15] D. Bagnell and A. Ng. On Local Rewards and Scaling Distributed Reinforcement Learning. *Advances in Neural Information Processing Systems*, 18:91, 2006.

[16] L.C. Baird. Residual algorithms: Reinforcement learning with function approximation. *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.

[17] F. Baker and P. Savola. RFC 3704. Ingress Filtering for Multihomed Networks. Technical report, BCP 84, RFC 3704, March 2004, 2004.

[18] J.S. Balasubramaniyan, J.O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th IEEE Computer Security Applications Conference*, 1998.

[19] P. Barford, S. Jha, and V. Yegneswaran. Fusion and Filtering in Distributed Intrusion Detection Systems. In *Proceedings of the 42nd Annual Allerton Conference on Communication, Control and Computing, September*, 2004.

[20] M. Barreno, B. Nelson, R. Sears, A.D. Joseph, and JD Tygar. Can machine learning be secure? *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[21] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[22] Tim Bass. Intrusion detection systems and multisensor data fusion. *Commun. ACM*, 43(4):99–105, 2000.

[23] BBC. BBC team exposes cyber crime risk. `http://news.bbc.co.uk/1/hi/programmes/click_online/7932816.stm`, 2009.

[24] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[25] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, and M. Embrechts. Network-Based Intrusion Detection Using Neural Networks. *Rensselear Politechnic Institute, New York*, 2002.

[26] D. Bolzoni, S. Etalle, P. Hartel, and E. Zambon. POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. *Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA'06)-Volume 00*, pages 144–156, 2006.

[27] R. Boutaba and J. Xiao. Network management: state of the art. In *Communication Systems: The State of the Art: IFIP 17th World Computer Congress, TC6 Stream on Communication Systems, the State of the Art, August 25-30, 2002, Montréal, Québec, Canada*, page 127. Kluwer Academic Publishers, 2002.

[28] J. Boyan and A.W. Moore. Generalization in Reinforcement Learning: Safely Approximating the Value Function. *Advances in Neural Information Processing Systems*, pages 369–376, 1995.

[29] J.A. Boyan and M.L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems*, 6(1994):671–678, 1994.

[30] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.

[31] JBD Cabrera, L. Lewis, X. Qin, C. Gutierrez, W. Lee, and RK Mehra. Proactive intrusion detection and SNMP-based security management: new experiments and validation. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, pages 93–96, 2003.

[32] JBD Cabrera, L.X.Q.W.L. Prasanth, RK Ravichandran, B. Mehra, R.K.S.S. Co, and MA Woburn. Proactive detection of distributed denial of service attacks usingMIB traffic variables-a feasibility study. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 609–622, 2001.

[33] Joao Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, and Raman K. Mehra. Proactive intrusion detection and distributed denial of service attacks. a case study in security management. *J. Netw. Syst. Manage.*, 10(2):225–254, 2002.

[34] J. Cannady. Applying CMAC-based on-line learning to intrusion detection. *Proceedings of the International Joint Conference on Neural Networks*, 5:405–410, 2000.

[35] J. Cannady. Next Generation Intrusion Detection: Autonomous Reinforcement Learning of Network Attacks. *Proc. 23rd National Information Systems Security Conference*, 2000.

[36] CERT® Coordination Center. CERT® Coordination Center. `http://www.cert.org`, 2002.

[37] CERT® Coordination Center. CERT® Statistics 1988-2005, 2006.

[38] E. Chan, H. Chan, K. Chan, V. Chan, S. Chanson, M.M.H. Cheung, CF Chong, KP Chow, A.K.T. Hui, L.C.K. Hui, et al. IDR: An intrusion detection router for defending against distributed denial-of-service (DDoS) attacks. In *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks*, volume 2004, pages 581–586. IEEE Press, 2004.

[39] E.Y.K.and Chan H.W. Chan, K.M. Chan, and Chan. IDR: an intrusion detection router for defending against distributed denial-of-service (DDoS) attacks. *Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on*, pages 581–586, 2004.

[40] Y.H. Chang, T. Ho, and L.P. Kaelbling. All learning is local: Multi-agent learning in global reward games. *Proceedings of Neural Information Processing Systems (NIPS-03)*, 2003.

[41] CheckPoint. CheckPoint NGX Firewall SmartDefense. `http://www.checkpoint.com/products/ips-1/index.html`, 2008.

[42] S. Chen and Y. Tang. Slowing Down Internet Worms. In *International Conference on Distributed Computing Systems*, volume 24, pages 312–319. IEEE Computer Society; 1999, 2004.

[43] S.P. Chung and A.K. Mok. Allergy Attack Against Automatic Signature Generation. *Lecture Notes in Computer Science*, 4219:61, 2006.

[44] Cisco. Cisco IOS Netflow. `http://www.cisco.com/warp/public/732/Tech/nmp/netflow`, 2008.

[45] Cisco. Cisco Security Monitoring, Analysis, and Response System (MARS) . `http://www.cisco.com/en/US/products/ps6241/,November`, 2008.

[46] Cisco. Configuring Anomaly Detections. `http://www.cisco.com/en/US/docs/security/ips/6.1/configuration/guide/cli_anomaly_detection.html`, 2008.

[47] C. Claus and C. Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 746–752. John Wiley & Sons LTD, 1998.

[48] Secure Computing. Intrusion Prevention Systems (IPS), 2003.

[49] Evan Cooke, Farnam Jahanian, and Danny Mcpherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 39–44, June 2005.

[50] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *DARPA Information Survivability Conference and Expo (DISCEX)*, pages 23–26, 2000.

[51] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems*, 8:1017–1023, 1996.

[52] KDD-99 Data Set for Intrusion Detection Cup. Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999.

[53] D. Dasgupta. Immunity-Based Intrusion Detection System: A General Framework. *Proceedings of the 22nd National Information Systems Security Conference*, pages 147–160, 1999.

[54] D. Dasgupta, F. Gonzalez, K. Yallapu, J. Gomez, and R. Yarramsettii. CIDS: An agent-based intrusion detection system. *Computers & Security*, 24(5):387–398, 2005.

[55] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 240–250, 1992.

[56] Dorothy E. Denning. An Intrusion-Detection Model. *Software Engineering, IEEE Transactions on*, pages 222–232, 1987.

[57] Dorothy E. Denning. *Information warfare and security*. Addison Wesley Reading, Ma, 1999.

[58] T.G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Arxiv preprint cs.LG/9905014*, 1999.

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

[60] DShield; Cooperative Network Security Community. `http://www.dshield.org/indexd.html`, 2008.

[61] R. Eisner. Prediction Measures. `http://www.cs.ualberta.ca/~eisner/measures.html`.

[62] S. Elfwing, E. Uchibe, K. Doya, and HI Christensen. Multi-agent reinforcement learning: using macro actions to learn a mating task. *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 4, 4.

[63] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, 1998.

[64] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, 2001.

[65] S.S. Foundation. DDoS. `http://www.shadowserver.org/wiki/pmwiki.php?n=Stats.DDoS`, 2009.

[66] T.B. Fowler et al. A short tutorial on fractals and internet traffic. *The telecommunications review*, 10:1–14, 1999.

[67] N. Fulda and D. Ventura. Predicting and Preventing Coordination Problems in Cooperative Q-learning Systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[68] M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229, 2006.

[69] LR Halme. AIN'T Misbehaving–A Taxonomy of Anti-Intrusion Techniques. *Computers and Security*, 14(7):606–606, 1995.

[70] B. Hamid and R. Vengerov. Advantages of cooperation between reinforcement learning agents in difficult stochastic problems, 2000.

[71] X. Hang and H. Dai. Applying both positive and negative selection to supervised learning for anomaly detection. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 345–352. ACM New York, NY, USA, 2005.

[72] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical report, LA-SUB–93-219, Los Alamos National Lab., NM (United States); New Mexico Univ., Albuquerque, NM (United States). Dept. of Computer Science, 1990.

[73] LT Heberlein, GV Dias, KN Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*, pages 296–304, 1990.

[74] GG Helmer, JSK Wong, V. Honavar, and L. Miller. Intelligent agents for intrusion detection. In *Information Technology Conference, 1998. IEEE*, pages 121–124, 1998.

[75] A. Hess, H.F. Geerdes, and R. Wessaly. Intelligent Distribution of Intrusion Prevention Services on Programmable Routers. *In Proc. of 25th IEEE INFOCOM*, may 2006.

[76] A. Householder, A. Manion, L. Pesante, G. Weaver, and R. Thomas. Managing the threat of denial-of-service attacks. *CERT Coordination Center*, 10, 2001.

[77] J. Hu and M.P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 242, page 250. San Francisco, California, 1998.

[78] J. Hu and M.P. Wellman. Nash q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003.

[79] J. Huang, B. Yang, and D.Y. Liu. A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 1, 2005.

[80] L. Huang, M. Garofalakis, J. Hellerstein, A. Joseph, and N. Taft. Toward sophisticated detection with distributed triggers. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 311–316. ACM New York, NY, USA, 2006.

[81] R. Huebsch, B. Chun, J.M. Hellerstein, B.T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A.R. Yumerefendi. The Architecture of PIER: an Internet-Scale Query Processor. In *Proc. Conference on Innovative Data Systems Research (CIDR)(Jan. 2005)*, 2005.

[82] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 321–332. VLDB Endowment, 2003.

[83] A. Hussain, J. Heidemann, and C. Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110. ACM New York, NY, USA, 2003.

[84] K.S. Hwang, S.W. Tan, M.C. Hsiao, and C.S. Wu. Cooperative Multiagent Congestion Control for High-Speed Networks. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 35(2):255–268, 2005.

[85] SANS Institute. Sans top-20 2007 security risks, 2007 annual update. `http://www.sans.org/top20/`, 2008.

[86] Internet Storm Center. `http://isc.sans.org/`, 2008.

[87] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of IEEE WETICE 2003*, June 2003.

[88] N.R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[89] K.C. Jim and C.L. Giles. Talking Helps: Evolving Communicating Agents for the Predator-Prey Pursuit Problem. *Artificial Life*, 6(3):237–254, 2000.

[90] P. Kanerva. *Sparse Distributed Memory*. MIT Press, 1988.

[91] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 326–331, 2002.

[92] Peter Vrancx Katja Verbeeck1 and Ann Nowe. Networks of learning automata and limiting games. In *Adaptive Learning Agents and Multi Agent Systems 2007*, pages 171–182, 2007.

[93] R. Khanna and H. Liu. System approach to intrusion detection using hidden Markov model. *International Conference On Communications And Mobile Computing*, pages 349–354, 2006.

[94] J. Kim and P. Bentley. The Artificial Immune Model for Network Intrusion Detection. In *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, 1999.

[95] R.R. Kompella, S. Singh, and G. Varghese. On scalable attack detection in the network. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 187–200. ACM New York, NY, USA, 2004.

[96] R.E. Korf. A simple solution to pursuit games. In *Proc. 11th International Workshop on Distributed Artificial Intelligence*, 1992.

[97] K. Kostiadis and H. Hu. KaBaGe-RL: Kanerva-based generalisation and reinforcement learningfor possession football. *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 1, 2001.

[98] S.C. Lai, W.C. Kuo, and M.C. Hsieh. Defending against Internet worm-like infestations. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, volume 1, 2004.

[99] Shou-Chuan Lai, Wen-Chu Kuo, and Mu-Cheng Hsieh. Defending against internet worm-like infestations. In *AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, page 152, Washington, DC, USA, 2004. IEEE Computer Society.

[100] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542, 1929.

[101] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.

[102] M. Lesk. The new front line: Estonia under cyberassault. *IEEE Security & Privacy*, 5(4):76–79, 2007.

[103] Jun Li and C. Manikopoulos. Early statistical anomaly intrusion detection of dos attacks using mib traffic parameters. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 53–59, June 2003.

[104] Z. Li, Y. Chen, and A. Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 115–122. ACM New York, NY, USA, 2006.

[105] R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

[106] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, volume 157163, 1994.

[107] M.L. Littman, N. Ravi, E. Fenson, and R. Howard. Reinforcement learning for autonomic network repair. In *Proceedings of the First International Conference on Autonomic Computing*, pages 284–285, 2004.

[108] Z. Liu, G. Florez, and S. Bridges. A comparison of input representations in neural networks: a case study in intrusion detection. In *Proceedings of the 2002 International Joint Conference on Neural Networks (ICJNN 02)*, 2002.

[109] M.E. Locasto, J.J. Parekh, A.D. Keromytis, and S.J. Stolfo. Towards collaborative security and p2p intrusion detection. In *Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE*, pages 333– 339, June 2005.

[110] C.E. Loo, M.Y. Ng, C. Leckie, and M. Palaniswami. Intrusion Detection for Routing Attacks in Sensor Networks. *International Journal of Distributed Sensor Networks*, 2(4):313–332, 2006.

[111] D. Malas. SIP End-to-End Performance Metrics. *draft-ietf-pmol-sip-perf-metrics-09 (work in progress)*, 2009.

[112] C. Manikopoulos and S. Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *Communications Magazine, IEEE*, 40(10):76– 82, October 2002.

[113] P. Marbach, O. Mihatsch, M. Schulte, and J.N. Tsitsiklis. Reinforcement Learning for Call Admission Control and Routing in Integrated Service Networks. *Advances in Neural Information Processing Systems*, pages 922–928, 1998.

[114] K. McCloghrie and MT Rose. RFC1213: Management Information Base for Network Management of TCP/IP-based internets: MIB-II. *RFC Editor United States*, 1991.

[115] D. McPherson and Labovitz. Worldwide Infrastructure Security Report, vol. 4, Arbor Networks, 2008.

[116] P. Miller and A. Inoue. Collaborative intrusion detection system. In *North American Fuzzy Information Processing Society, NAFIPS 2003, 22nd International Conference of the*, pages 519–524, 2003.

[117] J. Mirkovic and P. Reiher. D WARD, A Source-End Defense against Flooding Denial of Service Attacks. *Dependable and Secure Computing, IEEE Transactions on*, 2(3):216–232, 2005.

[118] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[119] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[120] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Internet quarantine: Requirements for containing self-propagating code. In *INFOCOM*, 2003.

[121] S. Muftic. *Security Architecture for Open Distributed Systems*. John Wiley & Sons, 1993.

[122] J. Nazario. DDoS attack evolution. *Network Security*, 2008(7):7–10, 2008.

[123] Nazario, J. Jose Nazario on botnets, cyberwarfare. `http://asert.arbornetworks.com/2009/03/jose-nazario-on-botnets-cyberwarfare/`, 2009.

[124] M.R. Neilforoshan. Network security architecture. *Journal of Computing Sciences in Colleges*, 19(4):307–313, 2004.

[125] Peter G. Neumann and Phillip A. Porras. Experience with EMERALD to date.

[126] N.J. Nilsson. Introduction to Machine Learning. *Unpublished draft, Department of Computer Science, Stanford University*, 1996.

[127] A. Nowe, K. Steenhaut, M. Fakir, and K. Verbeeck. Q-learning for adaptive load based routing. *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, 4, 1998.

[128] L. Nunes and E. Oliveira. On Learning by Exchanging Advice. *Arxiv preprint cs/0203010*, 2002.

[129] Tomas Olovsson. A structured approach to computer security, 1992.

[130] Opnet. OPNET. `http://www.opnet.com/`, 2008.

[131] L. Panait, K. Tuyls, and S. Luke. Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective. *The Journal of Machine Learning Research*, 9:423–457, 2008.

[132] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[133] R. Parr and S. Russell. Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Inforamtion Processing Systems*, pages 1043–1049, 1998.

[134] Vern Paxson and Sally Floyd. Wide-area traffic: the failure of poisson modeling. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 257–268, New York, NY, USA, 1994. ACM Press.

[135] C.P. Pfleeger and S.L. Pfleeger. *Security in Computing*. Prentice Hall PTR, 2003.

[136] P.A. Porras, M.W. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. *Lecture Notes in Computer Science*, pages 95–114, 2002.

[137] R. Power. Experts discuss present and future intrusion detection systems. *Computer Security Journal*, 14:1–18, 1998.

[138] R. Powers and Y. Shoham. New criteria and a new algorithm for learning in multi-agent systems. *Advances in Neural Information Processing Systems*, 17:1089–1096, 2005.

[139] D. Precup, R.S. Sutton, and S. Dasgupta. Off-Policy Temporal-Difference Learning with Function Approximation. In *Machine Learning-International Workshop*, pages 417–424, 2001.

[140] X. Qin, W. Lee, L. Lewis, and JBD Cabrera. Integrating intrusion detection and network management. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 329–344, 2002.

[141] M.J. Ranum. Intrusion Detection: Challenges and Myths. *Network Flight Recorder, Inc.*, 1998.

[142] M.J. Ranum. Experiences Benchmarking Intrusion Detection Systems. *NFR Security Technical Publication*, 2001.

[143] Y. Rekhter, B. Moskowitz, D. Karrenberg, GJ de Groot, and E. Lear. RFC1918: Address Allocation for Private Internets. *RFC Editor United States*, 1996.

[144] B.C. Rhodes, J.A. Mahaffey, and J.D. Cannady. Multiple Self-Organizing Maps for Intrusion Detection. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.

[145] R. Richardson. CSI Computer Crime and Security Survey. *Computer Security Institute*, 2008.

[146] M. Roesch. Snort–Lightweight Intrusion Detection for Networks. *LISA XIII*, 1998.

[147] LM Rossey, RK Cunningham, DJ Fried, JC Rabek, RP Lippmann, JW Haines, and MA Zissman. LARIAT: Lincoln adaptable real-time information assurance testbed. *Aerospace Conference Proceedings, 2002. IEEE*, 6, 2002.

[148] J.C. Santamaria, R.S. Sutton, and A. Ram. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2):163, 1997.

[149] J. Schneider, W.K. Wong, A. Moore, and M. Riedmiller. Distributed Value Functions. In *Machine Learning-International Workshop*, pages 371–378. Morgan Kaufmann Publishers, Inc., 1999.

[150] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC3550: RTP: A Transport Protocol for Real-Time Applications. *RFC Editor United States*, 2003.

[151] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, et al. RTP: A Transport Protocol for Real-Time Applications(Request for Comments: 1889), 1996.

[152] Toby Segaran. *Collective Intelligence*. O'Reilly, California, 2007.

[153] S. Sen and G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter Chapter 6, Learning in Multiagent Systems. The MIT Press, New York, NY, USA, July 2000.

[154] S.A. Shaikh, H. Chivers, P. Nobles, J.A. Clark, and H. Chen. Characterising intrusion detection sensors. *Network Security*, 2008(9):10–12, 2008.

[155] S.A. Shaikh, H. Chivers, P. Nobles, J.A. Clark, and H. Chen. Characterising intrusion detection sensors, part 2. *Network Security*, 2008(10):8–11, 2008.

[156] S.A. Shaikh, H. Chivers, P. Nobles, J.A. Clark, and H. Chen. Network reconnaissance. *Network Security*, 2008(11):12–16, 2008.

[157] A.A. Sherstov and P. Stone. Function Approximation via Tile Coding: Automating Parameter Choice. *Lecture Notes in Computer Science*, 3607:194, 2005.

[158] Y. Shoham, R. Powers, and T. Grenager. Multi-agent reinforcement learning: a critical survey. In *AAAI Fall Symposium on Artificial Multi-Agent Learning*, 2004.

[159] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[160] C. Siaterlis and B. Maglaris. Towards multisensor data fusion for dos detection. In *Proc. of the 19th ACM Symposium on Applied Computing, Nicosia, Cyprus*, pages 439–446, 2004.

[161] S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, et al. DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and an Early Prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, 1991.

[162] Inc. SourceFire. Snort. `http://www.snort.org/`, 2008.

[163] W. Stallings. *SNMP, SNMPV2, Snmpv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1998.

[164] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[165] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS-a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pages 361–370. September, 1996.

[166] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, CY Tseng, T. Bowen, K. Levitt, et al. A General Cooperative Intrusion Detection Architecture for MANETs. In *Proceedings of the 3rd IEEE International Workshop on Information Assurance, March*, 2005.

[167] P. Stone and R.S. Sutton. Scaling reinforcement learning toward RoboCup soccer. *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 537–544, 2001.

[168] P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement Learning for RoboCup Soccer Keepaway. *Adaptive Behavior*, 13(3):165, 2005.

[169] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[170] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8:1038–1044, 1996.

[171] R.S. Sutton. Tile Coding Software, Version 2.0., 2007.

[172] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[173] R.S. Sutton and S.D. Whitehead. Online learning with random representations. *Proceedings of the Tenth International Conference on Machine Learning*, pages 314–321, 1993.

[174] K.P. Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

[175] Internet Security Systems. A Guide to Intrusion Detection Technology, Network vs. Host-based Intrusion Detection, 1998.

[176] Y. Takahashi and M. Asada. Multi-controller fusion in multi-layered reinforcement learning. In *Multisensor Fusion and Integration for Intelligent Systems, 2001. MFI 2001. International Conference on*, pages 7–12, 2001.

[177] Y. Takahashi, M. Takeda, and M. Asada. Continuous valued Q-learning for vision-guided behavior acquisition. In *Multisensor Fusion and Integration for Intelligent Systems, 1999. MFI'99. Proceedings. 1999 IEEE/SICE/RSJ International Conference on*, pages 255–260, 1999.

[178] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. *Readings in Agents*, pages 487–494, 1997.

[179] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. *Advances in Neural Information Processing Systems*, 16, 2004.

[180] H. Tong and TX Brown. Adaptive call admission control under quality of service-constraints: a reinforcement learning solution. *Selected Areas in Communications, IEEE Journal on*, 18(2):209–221, 2000.

[181] J. Undercoffer, F. Perich, and C. Nicholas. SHOMAR: An Open Architecture for Distributed Intrusion Detection Services. *University of Maryland, Baltimore County*, 2002.

[182] ISI USC. Network Simulator NS-2 (NS2). `http://www.isi.edu/nsnam/`, 2008.

[183] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[184] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. *Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 11–18, 2003.

[185] S. Whiteson and P. Stone. Concurrent layered learning. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 193–200. ACM New York, NY, USA, 2003.

[186] C Wong, S Bielski, A Studer, and C Wang. Empirical analysis of rate limiting mechanisms. In *Proc. 8th International Symposium on Recent Advances in Intrusion Detection*, 2005.

[187] X. Xu and Y. Luo. A Kernel-Based Reinforcement Learning Approach to Dynamic Behavior Modeling of Intrusion Detection. *Lecture Notes in Computer Science*, 4491:455, 2007.

[188] X. Xu, Y. Sun, and Z. Huang. Defending DDoS Attacks Using Hidden Markov Models and Cooperative Reinforcement Learning. *Lecture Notes in Computer Science*, 4430:196, 2007.

[189] X. Xu and T. Xie. A Reinforcement Learning Approach for Host-Based Intrusion Detection Using Sequences of System Calls. *Proceedings of the International Conference on Intelligent Computing*, 2005.

[190] D. Yagan and C.K. Tham. Adaptive QoS provisioning in wireless ad hoc networks: a semi-MDP approach. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, 2005.

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

[192] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. In *'In Proceedings of NDSS, San Diego, CA, 2004.*, 2004.

[193] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju. VisFlowConnect: netflow visualizations of link relationships for security situational awareness. *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 26–34, 2004.

[194] X.Y. Zhang, C.Z. Li, and Q.G. Hu. The network management design integrated with the intrusion detection system. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 1, 2004.

[195] X.Y. Zhang, C.Z. Li, and W. Zheng. Intrusion Prevention System Design. *Computer and Information Technology*, pages 386–390, 2004.

[196] Y.F. Zhang, Z.Y. Xiong, and X.Q. Wang. Distributed Intrusion Detection Based on Clustering. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 4, 2005.