BUILDING THE INTELLIGENT IOT-EDGE: BALANCING SECURITY AND FUNCTIONALITY USING DEEP REINFORCEMENT LEARNING

by

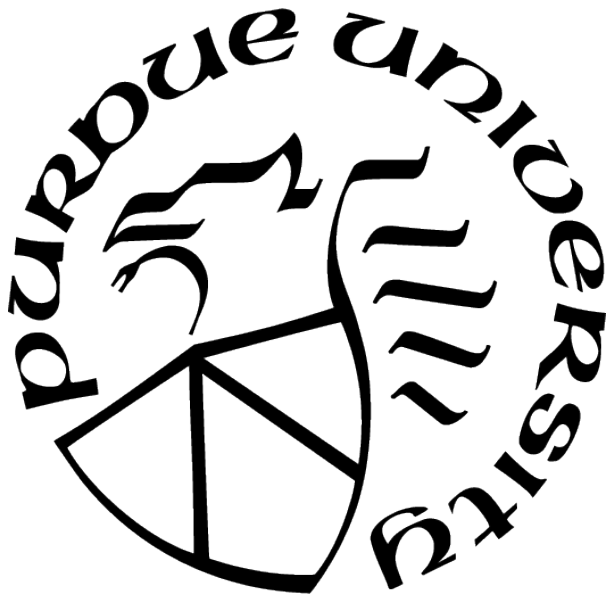
Anand Mudgerikar

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



Department of Computer Science

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Elisa Bertino, Chair

School of Computer Science

Dr. Puneet Sharma

HPE Labs

Dr. Clifton W. Bingham

School of Computer Science

Dr. Chunyi Peng

School of Computer Science

Dr. Ninghui Li

School of Computer Science

Approved by:

Dr. Kihong Park

Dedicated to my parents who have always
supported me in all my endeavors.

ACKNOWLEDGMENTS

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to gratefully acknowledge my PhD advisor Dr. Elisa Bertino, for her continued support and guidance throughout this project. Your insightful feedback and ideas pushed me to sharpen my thinking and brought my work to a higher level.

I would like to thank my mentor during my internships at HPE labs, Dr. Puneet Sharma whose expertise was invaluable in formulating the research questions and methodology.

I would also like to acknowledge the insightful comments and feedback provided by DAIS-ITA group which has helped shape the direction of the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IoT	Internet of Things
SDN	Software Defined Networks
RL	Reinforcement Learning
DRL	Deep-Reinforcement Learning
IDS	Intrusion Detection System
IPS	Intrusion Prevention Learning
DoS	Denial of Service
DDoS	Distributed Denial of Service
VLAN	Virtual Local Area Network
SLA	Service Level Agreement

ABSTRACT

The exponential growth of Internet of Things (IoT) and cyber-physical systems is resulting in complex environments comprising of various devices interacting with each other and with users. In addition, the rapid advances in Artificial Intelligence are making those devices able to autonomously modify their behaviors through the use of techniques such as reinforcement learning (RL). There is thus the need for an intelligent monitoring system on the network edge with a global view of the environment to autonomously predict optimal device actions. However, it is clear however that ensuring safety and security in such environments is critical. To this effect, we develop a constrained RL framework for IoT environments that determines optimal devices actions with respect to user-defined goals or required functionalities using deep Q learning. We use anomaly based intrusion detection on the network edge to dynamically generate security and safety policies to constrain the RL agent in the framework. We analyze the balance required between 'safety/security' and 'functionality' in IoT environments by manipulating the exploration of safe and unsafe benefit state spaces in the RL framework. We instantiate the framework for testing on application layer control in smart home environments, and network layer control including network functionalities like rate control and routing, for SDN based environments.

1. INTRODUCTION

The deployment of Internet of Things (IoT) combined with cyber-physical systems has resulted in complex environments comprising of various devices interacting with each other and with users through apps running on computing platforms like mobile phones, tablets, and desktops. The development of communication protocols for IoT devices, such as 6LowPAN, CoAp, and Zigbee, and the progresses in AI have made it possible to interconnect these different IoT devices and achieve smart autonomous IoT systems. However, smart IoT-based systems require interconnection and inter-operation among devices, apps, users and the edge. Such a complex environment of IoT devices and apps dynamically interacting with each other is prone to security/safety issues [1]–[3]. Another issue is that, in terms of functionality, each app has specific individual goals. However, when selecting the actions to be executed, the app does not take into consideration the global view of the environment where the IoT device is deployed. Such a lack of awareness can lead to decisions and actions that are not globally optimal in terms of user requirements or goals. Evidently, there is a need for 'intelligent' monitoring systems to ensure safety and maximize functionality with a global view of all devices and apps and their interactions.

A truly 'intelligent' system requires that the IoT devices in the environment work in concert intelligently to support users in carrying out their activities in a 'safe' and 'optimal' way using information and intelligence that is hidden in the network connecting these devices, requiring little to no management on a user's part, and in making intelligent decisions based on historical and real-time data. The network edge is the ideal place for deployment of such intelligent monitoring systems as it has access to real-time and historical data required for learning along with significant computational and storage power to run powerful machine learning methods. With the recent advances in edge-computing, the fog computing paradigm [4], [5] used for developing our framework provides advantages over traditional distributed and cloud computing in terms of security, privacy, scalability, reliability, speed and efficiency. On top of this, the emergence and successful deployment of software defined networks (SDN), zero-trust security architecture, and network function virtualization (NFV) in large scale modern enterprise and 5G networks, it is possible to build "smart" network

controllers that leverage machine learning (ML) techniques to learn policies for optimal and secure traffic engineering.

With the rapid advances in Artificial Intelligence, Reinforcement Learning (DRL) has emerged as a powerful tool for making these devices be able to autonomously modify their behaviors. Application of DRL to IoT and SDN environments is a lucrative notion. However, application of any machine learning intelligence with the single goal of optimization would never account for unsafe states being reached because of the environment parameters. Previous work has focused on building application layer RL based solutions to optimize IoT environments with specific goals, like energy management [6], [7], optimal resource allocation [8], minimization of energy prices [9] etc. Similar approaches have been used for optimizing network layer functionalities like routing in SDN environments such as routing [10]–[13] and rate control/load balancing [14], [15]. The drawback of those RL frameworks, as mentioned before, is that they focus on optimizing certain goals but do not take into account safety and security of the environment.

1.1 Research Problem

There is the need for an autonomous control system deployed at the network edge able to support applications/users, by providing optimal device actions at the application and network layers to maximize QoS for the users in terms of required functionalities but at the same time maintaining safety and security the monitored IoT environment.

1.2 Challenges

Our approach to address such a research problem is based on the use of DRL techniques constrained by the use of safety and security policies. However, the design of such an approach for complex IoT environments requires addressing several challenges:

1. Security and safety policies depend on the type of intrusions detected in environments which often have various sources, stages and vectors of infection like malware, network based attacks, malicious insiders, side-channel attacks, benign user compromises, botnet attacks etc. Since the attack surface is growing rapidly and changes with the environment,

a challenge is thus to be able to specify suitable safety and security policies dynamically for different IoT environments, deployed devices.

2. RL frameworks learn by 'freely' exploring their environment (state space) in order to find the optimal policies. The challenge is to build a RL framework where the agent is 'constrained' by these security/safety policies but at the same time is 'free to explore' in order to learn 'optimal' and 'safe' actions in the IoT ecosystem. A challenge is thus to identify the correct trade-off between 'constraint' in terms of 'unsafe state space' and 'freedom' in terms of 'safe state space' in agent exploration.
3. All RL frameworks work on a model (simulation) of the environment or the environment itself. Building a model for an IoT environment is challenging because of inherently erratic distributions of variables like noise, user errors, malfunctions etc. A completely model-free approach requires actual experiences in order for training, which makes exploration more dangerous.
4. RL frameworks should be flexible in the sense of being applicable to different environments with minimum human effort. Thus Ideally, RL frameworks should not only learn optimal policies, that is, state-action pair rewards, from their experiences but also learn about the model itself or more specifically state transition probabilities.
5. A policy learnt for one IoT environment very often cannot be directly applied to other IoT environments because of differences in devices, users, environment specific functions etc. A challenge is thus to use knowledge gained from learnt policies in one environment and apply them in the context of a different environment.
6. A single point of failure for security in such environments is the central SDN controller itself which can be compromised through network layer attacks like DoS, DDoS, Brute-Force and web based attacks. A challenge is thus to learn optimal policies for the SDN edge controllers along with application layer edge controllers. This requires learning security constrained optimization policies for core network functionalities like routing, rate control etc.

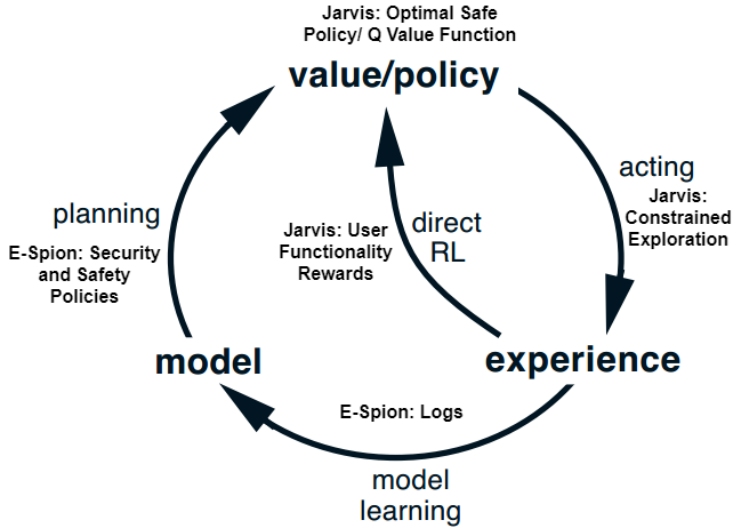


Figure 1.1. High Level Overview of Learning Architecture

1.3 Proposed Solution

Towards addressing the stated research problem, we have designed an architecture with three main components: `E-Spion`, `Jarvis` and `Jarvis-SDN`. The security and safety policies learnt from `E-Spion` logs are used to build the model or state transition probabilities of the `Jarvis` DRL framework as seen in figure 1.1. Formally, `Jarvis` and `Jarvis-SDN` can be defined as model-based RL frameworks based on the Dyna-Q framework [16], where: (i) the model of the environment in terms of safety/security policies is learnt from `E-Spion`, and (ii) the optimal policies for the learnt model are learnt by the RL agent from simulated experiences. We provide more details about each of the three components as follows.

1.3.1 E-Spion: Edge Based Intrusion Detection for IoT environments

`E-Spion` is anomaly-based system level Intrusion Detection System (IDS) for IoT devices. It profiles IoT devices according to their 'behavior' using system level information, like running process parameters and their system calls, in an autonomous, efficient, and scalable manner. These profiles are then used to detect anomalous behaviors indicative of intrusions. The modular design of our IDS along with a unique device-edge split architecture allows

for effective attack detection with minimal overhead on the IoT devices. Our device profile is built in three layers using three types of device logs (one per each layer) obtained from three types of information: running process names, running process parameters, and system calls made by these processes. Since each of these log types has different overheads for recording, storage, and analyzing, we maintain three separate modules which handle each type of device log, namely PWM, PBM, and SBM. These modules can run concurrently with different configuration values (recording intervals, sleep times etc.) according to the device/network requirements (resource consumption, associated risk etc.). The modules interact with each other using the common module manager to improve overall detection efficiency, provide more fine grained intrusion alerts, and reduce overhead on the devices. The security and safety policies for the environment can be extracted from the `E-Spion` logs by searching for anomalous behaviors.

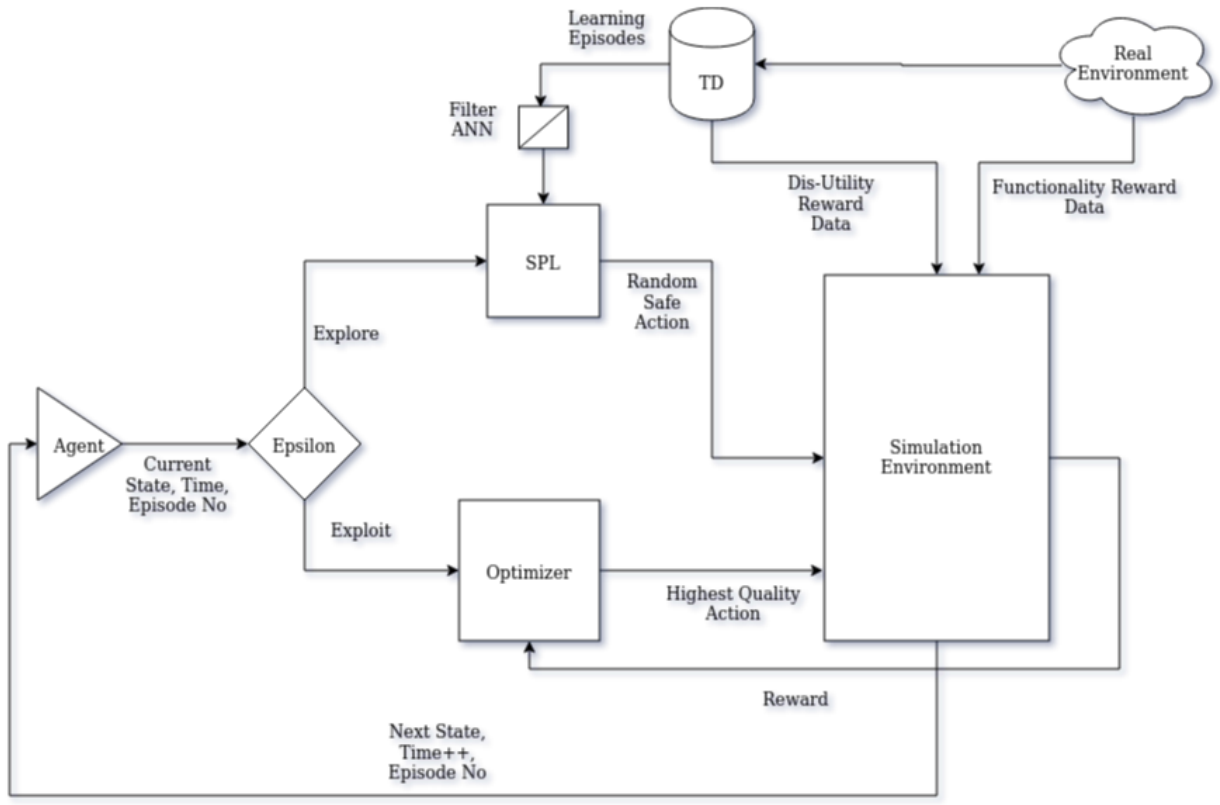


Figure 1.2. `Jarvis` RL Framework

1.3.2 `Jarvis`: A Constrained DRL Framework tailored for IoT environments

By observing the specific IoT environment, `Jarvis` first dynamically builds a simulated environment in terms of devices states and actions. An agent, constrained by security policies, can traverse the simulated environment in multiple episodes of specific time periods and find the optimal safe actions in terms of functionality requirements provided by the user. A Deep Q learning network (DQN) is used to determine the highest rewarding (quality) actions for each environment state and *time instance*. A Q learning approach is ideal for such an environment/ecosystem where, for each state-action pair, we can determine its quality through a cumulative reward for the time period in terms of the user goals. We train the agent using a deep neural network (DNN) to maximize the cumulative reward and thus generate the optimal quality function. A high level outline of the framework is shown in figure 3.3. It is important to note that, the security and safety policies used in the `Jarvis` framework can be obtained dynamically from `E-Spion` logs or through other attack signatures of IDS systems available offline.

1.3.3 `Jarvis-SDN`: A Constrained DRL Framework for SDN environments

`Jarvis-SDN` is based on a similar RL framework used for `Jarvis` but operates at the network layer rather than the application layer. The goal of the RL framework is optimal network control. Although there are numerous aspects to network control, in this thesis we focus on the two core network functionality components required for successful operation of a network: 1. Routing: Determines the path to take for transferring packets intended from node A to B to minimize latency, and 2. Rate Control: Determines how much bandwidth or priority to allocate for each network session between nodes A and B to meet user SLAs. The security constraints are then encoded as risk metrics into the `Jarvis-SDN` optimization criteria to protect the network controller against DoS, DDoS, Brute-Force and Web based attacks but at the same time learn optimal networking policies. For example, in terms of routing, ideally a suspicious network flow, which could be a DDoS attack, should be routed through nodes[1] running a DDoS IDS/IPS on the network. In terms of rate control, the

[1]↑By node, we refer to a gateway/access point of an autonomous system (AS) or VLAN network segment

malicious looking flows should be throttled until an accurate inference can be made by the IDS/IPS.

1.4 Thesis Outline

The rest of this thesis is organized as follows. First, in Chapter 2, we review IDS/IPS systems designed for IoT and develop E-Spion, a host based IDS specifically for IoT environments. In Chapter 3, we define the model of `Jarvis` for application layer control in IoT smart home environments. In Chapter 4 and Chapter 5, we define the model of `Jarvis-SDN` for network layer rate control and routing in SDN environments, respectively.

The instantation of our DRL model `Jarvis` for application layer control and `Jarvis-SDN` for network control requires addressing a key challenge, that is, the application of these techniques at scale. A pure offline RL model degrades with scale as the quality of the simulation environment also degrades. On the other hand, learning safety and security policies in a purely online manner is not feasible for obvious reasons. This is a fundamental problem in application of RL models to real world environments. In 6, we discuss this challenge in more detail and provide a solution to this challenge using a Generative Adversarial Network (GAN) based on the approach of augmenting explorations in the offline environment with real explorations in the online environment. Finally, in chapter **??**, we provide a conclusion and avenues for future work.

2. E-SPION

With the growing use of IoT devices in health care, transportation, home appliances and smart cities, security of these devices is a primary concern. The lack of security is evident in the huge number of IoT devices that have been compromised and exploited for launching large scale DDoS attacks like Mirai in 2016, Hajime in early 2017, BrickerBot and IoT Reaper later in 2017 and Hakai in 2018[17], [18]. The alarming fact about these attacks is that these malware are basic and straight-forward when compared to malware used to attack traditional computer systems. We should thus expect more sophisticated IoT malware in the future. It might even be the case that such advanced malware are already present on current devices going completely undetected. An example of such evolution of IoT malware is the Reaper malware [19] which is a successor to Mirai. It builds on Mirai's code by adding various known exploits for different IoT devices/architectures, along with the simple password guessing techniques of Mirai, to its infection methods. Recently, file-less attacks have been observed along with traditional malware based attacks against IoT devices[20]. Such attacks are significantly harder to detect and comprise a high percentage of recently seen attacks in the wild.

To keep up with evolution in malware, we must also keep evolving our security techniques. Achieving comprehensive security for IoT devices and systems requires combining different layers of security techniques and systems[21] – among which intrusion detection is one of the most important. Thus, evolution of intrusion detection techniques for IoT devices is of utmost importance.

There has been significant research in designing intrusion detection systems (IDSes) for traditional computer systems, such as Snort [22] and Bro [23], and more recently for IoT devices, such as Svelte [24], Kalis [25] and Heimdal [26]. Most IDSes for IoT devices are network traffic based and detect attacks by analyzing the network traffic for either anomalies or attack signatures. The main problem of these IDSes is that, because of the large amounts of network traffic in today's IoT networks coupled with the heterogeneity of IoT devices in terms of protocols, manufacturers, applications, etc., they tend to miss some attacks and

have a significant number of false positives. To overcome those shortcomings, we propose `E-Spion`, which monitors and analyzes system data rather then network traffic of IoT devices.

Motivation. System level IDSes like anti-viruses, commonly used for traditional computer systems, employ attack signature based detection. System level anomaly based detection in such IDSes is not practical as a traditional computer system runs a number of different kinds of applications which might be very similar to malware in terms of computing operations and commands. So, it becomes very difficult to differentiate between benign applications and malware, resulting in high numbers of false positives and false negatives. This, however, is not the case with IoT devices.

In general, IoT devices have a primary function for which computation is required. For example, a DVR is meant to record and store videos but computation is required for this, like receiving an input video stream on a network socket and then storing the files in a local database. Similarly, a television is meant to display, a camera is meant to record videos, a car is meant to drive safely etc. We see that the computations done on the device are a means to an end or the main function. This is not the case with traditional computer systems, like desktops and laptops, where computation is the end goal. These devices can run multiple applications and allow us to perform computations, like browsing the Internet, performing calculations, playing games and so on. So, we see intuitively that the IoT devices, like DVRs, cameras, cars, televisions etc., have a main function and these IoT devices, when not compromised, should be performing just this main function and nothing else. We also note that these main functions are periodic in nature and consistently repeat themselves with different arguments after device specific time intervals. We use these intuitions to build IoT device profiles later used for anomaly detection.

Approach. At a high level, the goal of our IDS is to identify the main functions of the IoT device in terms of data collected from the running processes and system calls made by these processes, in order to create the baseline behavior profile of the IoT device. We then continuously monitor the device and use this baseline to detect anomalous behavior that may be indicative of intrusions or other malicious activities.

Contributions. To summarize, we make the following contributions in this chapter:

1. A system level IDS that uses anomaly detection to detect attacks on IoT devices in an efficient and scalable manner.
2. An approach to build baseline behavior profiles of IoT devices according to system information (device logs) collected from running processes and system calls on these devices.
3. A device-edge split architecture with the server components running on the network edge-server performing the bulk of the computational work and the components running on the IoT device performing minimal work.
4. A three layer anomaly detection engine with each more advanced layer providing more fine-grained accuracy but at the same time having higher overhead costs on the device.
5. An extensive evaluation of our system using 3973 malware samples assembled from recent attacks on IoT devices and an analysis of the malware samples in terms of our device logs.
6. The first IDS for IoT devices effective against 8 types of file-less attacks.

The rest of the chapter is organized as follows. First we give some background on how IoT attacks are propagated, different types of IoT malware, file-less attacks and the IoT security architecture in general. Then we discuss in detail the design of our system and its components in Section 5.4. Next we discuss some of the key implementation challenges and analyze the design decisions made in Section 2.3. We perform the evaluation in terms of detection efficiency and overhead costs in Section 3.5. In Section 5.6, we discuss and compare related work in this area and finally conclude in Section 5.7.

2.1 Background

2.1.1 IoT Attacks

Most of the IoT attacks comprise of three operation stages: injection, infection and attack. The *injection stage* involves gaining 'control' of the IoT device (getting root access in most cases) through credential hijacking, password brute-forcing/dictionary attacks or utilizing known device/system/firmware vulnerabilities [27], [28]. The injection stage follows in which the attacker 'prepares' the device by performing operations on the device like setting up communication with the bot master (C&C server), downloading required malware/rootk-

its, stopping security services etc. Most of the popular IoT attacks involve downloading some form of malware on to the device in the 'prepare' stage but the recently seen file-less attacks do not follow this trend. Some attacks might even skip the prepare step and proceed directly to the attack stage. Finally the 'attack' stage includes performing Denial of Service (DoS), coordinated DDoS attacks, ransom attacks, bitcoin mining, device specific malicious/unsafe activities etc. [29], [30].

2.1.2 File-less Attacks

Recently another class of attacks has been observed which does not involve downloading any malware during the infection stage. This makes attacks harder to detect since malware fingerprinting is the primary detection technique used in traditional IDSs. The attacks involve setting up back-doors, port-forwarding etc. using shell scripting or modifying system-level files. They can be classified into 8 categories [20] described below which we use for our evaluation.

1. Type 1: Changes the password of the device using the *passwd* command; it thus locks up the device and does not allow legitimate users to access it.
2. Type 2: Removes certain config files or system programs using the *rm* or *dd* commands. The goal is to remove watchdog and other security service daemons so that the infection/malfunction is not detected.
3. Type 3: Stops certain system processes or services using the *kill* or *service* commands. The goal is similar to Type 2 in that the infection is not detected.
4. Type 4: Retrieves system information like architecture, operating system and networking/process information using commands like *lscpu, uname, netstat, ps* etc. The goal is to gain more information about the device and network for further attacks.
5. Type 5: Steals user information using the *cat* command to read stored hashed passwords, config files etc. The goal is to breach privacy, learn user information and/or behavior patterns.
6. Type 6: Launches network attacks through malformed HTTP requests to web servers using the *wget* or *curl* commands exploiting known vulnerabilities like HeartBleed, SQL

injection etc. The goal is to propagate the attack in the local network and compromise other devices on the network.

7. Type 7: Uses various shell commands for collecting device/user data like *who, help, lastlog, sleep* etc. The goal is to learn how the user uses the device and if other users are using the device.
8. Type 8: Sets up port forwarding using either the *ssh* or *iptables* utilities in order to use the device as a port forwarding proxy so that the real IP address of the attacker is masked.

2.1.3 IoT Security Architecture

The IoT security solutions and services can be broadly classified into two categories: centralized cloud based and distributed edge based. The current trend in enterprise security is towards providing a security overlay network [31] using a centralized cloud architecture where the service providers have primary ownership of the data and IDS service. The advantage of such services is the flexibility in deployment and management, lower infrastructure costs, performance benefits and a centralized point of control. However, a completely centralized cloud based security architecture is not scalable [32], [33] for the distributed nature of an IoT environment because of the huge number of devices, high volume of data, low-latency requirements, ad-hoc environments and user privacy concerns. The edge based security architecture proposed in this chapter follows the fog computing paradigm [4], [5] where the main workload of the IDS is performed at the edge device rather than at the cloud. It is important to note that the cloud computing resources can be utilized as an additional layer over the edge as discussed later in Section 2.3.

2.2 Design

In this section, we first provide an overview of our system design. We then discuss in detail our anomaly detection engine and the typical life cycle of a device in the network. Finally, we detail our log authentication scheme (hash-chain-verifier) for secure device log transfer.

2.2.1 Design Overview

Our system, called `E-Spion`, proposes a novel device-edge split architecture with two components: a server side (Edge-Server) and a client side (Device) (see Figure 2.1). The server component is maintained on the edge system or gateway router of the network. The client component is installed on each IoT device connected to the edge system. The client component is responsible for recording all system logs on the device and periodically transferring them to the edge server. The edge-server and IoT devices communicate periodically via a secure encrypted and authenticated channel. All the computationally intensive operations, i.e. parsing logs to extract features, authentication of logs, training classifiers, running classifier models, module management etc., are performed on the edge-server. We have employed such a local edge compute strategy to minimize workload on the IoT devices.

Our device 3-layered behavior profile is built in three layers using three types of device logs (one for each layer) obtained from three types of information: running process names, running process parameters, and system calls made by these processes. Since each of these log types has different overheads for recording, storing and analyzing, we maintain three separate modules which handle each type of device log, namely Process White listing Module (PWM), Process Behavior Module (PBM), and System-call Behavior Module (SBM). These modules can run concurrently with different configuration values (recording intervals, sleep times etc.) according to the device/network requirements (resource consumption, associated risk etc.). All three modules are managed by a module manager which helps them interoperate efficiently and according to the user/device requirements. The modules interact with each other using the common module manager to improve overall detection efficiency, provide more fine grained intrusion alerts and reduce overhead on the devices.

2.2.2 Anomaly Detection Engine

As stated before, our anomaly detection engine is organized into three detection modules as follows:

PWM. This module uses a white listing based approach and is the least expensive module. In the learning stage, our system monitors all the benign processes running on the

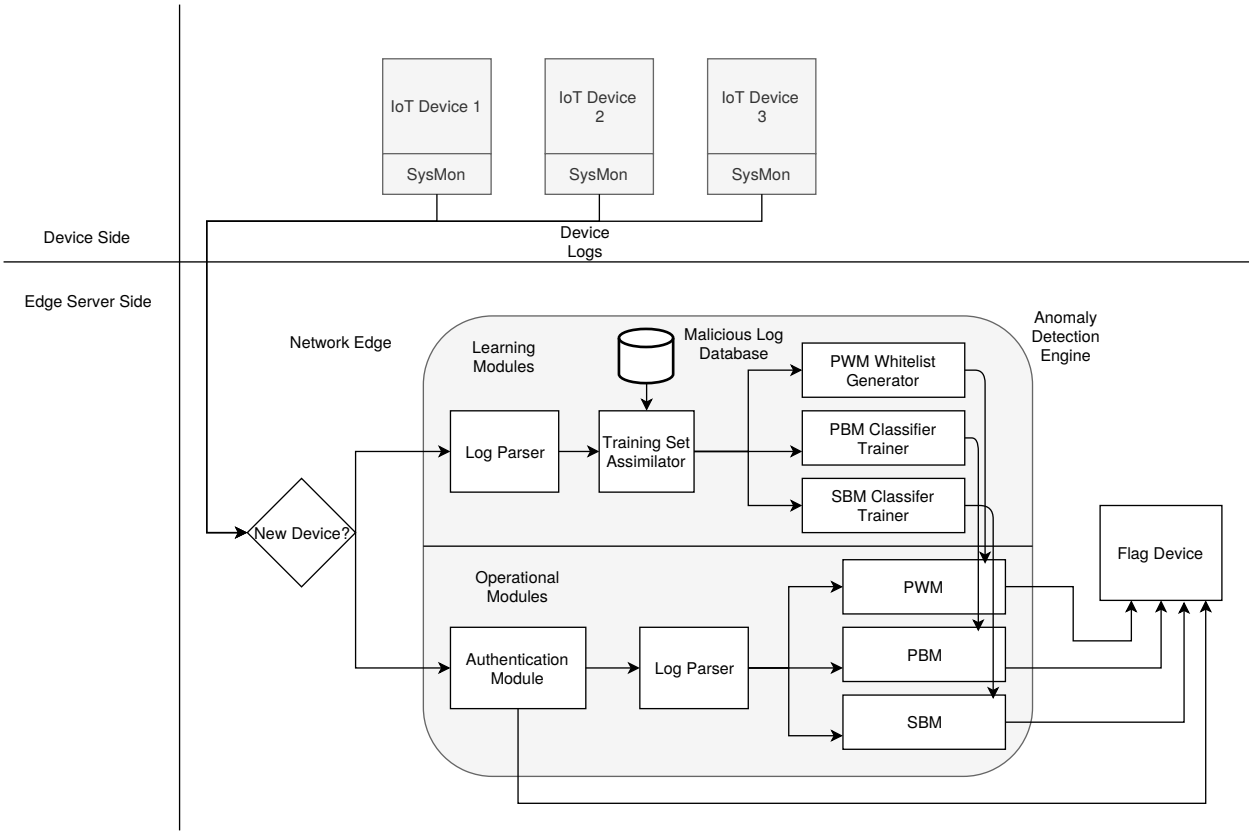


Figure 2.1. Architecture of `E-Spion`

device and builds a device specific white-list of all running processes. The system collects all the running process names and PIDs during the operation stage and compares them with the device white-list to detect anomalous new processes. The goal is to detect simple malware which spawn new malicious processes on the device, as efficiently as possible.

PBM. This module monitors the behavior of each running process on the device and detects any process behaving anomalously. The module monitors various parameters for each running process on the device during the learning stage. After collecting the logs during the learning stage, we train a machine learning classifier for each device and use it for detecting anomalous process behavior. We extract 8 metrics/features (see Table 2.1) from the parameters and use them in our PBM classifier model. This module is more expensive than the PWM but it provides more fine-grained detection and is able to detect more sophisticated malware that have the ability to masquerade as benign processes rather than spawning new malicious processes.

Table 2.1. Metrics for the *PBM* module

Metric #	Metric Name	Description
1	SysCPU	CPU time consumption of the process in system mode (kernel mode)
2	UsrCPU	CPU time consumption of the process in user mode
3	CPU Usage	Overall CPU utilization
4	RGROW	The amount of resident memory that the process has grown during the last interval
5	VGROW	The amount of virtual memory that the process has grown during the last interval.
6	WRDSK	The number of write accesses issued physically on disk
7	RDDSK	The number of read accesses issued physically on disk
8	Instance Count	The number of instances of the process spawned in the interval

SBM. This module monitors the behavior of each running process on the device according to system calls issued by the process. This is the most expensive module but it provides the most effective and fine-grained detection strategy. The module monitors 34 different kinds of system calls issued by each running process as described in Table 2.2. We list the metrics/features used for training our classifier in Table 2.3. For each type of system call recorded, we monitor four metrics/features: number of calls made (#.1), %of time taken (#.2), total time taken (#.3) and average time taken per each call (#.4). So, in total we have $34 * 4 = 136$ metrics in our SBM classifier model.

2.2.3 Life Cycle of a Device

For the purpose of our IDS system, a device in the network goes through the following phases: Initialization, Learning, Operation, and finally Anomaly Detection.

Initialization Phase: When a new device joins the network, a new empty device profile and public-private key pair are created on the edge-server. Then the private key is uploaded to the device and all SSH sessions between the edge and device are secured using this key pair. The corresponding device architecture (Mips, Arm, x86, Spark etc.) specific client side modules are transferred to the device. The client side modules include Linux shell script recording modules for the PWM, PBM and SBM modules, scripts to create and maintain hash chains for log authentication and finally scripts to securely transfer logs from the device to the edge.

Learning Phase: In this phase, the edge-server receives PWM, PBM and SBM logs from the device and builds a single 3-layered baseline profile for the device. The PWM logs

Table 2.2. System calls monitored by *SBM*

System Call	Description
connect	initiate a connection on a socket
_newselect	synchronous I/O multiplexing
close	close a file descriptor
nanosleep	high-resolution sleep
fcntl64	manipulate file descriptor
socket	create an endpoint socket for communication
rt_sigprocmask	examine and change blocked signals
getsockopt	get and set options on sockets
read	read from a file descriptor
open	open and possibly create a file
execve	execute program
chdir	change working directory
access	check user's permissions for a file
brk	change data segment size
ioctl	manipulates the underlying device parameters of special files
setsid	creates a session and sets the process group ID
munmap	map or unmap files or devices into memory
wait64	wait for process to change state
clone	create a child process
uname	get name and information about current kernel
mprotect	set protection on a region of memory
prctl	operations on a process
rt_sigaction	examine and change a signal action
ugetrlimit	get/set resource limits
mmap2	map or unmap files or devices into memory
fstat64	get file status
getuid32	returns the real user ID of the calling process
getgid32	returns the real group ID of the calling process.
geteuid32	returns the effective user ID of the calling process.
getegid32	returns the effective group ID of the calling process.
madvise	give advice about use of memory
set_thread_area	set a GDT entry for thread-local storage
get_tid_address	set pointer to thread ID
prlimit64	get/set resource limits

are used to build the baseline process white list for the device. The data collected from the PBM and PWM logs in the learning stage combined with pre-recorded malicious data serves as the training dataset for these modules. Using these training sets, device specific binary classifiers are built for both PBM and SBM modules. We discuss in detail our design choice of binary classifiers over unary classifiers in Section 2.3.

Training Set Creation and On-the-Fly Classifier Training: In order to train the binary classifiers for PBM and SBM modules for each new device, we require both benign and malicious labeled logs in our training set. In our threat model [34], we assume that the device is uncompromised during the learning stage. In practice, one can also consider the

Table 2.3. Metrics for the *SBM* module

Metric #	Metric Name #.1	Metric Name #.2	Metric Name #.3 (seconds)	Metric Name #.4 (usecs/call)
1	No. of connect calls	%Time of connect calls	Time taken by connect calls	Time/call of connect calls
2	No. of _newselect calls	%Time of _newselect calls	Time taken by _newselect calls	Time/call of _newselect calls
3	No. of close calls	%Time of close calls	Time taken by close calls	Time/call of close calls
4	No. of nanosleep calls	%Time of nanosleep calls	Time taken by nanosleep calls	Time/call of nanosleep calls
5	No. of fcntl calls	%Time of fcntl calls	Time taken by fcntl calls	Time/call of fcntl calls
6	No. of socket calls	%Time of socket calls	Time taken by socket calls	Time/call of socket calls
7	No. of rt_sigprocmask calls	%Time of rt_sigprocmask calls	Time taken by rt_sigprocmask calls	Time/call of rt_sigprocmask calls
8	No. of getsockopt calls	%Time of getsockopt calls	Time taken by getsockopt calls	Time/call of getsockopt calls
9	No. of read calls	%Time of read calls	Time taken by read calls	Time/call of read calls
10	No. of open calls	%Time of open calls	Time taken by open calls	Time/call of open calls
11	No. of execve calls	%Time of execve calls	Time taken by execve calls	Time/call of execve calls
12	No. of chdir calls	%Time of chdir calls	Time taken by chdir calls	Time/call of chdir calls
13	No. of access calls	%Time of access calls	Time taken by access calls	Time/call of access calls
14	No. of brk calls	%Time of brk calls	Time taken by brk calls	Time/call of brk calls
15	No. of ioctl calls	%Time of ioctl calls	Time taken by ioctl calls	Time/call of ioctl calls
16	No. of setsid calls	%Time of setsid calls	Time taken by setsid calls	Time/call of setsid calls
17	No. of munmap calls	%Time of munmap calls	Time taken by munmap calls	Time/call of munmap calls
18	No. of wait calls	%Time of wait calls	Time taken by wait calls	Time/call of wait calls
19	No. of clone calls	%Time of clone calls	Time taken by clone calls	Time/call of clone calls
20	No. of uname calls	%Time of uname calls	Time taken by uname calls	Time/call of uname calls
21	No. of mprotect calls	%Time of mprotect calls	Time taken by mprotect calls	Time/call of mprotect calls
22	No. of prctl calls	%Time of prctl calls	Time taken by prctl calls	Time/call of prctl calls
23	No. of rt_sigaction calls	%Time of rt_sigaction calls	Time taken by rt_sigaction calls	Time/call of rt_sigaction calls
24	No. of ugetrlimit calls	%Time of ugetrlimit calls	Time taken by ugetrlimit calls	Time/call of ugetrlimit calls
25	No. of mmap2 calls	%Time of mmap2 calls	Time taken by mmap2 calls	Time/call of mmap2 calls
26	No. of fstat calls	%Time of fstat calls	Time taken by fstat calls	Time/call of fstat calls
27	No. of getuid calls	%Time of getuid calls	Time taken by getuid calls	Time/call of getuid calls
28	No. of getgid calls	%Time of getgid calls	Time taken by getgid calls	Time/call of getgid calls
29	No. of geteuid calls	%Time of geteuid calls	Time taken by geteuid calls	Time/call of geteuid calls
30	No. of getegid calls	%Time of getegid calls	Time taken by getegid calls	Time/call of getegid calls
31	No. of madvise calls	%Time of madvise calls	Time taken by madvise calls	Time/call of madvise calls
32	No. of set_thread_area calls	%Time of set_thread_area calls	Time taken by set_thread_area calls	Time/call of set_thread_area calls
33	No. of get_tid_address calls	%Time of get_tid_address calls	Time taken by get_tid_address calls	Time/call of get_tid_address calls
34	No. of prlimit calls	%Time of prlimit calls	Time taken by prlimit calls	Time/call of prlimit calls

scenario where device vendors can perform the learning phase in their isolated environments and provide device profiles to their users. So, all logs obtained during the learning stage can also be assumed to be benign. In order to obtain malicious labeled logs, we emulated different CPU architectures and firmwares using qemu [35] system level emulation and [36]. On these device specific virtual machines, we ran a portion of the IoT malware samples and collected the device logs for various *interval* values from all malicious processes spawned for different CPU architectures. The generation of maliciously labeled logs is done offline, that is, before the initialization phase, in order to create pre-recorded malicious logs for different CPU architectures, endianess and *intervals*. We store these in the malicious log database. During the learning stage, the training set assimilator creates device specific training data by combining benign labeled logs from the learning phase and malicious labeled logs from the the malicious log database according to the device CPU architecture and *interval*. Our binary classifier is then trained during the learning stage to distinguish between malicious

and benign logs. These classifier models for the PBM and the SBM modules along with the running process white-list for the PWM module are combined together to form the complete behavioral baseline profile for the device. In our current implementation, we choose a random forest classifier for both the PBM and SBM modules as it provides the highest detection efficiency. We discuss more about the performance of various binary classifiers in Section 2.3.

Operational Phase: Once the baseline profiles for all modules are built at the edge, the operational phase starts. In this phase, the device can operate as desired and we assume the attacker has full access to the device. The device continuously records and transfers logs and corresponding hashes to the edge-server according to the configuration values of *interval* and *windowsize*.

The device constantly records logs for each module every *interval* seconds and transfers them to the edge every *windowsize* seconds. Overall, every transfer contains *windowsize/interval* number of PWM, PBM and SBM logs. We discuss about the choice of values of *windowsize* and *interval* and how it impacts the performance of `E-Spion` in Section 2.3.

Anomaly Detection Stage: For each of the logs received, the hash-chain-verifier at the network edge first checks the integrity of the logs. If the logs fail the integrity check or no logs are received, then the device is considered compromised or malfunctioning and the IDS raises an alert. We can find out of the approximate time of infection and the likely source of compromise by analyzing the received logs. If the integrity check goes through, the logs are forwarded to the anomaly detection engine at the edge-server. Here, the logs are compared with the baseline profiles for each of the three modules and, in case an anomaly is detected, an alert is raised. The PWM module simply compares the current running process list to the PWM whitelist, while the PBM and SBM modules classify the current device logs using the binary classifiers trained in the learning stage.

2.2.4 Hash Chain Verifier

This component verifies the integrity of the logs received by the edge-server. It maintains hash chains of the logs as shown in Figure 2.2. For the window of time *windowsize* as defined

before, the client generates a hash chain of the device logs during this window of time. We use the SHA256sum utility [37] to compute the SHA-256 one-way hashes of the logs. The device logs include the running process lists, process behavior logs, and system call behavior logs collected for the PWM, PBM and SBM, respectively. A new hash chain link is added after every interval specified by *interval*. So, each window contains a hash chain made of *windowsize/interval* hash links. We use the Merkle-Damgard construction [38] to compute these hash links. So, at the start of every hash chain initiation (every *windowsize* seconds) the server sends an encrypted random nonce c to the device.

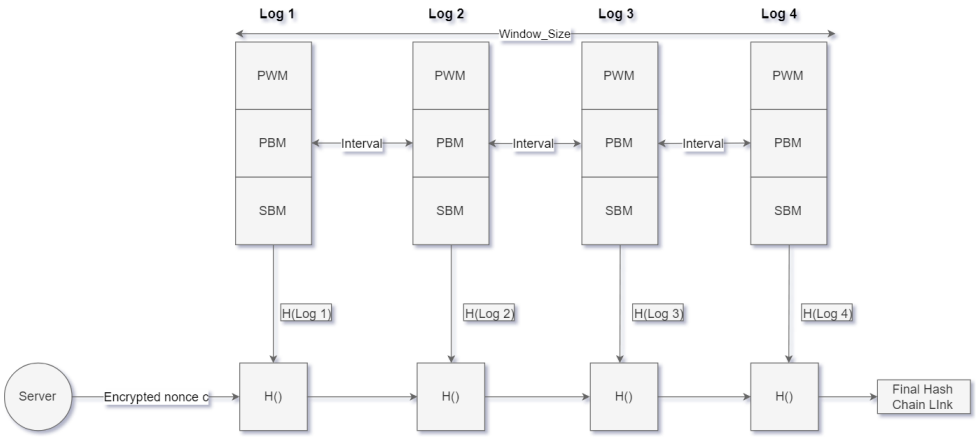


Figure 2.2. Hash Chain Verifier

The hash list uses c as an initialization vector for the hash chain. This value is deleted as soon as the first hash is created. The hash value and the log are stored. After *interval* seconds, the second link of the hash chain is computed using the hash of the previous log and the hash of the current device log. After which the hash of the previous log is deleted from the system. This is done every *interval* seconds for the duration of the window. After the window closes (*windowsize* seconds), the final hash value along with the corresponding files (device logs) in the *windowsize* are sent to the edge-server. The edge-server then computes the final hash from the received logs (and random nonce c) and compares it with the hash value received from the device. If the values do not match, the authentication fails. We do not go into details about the security of the hash chain verifier, and refer the reader to our previous paper for details on the threat model and security analysis [34].

2.3 Implementation Details

In this section, we discuss key challenges in the implementation of the prototype of our system and how we have addressed these challenges, namely: deployability, choice of classifiers, timing/interval choices, modularity and distributed nature of logs.

Deployability. Due to the heterogeneity and resource constrained nature of IoT devices, deployment of a system level IDS like ours is a challenge, and as with all host based IDSs, it is the limiting factor. We implement our system with the goal to overcome this challenge and make sure that deployment of the system is feasible for all IoT devices. We observed that 71.3% of all IoT devices run some version of Linux as their operating system and "Linux is becoming the standard OS for all gateway and resource constrained devices" according to the 2017 IoT developer survey [39]. So in order to maximize deployment, we build our client side (SysMon) modules using common system tools like atop, ps, sha256sum, openssh and strace present on all standard Linux distributions. We tested our system on various CPU architectures and Linux operating systems along with several IoT device emulations using Firmadyne [36] in order to make the device modules scalable and easy to deploy. It must be noted that similar tools are available on other OSes and `E-Spion` variants for them can be implemented along similar lines.

Choice of Classifiers. The common approach for detecting anomalies involves using unary or one class classifiers. However, IoT devices are prone to malfunctions and fluctuations in performance [40]. This results in one class classifiers classifying benign malfunctions and fluctuations as malicious activities, thus causing high numbers of false positives. To avoid this problem, we use binary classifiers and train our classifiers using existing IoT malware samples.

For both the PBM and SBM module, we tested different binary classifiers like naive Bayes, decision tree, logistic regression, random forest, K nearest neighbor (K-NN) and multi-layer perceptron (feed forward artificial neural network (ANN)) classifiers. We found that the random forest classifier works best for detecting intrusions for our dataset as shown in Tables 2.4 and 2.5. Also, tree based classifiers are the fastest and most efficient to build. So, we use the random forest classifiers for both our PBM and SBM modules.

Table 2.4. Performance evaluation for various PBM module classifiers

Classifier	Accuracy (%)	ROC Area	True Positive Rate	False Positive Rate	Precision	Recall	F1-Measure
Naive Bayes	26.24	0.82	0.26	0.15	0.84	0.26	0.23
Logistic Regression	85.72	0.84	0.85	0.66	0.83	0.85	0.82
Decision Tree	97.64	0.94	0.97	0.12	0.97	0.97	0.97
Random Forest	97.75	0.98	0.97	0.10	0.97	0.97	0.97
K-NN	97.66	0.97	0.97	0.10	0.97	0.97	0.97
ANN	95.26	0.89	0.95	0.24	0.95	0.95	0.94

Table 2.5. Performance evaluation for various SBM module classifiers

Classifier	Accuracy (%)	ROC Area	True Positive Rate	False Positive Rate	Precision	Recall	F1-Measure
Naive Bayes	100.00	1	1	0.0	1	1	1
Logistic Regression	99.74	0.99	0.99	0.002	0.99	0.99	0.99
Decision Tree	99.94	0.99	0.99	0.001	0.99	0.99	0.99
Random Forest	100.00	1	1	0.0	1	1	1
K-NN	99.98	1	1	0.0	1	1	1
ANN	99.94	1	0.99	0.001	0.99	0.99	0.99

The low accuracy($\sim$26%) of the naive Bayes classifier in the PBM module can be attributed to its class independence property. From our observation, IoT malware usually employs a combination of CPU, memory and disk resources; thus the usages of these resources are correlated to each other. Therefore, assuming that these features are independent of each other leads to a lower accuracy for the PBM module. However, for the SBM module, the naive Bayes classifier works perfectly. The reason is that the system calls made by a process are not directly correlated to other system calls made by the device, so the class independence property is a fair assumption in the case of the SBM module. We find that all the binary classifiers give high detection rates for the SBM module. This can be attributed to the larger number of good features (that is, 136) used for training our SBM classifier.

The regression classifier works better than the naive Bayes classifier with an accuracy of $\sim$85% for the PBM module. A logistic regression model searches for a single linear decision boundary in the feature space. Hence, the lower accuracy can be attributed to the fact that our data does not have a completely linear boundary for decisions. Therefore, the ideal decision boundary for our dataset would be non-linear. We observe that tree based classifiers (decision trees and random forests) work best for our system. However, decision trees are prone to over-fitting which is the reason we choose a random forest classifier over a decision tree classifier. Although the distance based K-NN classifier and the deep learning based ANN classifer give comparable results to tree based classifiers, they take more time

and computational power to build which is unsuitable for the real-time requirements of IoT networks.

Timing/Interval choices. The configuration values for the variables *windowsize* and *interval* play a significant role in the detection efficiency and overhead costs of our system. We performed several tests with different configuration values on different devices in our testbed (see Subsection 2.4.2). We tested our system with *windowsize* of 20, 50, 100, 500 and 1000 seconds and *interval* of 2, 10 and 20 seconds. We found that for all devices, the detection accuracy remains nearly constant (75%, 97%, 99% for PWM, PBM, SBM respectively) for all these configurations. This can be attributed to the fact that the devices behave similarly in all these intervals and the choice of the recording intervals has minimal impact on the detection accuracy for our current dataset. However, the larger the *interval* size, the higher chance the attacker has of evading the system as discussed in [34]. Also, our system can only detect attacks after *windowsize* seconds when logs are transferred to the edge-server. If the *windowsize* is too high, then the detection time of the attack will also be higher. Large scale time-critical networks, like vehicular networks, smart-grids etc., require very short attack detection time because of their real-time safety requirements. So in such networks, the *windowsize* should be small enough to detect these attacks in real-time. In terms of overhead costs, a lower *windowsize* results in higher communication overhead at the device as the logs need to be transferred more frequently. A lower *interval* results in a higher computational overhead at the device mainly because more hashing operations are required. Such computational and communication overheads can be detrimental in scenarios where resource constrained embedded devices are present in the network. So, we leave the choice of the optimal values of these configuration parameters to the system administrator as it depends on the system requirements, devices used, deployment scenario and associated risk for each device. In our current prototype, we use a *windowsize* of 100 seconds and *interval* of 10 seconds for all the devices.

Modularity of Logs. Our system has three detection modules that provide varying degrees of detection efficiency and overhead costs. We deploy these modules independently in separate threads rather than sequentially. The outputs from these modules are combined by the module manager to provide a 3-layered detection output rather than a simple alert.

This implementation is essential to weed out false positives and differentiate between benign malfunctions and attacks. For example, in case of a device malfunction, the PBM module will raise an alert due to the anomalous process behavior. However, the SBM module will not raise an alert as no anomalous system calls were issued by the process. These modular logs provide the module manager enough information to classify this incident as a malfunction rather than an attack and thus reduce false positives.

The modules have different overhead costs associated with them as well. The modular design allows activating/deactivating each module as required by the device according to its resources, associated risk, network conditions etc. For example, a resource constrained device can be configured to activate the expensive SBM module only when an alert is received from either PWM/PBM modules. On the other hand, a device with a high associated risk will be configured to have all modules active at all times. The modular design enhances flexibility and scalability in the deployment of our system.

Distributed Nature of Logs. In our current prototype, we store the learnt behavior from the learning stage for each device on the edge and we assume that the device functions benignly until the end of the learning phase. This assumption holds in our current threat model but would be a limitation for real world scenarios where devices are compromised as soon as they connect to the network or are compromised in production. To address this challenge, we added some additional functionality in our current prototype to move further towards a fog computing paradigm [4]. The edge-server stores the logs in a cloud repository accessible by other edge-servers. This repository holds learnt logs from devices across different `E-Spion` enabled networks. Such logs enable one to compare behaviors of the same devices in different networks and to detect anomalous behavior during the learning stage. A device which has already been compromised during the learning stage shows significantly different behavior when compared to the same device in another network which has not been compromised during the learning stage. This distributed nature of logs allows inter-operation between different `E-Spion` edge-servers and a fail-check in case of devices behaving maliciously during the learning phase.

2.4 Evaluation

We perform an extensive evaluation of `E-Spion` on a typical enterprise IoT setup with 3973 of the most recent IoT malware samples. In this section, we provide details on our malware dataset and evaluation testbed. We then discuss the detection efficiency of our system and finally provide an analysis of the malware samples in terms of our device logs. After which we discuss the effectiveness of our system against 8 types of file-less attacks and finally look at the overhead costs associated with our system. We focus on collecting IoT malware extensively rather than simulating various network based attacks as seen in prior work. The reason for this is that the goal of our host-based system is detecting the compromised host/device during the injection or infection stage rather than the attack stage. More details on our threat model are given in our previous paper [34].

2.4.1 IoT Malware

IoT malware is downloaded during the infection stage according to the device operating system and architecture. For evaluating our system, we collected different variants of IoT malware and built a comprehensive dataset using 3973 malware samples from the most popular malware families: Zorro, Gayfgt, Mirai, Hajime, IoTReaper, Bashlite, nttpd, linux.wifatch etc. The malware samples were collected from IoTPOT [41], VirusTotal [42], and Open Malware [43]. These malware executables are compiled for different CPU architectures and endianess. The collected malware executables are classified according to different device architectures in Table 2.6. 2572 of the samples are compiled for little endian processors while 1421 of them are for big endian processors.

Figure 2.3. Experiment Testbed

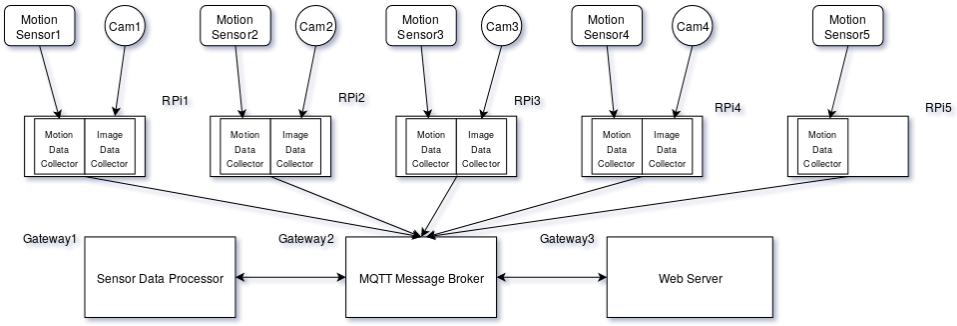


Figure 2.4. Testbed Implementation Details

We have used 20% (795) of our malware samples as training malware samples for the learning stage and 80%(3178) of them as testing data for evaluation in our experimental setup.

Table 2.6. Malware Executables breakdown according to CPU architecture

Architecture	Number of Malware samples
Mips	935
Arm	912
x86	576
Sparc	299
Renesas/SuperH	310
x86_x64	294
MC68000	294
PowerPC	353
Other	26

2.4.2 Evaluation Testbed

We create a typical motion sensing network using 4 webcams, 5 raspberry pi devices (4 mounted and 1 tethered), 3 HPE GL10 IoT gateways and 1 Aruba PoE Switch (see Figure 6.4).

The raspberry pi devices are responsible for recording images from the cameras and movement from the sensors. The motion sensor is an accelerometer installed on the raspberry pi device and is responsible for detecting any movement of the raspberry pies. The devices

communicate with each other using MQTT (Message Queuing Telemetry Transport). A high level outline of the functions of the devices in the testbed can be seen in Figure 2.4.

2.4.3 Detection Efficiency and Analysis

To test our system, we manually download and run 3178 malware executables on the devices in our testbed. We evaluate the performance of our system by running the malware samples sequentially. Every time a malware sample is executed, we check if our system is able to correctly flag each malicious process spawned by the malware. After every run, we restore the system with a clean OS and execute the next malware sample. In what follows we discuss the detection accuracy for each layer of our system.

PWM layer. The PWM layer had a detection rate of 79.09%. We see that the simplistic PWM is able to detect most of the IoT malware samples just through the simple white-listing of process names. This reinforces our claim that most of the IoT malware is basic and does not employ any obfuscation or deception techniques. On average, each malware spawns a mean of 1.79, median of 1 and mode of 2 new processes. 20.91% of the malware spawn no new processes but rather manipulate or masquerade as a benign process (e.g., white-listed process). We observe that most malware invokes the prctl system call and uses the PR_SET_NAME request. We also observed that some malware simply change the name of the malicious program to a benign one. Most of the malware masquerades as common system utilities such as sshd, telnetd etc. Another observation we made from the PWM layer is that some of the malware samples produce a randomly generated process name for each execution of the sample. This would seem an appropriate approach taken by attackers to bypass process name blacklisting approaches.

Another key point to note is that we do not see any false positives from the PWM layer. This can be attributed to the fact that all the processes spawned by benign applications are already whitelisted during the learning stage and none of the benign applications spawn any new unwhitelisted processes during the operation stage.

PBM Layer. The PBM layer has a high detection rate of 97.02% but at the same time a significant false positive rate of 2.97%. As we are monitoring for anomalous activity for all

the running processes of the system, the PBM layer is able to capture malware masquerading as benign processes which the PWM layer is unable to detect. As this is a machine learning predictive approach, we do encounter false positives in this layer and some benign processes are incorrectly classified as malicious. We observe that most malware is very aggressive in terms of CPU and memory usage when it infects the system. Therefore they can be easily detected by the PBM as this module is able to quickly detect anomalous behavior even for malware masquerading as benign processes.

To demonstrate the effectiveness of each metric/feature in our PBM module, we compare baseline logs against the malicious logs over time according to average CPU usage (syscpu, usrcpu), average memory sage (Vgrow, Rgrow) and average disk usage (wrdsk, rddsk) in Figure 5. The malicious logs are represented in red while the benign logs are represented in blue. We observe that the malicious logs are clearly distinguishable from the baseline logs using just a subset of the metrics.This means that these features individually (CPU, memory and disk usage) also perform well in terms of detection efficiency. So, in devices where one of the metrics is inaccessible or unavailable (for example, small embedded devices which have no disk but just flash memory), the PBM module will still be able to detect intrusions effectively. We observe that most of the malware has a typical *bursty* behavior pattern in that it remains dormant most of the time and performs its malicious activities in a burst. Benign applications on the other hand have a *consistent* behavior where their operations are periodic and constant. These results confirm our original intuition about the periodic and consistent nature of IoT device processes. We also train our PBM classifier using the CPU, memory and disk usage metrics individually and observe detection accuracy of 94.2%, 96.67%, and 91.46%, respectively. Although using the metrics individually has high detection efficiency, it also results in higher false positive rates of 5.77%, 3.32%, and 8.5% respectively. So, we use all the metrics in conjunction to minimize the number of false positives.

SBM Layer. The SBM layer has a detection rate of 100% and 0 false positives. This is the most fine grained detection module. However, it is also the most expensive. Table 2.7 shows a comparison between malicious and benign SBM logs according to the different system call types in terms of average number of calls made and average % of execution time

Figure 2.5. Comparison between malicious vs baseline PBM log samples over time according to (a) CPU usage(usrcpu, syscpu), (b) Memory Usage(vgrow, rgrow) and (c) Disk Usage (wrdsk, rddsk)

taken by the system call. We can see that malicious processes use a typical combination of system calls, like connect, socket, read, write, munmap, ioctl etc., and the behavior is highly different from benign processes.

2.4.4 Effectiveness against File-less Attacks

As seen before, we have categorized file-less attacks into 8 categories. We manually performed all 8 types of attacks in our `E-Spion` monitored environment. We observe that our system is able to effectively detect all 8 types of the attacks in our evaluation testbed. The reason being that all file-less attacks propagate using either shell or system commands like passwd, rm, kill etc. on the device using root privileges. Whenever a command is issued in Unix/Linux, it creates/starts a new process. This new process is flagged by the PWM layer. Only the Type 8 attack manages to evade the PWM layer when it it uses the ssh utility to set up the port forwarding. This however is flagged by the PBM layer as it detects an anomalous use of the whitelisted ssh processes. The SBM layer also detects the anomalous use of socket, bind system calls made by the SSH processes and flags the Type 8 attacks. Even more sophisticated file-less attacks which employ benign application based commands for infection would be detected by our PBM and SBM modules as the application's "behavior" is "anomalous" or different from the learnt baselines.

2.4.5 Overhead Analysis

We now discuss the performance and storage overheads for the different modules of `E-Spion`. An important requirement of our design is that the system should impose minimal costs on the already resource constrained IoT devices. As we see in Table 2.8, most of the computational resources (CPU and memory usage) are required by the server side modules which run on the more powerful edge-server while the client side modules running on the IoT device require minimal computational resources. Table 2.8 also shows the overheads in terms of CPU, memory and disk usage required by the different modules on the device. We observe that the SBM module is the most computationally expensive while the PWM is the least expensive. The PWM module requires the least amount of CPU, memory and disk on the device. In terms of CPU usage and disk usage, the SBM module is the most expensive. It is important to note that the SBM also slows down benign processes because it uses the system tool *strace*. *strace* pauses the process twice for each syscall, and executes context-switches each time between the process and *strace*.

Table 2.7. Comparison of malicious and baseline logs of the SBM module

System Call Type	Malicious Process: Avg No of calls	Benign Process: Avg No of calls	Malicious Process: Avg Time%	Benign Process: Avg %Time
connect	10	1	29.375431	0.109712824
_newselect	6	0	7.652097	0
close	882	527	0.79684114	2.8245006
nanosleep	5	409	3.5325627	44.9521
fcntl64	53	34	0.07268107	0.3070208
socket	7	1	1.5146441	0.06425141
ra_sigprocmask	0	2	0	0.009006577
getsockopt	6	0	0.77036035	0
read	2868	984	2.2888484	7.1284814
open	22	1704	0.16306427	13.151322
execve	1	1	3.61E-05	0.0042437743
chdir	1	0	7.06E-04	0
access	11	79	0.066558525	0.848794
brk	3	33	0.005014777	0.19349128
ioctl	3464	498	0.52835435	12.714709
setsid	1	0	0.008818369	0
munmap	2769	18	13.151226	0.5787933
wait4	1	0	12.624713	0.3850142
clone	1	2	0.018954737	0.024077073
uname	1	0	0.0037215038	0.0037744138
mprotect	13	232	0.07808309	1.8957471
prctl	1	0	0.022474075	0
rt_sigaction	12	85	0.20787959	0.2348594
ugetrlimit	1	1	5.62E-04	0.003911133
mmap2	2801	291	12.589997	3.5121188
fstat64	24	656	0.026696749	2.4766097
getuid32	1	3	0.008650763	0.044761505
getgid32	1	1	0.007634516	0.026103668
geteuid32	1	2	0.0027905148	0.028454894
getegid32	1	0	0.0021606325	0.015322265
madvise	1	1	2.76E-04	0.19356513
gettid	1	0	6.24E-04	0
set_thread_area	1	0	3.96E-04	0
set_tid_address	1	0	4.08E-04	0.0020117187

Table 2.8. Overhead Analysis of the modules of `E-Spion`

	Device			Edge-Server Side	
Module	CPU Usage(%)	Memory Usage(Kb)	Disk Usage(Kb)	CPU Usage(%)	Memory Usage(Kb)
PWM	0.01	2896	67	23	74,684
PBM	0.2	4788	776	24	739,936
SBM	0.6	3912	1340	25	363,680

Table 2.9. Comparison of different modules of `E-Spion`

Module	Accuracy	Computational Cost on Device	Storage Cost on Device	Slow-Down of benign applications	Computational Cost on edge-server	False Positives
PWM	Moderate	Low	Low	No	Low	None
PBM	High	High	High	No	High	Moderate
SBM	Highest	High	High	Yes	Moderate	None

On the edge-server side, the PBM module is the most expensive in terms of memory and PWM is the least expensive. The PBM module requires the most amount of memory for extracting features because the process behaviour logs are denser than both process whitelists (PWM) and system call behaviour logs (SBM). As the SBM module only checks for certain system calls and records a summary of these calls in its logs, extraction of features from the logs is considerably less expensive than PBM. Both SBM and PBM modules require training and operating expensive random forest binary classifiers due to which they are 5x and 10x more expensive respectively than the PWM module in terms of memory usage. In terms of CPU usage, all 3 modules have similar overhead costs on the server side.

We see that the three modules have varying degrees of computational and storage overheads and detection efficiency. Finally in Table 2.9, we summarize our comparisons of each module in terms of accuracy and overhead.

2.5 Related Work

Intrusion Detection for IoT. IDSes for IoT devices use different strategies for placing intrusion detection tools, namely: centralized, distributed, and hybrid. IDSes like [26], [44], [45], use the centralized IDS placement approach and generally monitor traffic passing through the border routers. Such a strategy has the advantage of detecting attacks from the Internet into or out of the IoT network but this is not enough to detect attacks involving just the nodes of the IoT network. Also, if part of the network is disrupted, a centralized IDS

might not be able to monitor all the nodes. Light-weight distributed placement strategies have been proposed in [46]–[48] where each node is responsible for monitoring and analyzing their packet payloads, energy consumption and their neighbors respectively. However, these strategies impose a non-negligible computation overhead on the already resource constrained devices. Most recent IDSes [24], [49], [50] are hybrid approaches which combine centralized and distributed approaches. The Kalis IDS [25] has been designed with a flexible placement strategy, in that it can be placed on the devices, or at some gateway, or onto its own specialized device. Our system also uses a hybrid placement strategy where the data gathering module runs on the device while the analysis module runs on the edge-server.

Most existing IDSes for IoT devices and embedded devices, like [46], [51], [52], use signature-based detection schemes. These IDSes rely on network information gathered by a packet sniffer, and detect attacks using signature matching over this information. The approaches that only use signature-based detection are simpler to develop but cannot detect attacks for which the signature is unavailable. Also, it is difficult for resource constrained IoT devices to run computationally expensive signature storing and matching schemes. Another factor is that with the high heterogeneity in terms of protocols, functionality, manufacturers, device architectures etc., the attack signatures/rule list becomes very large and complicated. While running through a large signature list is sustainable for a traditional network, small IoT networks incur heavy overhead and this results in poor performance of the IDS. Also, going through a large signature list usually results in a higher number of false positives. Conversely, anomaly-based techniques are more versatile, as they can detect unknown attacks, but are harder to implement and more inaccurate, potentially yielding high false positive rates. A number of such anomaly based schemes detection schemes have been proposed [26], [44], [47], [49], [50] which rely on detecting anomalies by inspecting packet rates, sizes, payloads, headers, node connections, energy consumption, device profiles etc. Our system is also an anomaly based detection scheme but focuses on building device profiles using system information gained from the running processes and system calls rather than network information.

This section extends our previous work [34] by introducing modifications to the IDS implementation and reporting evaluation results concerning the effectiveness of the IDS

against recent sophisticated file-less attacks. Overall, our approach is significantly different from all the previous approaches in the area of IDS for IoT systems as we aim to build a hybrid light-weight IDS system which is able to detect anomalous behavior in terms of system level information from running processes and system calls.

Traditional system level techniques. Even though there is little research on system level IDS technology for IoT, there has been significant research in the area of intrusion detection using system events and provenance logging for traditional computer systems. There are two main approaches. The first approach is based on system event logging and then causally connecting these events during attack investigations to build causal graphs like Auditd [53] in Linux kernels which maintains audit logs of important system events. The other approach is using provenance propagation like in PASS [54] which stores and maintains provenance data where provenance is calculated for certain entities like network sessions after which the IDS captures the program dependencies during execution. There have been attempts to build such schemes for distributed systems [55]. ProTracer [56] proposes a light-weight provenance "tainting" scheme which is a hybrid of both those approaches. The most comprehensive of such hybrid IDSes is RAIN [57] which achieves higher run-time efficiency by pruning out unrelated executions in their provenance graphs. Although these approaches have similar ideas compared to ours, they are not suitable for IoT devices. The implicit assumption made in these approaches is that each input event has a causal effect on all the output events. These systems then try to build a model of the behavior of all the benign applications using this assumption. Such an approach works well for complex applications running on traditional computer systems as there are large numbers of running processes and threads interacting with each other on these systems. However, this model would be an overkill for IoT devices. Most IoT devices have much simpler program execution and a simpler approach to modeling IoT devices is required. The other major factor to consider is that these approaches incur large computational and storage costs which are infeasible for resource-constrained IoT devices. By contrast, our system uses a simpler and more efficient 3-layered approach to model IoT device behavior using system data. Also, we leverage network edge-servers and thus are able to minimize the workload on the devices. This makes our approach practical and cost-efficient for IoT devices.

2.6 Summary

In this chapter, we have proposed a system-level IDS `E-Spion` tailored for IoT devices. It builds a 3-layered baseline profile with varying overhead costs for IoT devices using system information and detects intrusions according to anomalous behavior. It is specifically designed for resource-constrained IoT devices with an efficient device-edge split architecture and modular 3-layered design. We extensively tested our system with a comprehensive set of 3973 IoT malware samples and 8 types of file-less attacks. We observed a detection rate of over 78%, 97% and 99% for our 3 layers of detection, respectively.

3. JARVIS: SECURITY CONSTRAINED RL FOR IOT ENVIRONMENTS

In the consumer market, IoT technology is mostly synonymous with products pertaining to the concept of "smart home", covering devices and appliances, such as lighting fixtures, thermostats, home security systems and cameras, that support one or more common ecosystems consisting of sensors of different kinds, such as motion, sound, light, heat, and touch, that can be controlled via devices associated with the ecosystems, such as smartphones.

A widely used approach for providing intelligent IoT services is through apps. These apps can support simple functionality such as "opening the door as an authorized user approaches the building" or more complex functionality like "navigating the car through a certain area autonomously". The apps communicate through API calls from the managing device to the IoT devices (sensors/actuators) through edge or cloud computing. Trigger-action apps platforms, like IFTTT [58], Zapier [59], and Apiant [60], allow 3rd parties to develop apps for these platforms. Such a strategy promotes the creation of communities of developers that build apps catering to different environments, devices, and protocols.

The development of communication protocols for Internet of Things (IoT) devices, such as 6LowPAN, CoAp, and Zigbee, and the progresses in AI have made it possible to interconnect different IoT devices and achieve smart autonomous IoT systems. However, smart IoT-based systems require interconnection and inter-operation among devices, apps, users and the edge. Such a complex environment of IoT devices and apps dynamically interacting with each other is prone to security/safety issues [1]–[3]. Another issue is that, in terms of functionality, each app has specific individual goals (e.g, "turn on the heater if the temperature is below the user requirement"). However, when selecting the actions to be executed, the app does not take into consideration the global view of the environment where the IoT device is deployed. Such a lack of awareness can lead to decisions and actions that are not globally optimal in terms of user requirements or goals. Evidently, there is a need for intelligent monitoring systems to ensure safety and maximize functionality with a global view of all devices and apps and their interactions.

To address such drawbacks, in this chapter we introduce `Jarvis`, a novel constrained RL framework for autonomously predicting 'optimal' and 'safe' decisions in an IoT ecosystem. An agent explores a simulated RL environment in order to find the optimal device actions according to the user's goals, such as saving energy and minimizing cost. Examples include turning off appliances to save energy, using devices during the off-peak hours to minimize electricity costs, etc. At the same time, the exploration of the agent is constrained by security and safety policies identified for the environment. For example, actions such as unlocking doors when user is not home or sleeping, turning off heater during winters, etc. are not permitted to the agent.

We design the framework to be context-independent and applicable to any IoT environment. By observing the specific IoT environment, `Jarvis` dynamically builds a simulated environment in terms of devices states and actions. An agent, constrained by security policies, can traverse the simulated environment in multiple episodes of specific time periods and find the optimal safe actions in terms of functionality requirements provided by the user. A Deep Q learning network (DQN) is used to determine the highest rewarding (quality) actions for each environment state and *time instance*. A Q learning approach is ideal for such an environment/ecosystem where, for each state-action pair, we can determine its quality through a cumulative reward for the time period in terms of the user goals. We train the agent using a deep neural network (DNN) to maximize the cumulative reward and thus generate the optimal quality function.

One of the main challenges in the design of our framework is that safety and security policies have to be specified by users and vary across different environments, deployed devices, and apps. Therefore, full manual specification of policies is not a viable approach. In order to minimize human effort in defining safety and security policies, `Jarvis` is designed to identify benign trigger-action (T/A) behavior in IoT environments by observing events occurring naturally in the environment or 'how the environment would function without machine intervention'. We define trigger-action (T/A) behavior based on enterprise platforms [58] and formal models [61] expressed in terms of device states and actions. Our assumption for safety and security is that such natural behavior is safe. Any unnatural or anomalous behavior is considered unsafe or malicious. However, in practice benign device malfunctions and human

errors are common. So, an artificial neural network (ANN) is trained to filter these benign anomalies using back propagation. `Jarvis` observes the state transitions of the environment during a specified learning phase and dynamically builds the white-list of "safe policies" or safe T/A behavior using the ANN. After the learning phase is completed, the RL agent uses the white-list to constrain the RL environment and thus prevent unsafe/insecure state transitions. We use the terms ANN and DNN to refer to neural networks, with a single hidden layer trained by back propagation and multiple hidden layers trained by reinforcement learning, respectively. Formally, `Jarvis` can be defined as a model-based RL framework based on the Dyna-Q framework [16], where: (i) the model of the environment in terms of safety/security policies is learnt from actual user experiences through supervised learning, and (ii) the optimal policies for the learnt model are learnt by the RL agent from simulated experiences.

To summarize, we make the following contributions:

1. A context independent RL framework `Jarvis` for IoT environments.
2. A novel approach to constrain RL using safety and security policies.
3. An approach for dynamically learning safety/security policies in terms of triggers and corresponding actions in IoT environments.
4. An instantiation of `Jarvis` for a smart home environment and an extensive evaluation of the security and safety of the system using simulated data from manually crafted safety violations collected from prior work [1], [2] and user defined anomalous activities from the SIMADL project [62].
5. An analysis of the functionality benefits of the system using real world data in terms of three functional requirements: energy conservation, cost minimization, and temperature optimization.

The rest of this chapter is organized as follows. In Section 5.1, we briefly review the general IoT architecture and the Deep Q Learning framework. Next, we formally define our system model and challenges, and formulate the functionality optimization goal as a Markov Decision problem (MDP) in Section 5.4. In Section 3.3, we define our deep RL based Q learning framework to solve the problem optimally. Next in Section 3.4, we instantiate `Jarvis` for a smart home environment, provide design details, and analyze the benefits of

`Jarvis` in a small smart home example. We quantitatively evaluate the instantiation in terms of safety and functionality in Section 3.5. Finally, we discuss related work in Section 5.6 and outline a few conclusions in Section 5.7.

3.1 Background

3.1.1 IoT Architecture

All popular IoT platforms, like Samsung Smartthings, Apple HomeKit, and OpenHAB, rely on the concept of separation of intelligence from devices. Smart applications can then be built by combining the intelligence layer with the functions and data provided by the devices. At a higher level, an IoT architecture can be considered as organized into four components: devices (sensors, actuators, appliances, etc.), edge (hub, router, connecting devices), cloud (cloud services, databases, analytics etc.), and control devices (mobile phones, desktops etc.). All IoT devices are standardized to provide certain 'capabilities' which allow devices to alter certain 'attributes' through 'commands'. Actuators like smart locks, lights, appliances have capabilities like lock/unlock, power on/off, increase temperature etc. Sensors for motion, sound, heat etc. have capabilities such as motion detected/no motion, high pitch sound/no sound/noise, optimal/high/low temperature etc.

These components inter-operate based on an event publish-subscribe architecture. The devices interact with the edge through device specific handlers which parse device specific messages (open/close, on/off, heat/cool, brew/do not brew) to/from the device and relay normalized edge-readable events (door opened, device turned on, heating temperature reached value x, etc.) to the edge. All publications of an event can be seen by the apps that have subscribed to that particular event.

3.1.2 Deep Q Learning

A Reinforcement learning (RL) framework [63] (see Figure 5.1) is an environment where for every state transition or state-action pair (s, a), a reward function $R(s, a)$ determines a reward value r. A RL agent traverses the environment according to a policy $\pi_\theta(s, a)$ for a time period θ and receives a total reward value accrued over all the state transitions involved

for a given model of state transition probabilities. In a Q learning framework, the goal is to find the optimal policy which maximizes the total reward through exploration in such a RL environment using a Q function which determines the quality or cumulative reward for all state-action pairs.

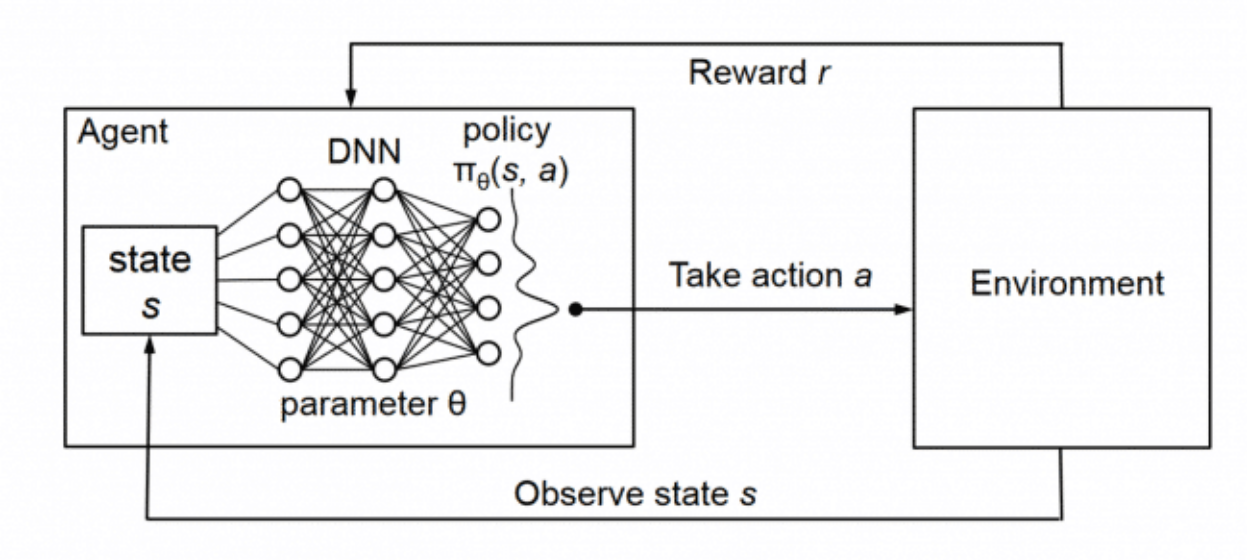


Figure 3.1. Deep Q Learning Environment

In a deep Q learning system, a Deep Neural Network (DNN) is used to determine the optimal Q function using a temporal difference equation defined as follows:

$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a) + \gamma Max_a\{Q_t(s,a)\} - Q_{t-1}(s,a)]$

where $Q_t(s,a)$ is the current Q function and $Q_{t-1}(s,a)$ is the previous Q function for the environment. The estimated next state and action are denoted by s and a, respectively. The learning rate (α) determines to what extent newly acquired information overrides old information. The discount factor (γ) determines the importance of future rewards.

3.2 System Model and Problem Formulation

In this section, we first discuss modelling an IoT environment as a finite state model (FSM). After which, we define the state transitions of the environment as "episodes" and formulate the functionality optimization goal of `Jarvis` as a MDP. Then, we discuss the challenges associated with applying such a model to cyber-physical systems.

3.2.1 IoT Environment

The IoT environment FSM consists of k devices $\{D_1, D_2, .., D_k\}$, η users $\{U_0, U_1, ..., U_\eta\}$, and m apps $\{ap_0, ap_1, ..., ap_m\}$. By convention, manual operations in the model are denoted by a pseudo app ap_0. Each device in the environment is modelled by a set of device-states and a set of device-actions. At any point of time, device D_{i} can be in one of a set with a number i_{ss} of device-states: $\{p_{\mathrm{i}_0}, p_{\mathrm{i}_1}, ...p_{\mathrm{i}_{ss}}\}$. At a *time instance* t, a device-action can be executed on device D_{i} from a set with a number i_{as} of device-actions: $a^t_{\mathrm{i}} \in \{a_{\mathrm{i}_0}, a_{\mathrm{i}_1}, , a_{\mathrm{i}_{as}}\}$. Specifically for IoT platforms, device capabilities and device attributes can be translated to device actions and device-states respectively.

Each device D_{i} has a transition function δ_{i} which is the link between a device-action and a device-state. For a device D_{i} in state p_{i_x} and having device action a_{i_y} take place on it, $\delta_{\mathrm{i}}(p_{\mathrm{i}_x}, a_{\mathrm{i}_y}) = p_{\mathrm{i}_x}$ gives the new state of the device. Along with this, each device D_{i} has a dis-utility function $\omega_{\mathrm{i}}(p_{\mathrm{i}_x}, a_{\mathrm{i}_y})$ which represents the dis-utility per *time instance* that results if the execution of device-action a_{i_y} is delayed in state p_{i_x}. A device can exist in different locations and thus have varying contexts in terms of accessibility, user permissions etc. To model this in our framework, we follow the container based approach followed by most IoT platforms. Each container acts as a boundary between the devices; the containers are organized hierarchically according to: user accounts, locations and groups. Therefore, device D_{i} can only be accessed by a set of authorized users u_{i}, $u_{\mathrm{i}} \subseteq \{U_0, U_1, ..., U_\eta\}$, depending on its location l_{i} and its group g_{i}, and corresponding device and app subscription policies.

3.2.2 State Transition Model

For the overall environment state S_t, at the next *time instance* $t+1$, a set of authorized users $U^t \subseteq \{U_0, U_1, ..., U_\eta\}$ can use a set of apps $AP^t \subseteq \{ap_0, ap_1, ..., ap_m\}$ to perform an *action* A_t on a set of devices $D \subseteq \{D_1, D_2, .., D_k\}$, to get the new state of the environment S_{t+1}. So the state transition of the environment is represented as the current state $S_t = (s_0, s_1, ., s_{\mathrm{i}}, ., s_k)$ plus a set of at most k (one per each device) device-actions. $A_t = \{a^t_0, a^t_1, ..., a^t_k\}$ is the set of actions taken at *time instance* t by a set of users U^t through a set of apps AP^t resulting in the next state S_{t+1} at *time instance* $t+1$. The next state

is computed using the transition function for each device and corresponding action on the device such that $S_{t+1} = (\delta_0(s_0, a_0^t), \delta_1(s_1, a_1^t), ..., \delta_k(s_k, a_k^t) = \Delta(S_t, A_t)$ where Δ is the overall transition function of the environment.

Definition 3.2.1. *A FSM consists of tuple* (SS, AS, Δ) *where:* $SS = \widetilde{\bigcup}_{\mathrm{i}=0}^{\nu} S_{\mathrm{i}}$ *is the state space with* $\nu = \prod_{\mathrm{i}=0}^{k} \mathrm{i}_{ss}$*;* $AS = \widetilde{\bigcup}_{\mathrm{i}=0}^{\upsilon} A_{\mathrm{i}}$ *is the action space with* $\upsilon = \prod_{\mathrm{i}=0}^{k} \mathrm{i}_{as}$*; and* Δ *is the overall state transition function. The overall state of the FSM at time instance t is defined as* $S_t = (s_0, s_1, ., s_{\mathrm{i}}, ., s_k)$*, where* s_{i} *is the state of the i-th device such that* $s_{\mathrm{i}} \in \{p_{\mathrm{i}_0}, p_{\mathrm{i}_1}, ...p_{\mathrm{i}_{ss}}\}$.

We monitor state transitions in the IoT environment in terms of "episodes". We define two configuration parameters for an episode: time period T and interval I. The state transitions occur every I seconds until the timestamp reaches T seconds, after which the state is reset to the initial state and marks the end of an episode. An episode basically consists of T/I *time instances* at which the state transitions of the environment are recorded. For example, for $\{T, I\} = \{60, 1\}$ minutes,the episodes are an hour long with state transitions every minute.

The following constraints apply to all state transitions:

1. Only one action per device per interval is allowed.
2. Only authorized users U^t at *time instance t* can access the app according to app subscription policies.
3. Only authorized app AP^t at *time instance t* for the device can take actions on the device according to device subscription policies.
4. Only one app can take action on the device per interval. Conflicts are resolved on a first come first serve basis.
5. In a single interval, each device can only change its state at most one time.

We following definition defines our model of the IoT environment state transitions in terms of episodes.

Definition 3.2.2. *An episode is defined as a tuple* (N, S_0, T, I)*.* $N = \{S_0, S_1, ., S_t, ..S_n\}$ *is an ordered list of states reached in the episode where each next state* $S_{t+1} = \Delta(S_t, A_t)$ *for an action* A_t *and* $0 < t \leq n$*;* $n = \lceil T/I \rceil$*;* S_0 *is the initial state of the episode;* T *is the time period;* I *is the interval.*

3.2.3 Problem Definition

We formulate the functionality optimization goal of `Jarvis` as a MDP. A MDP is a sequential decision making problem where outcomes are under the control of an agent. The agent's goal in this case is to maximize functionality as specified by the user by choosing a sequence of actions for the upcoming episode of the environment. In our model, functionality requirements defined by the user are measured through a reward function. The functionality requirement specified determines the utility ($F()$) gained by the user which is one part of the reward function in the environment. The other part derives from the dis-utility ($D()$) caused to the user in terms of delays, waiting time, and discomfort. The general structure of the reward function is defined as follows: $R(S, S, t) = F(S, S, t) - D(S, S, t)$ where S is the current state, S is the next state of the environment, and t is the current *time instance* of the episode. The goal of `Jarvis` is to maximize the cumulative reward at the end of the episode which is a MDP problem defined formally as follows.

Definition 3.2.3. *A MDP consists of a tuple* (F, R, P, T, I, S_0). *F is the FSM of the environment; R is the reward function; P is the state transition probability table; T is the time period; I is the interval, and S_0 is the initial state of the environment. The goal of the agent is to find a strategy of actions in accordance with P, maximizing the total value of R of the next upcoming episode, for the environment in state S_i in F where* $0 \leq \mathrm{i} \leq \lceil T/I \rceil$.

3.2.4 Challenges

There are two key challenges when applying our model to cyber physical systems:

Unknown Reward Function

Quantifying the exact value of "utility" or "reward" gained or lost for any action performed is a challenge. It is worth noting that the $R()$ function described is not strictly Markovian in practice. In practice, $F()$ and $D()$ directly depend on inherently non-Markovian components, like environment dependant variables (electricity prices, temperature varia-

tions etc.) and user behavior, respectively. So, the value of the reward function R for a given state, action and *time instance* is not exactly known to the agent.

Safety/security of state transitions

Since the model is used to control a cyber physical environment, some state transitions can result in safety or security threats to the user specifically or to the general environment. Therefore, using a uniform state transition probability is impractical and state transitions must be context and environment dependant. The state transition probabilities of such unsafe state transitions should be zero. So, environment specific safety policies have to be identified before setting the model's state transition probability table P.

3.3 RL based Solution

In this section, we propose a RL based solution to the problem in Definition 5.2.2 subject to the aforementioned challenges. We propose algorithms to: 1) learn the safety/security policies in the environment and estimate the safe state transition probability table P_{safe}, and 2) estimate the reward function R_{smart} and learn the optimal and safe behavior in terms of user required functionality.

3.3.1 Safe State Transition

We identify state transitions which occur naturally in the environment during a specified learning phase as safe state transitions. The learning phase can either occur during setup of the environment or during a period defined by the user. During this learning phase, the user must approve every action taken as safe or manually perform it. This gives assurance that the corresponding state transition is 'natural' and the safety/security of the user is not violated. All state transitions are learnt in the form of trigger-action (T/A) behavior defined as:

T: Current State $S_t \rightarrow$ **A: Next Action** A_{t+1}

The T/A behavior from the learning phase is recorded to form the training dataset TD. However during the learning phase, there is the possibility of benign device malfunctions or

human errors. To avoid learning this benign anomalous activity as unsafe, we filter TD to remove these anomalous state transitions using a feed forward ANN. The ANN is trained by back-propagation using environment specific user labelled benign anomalous activities. The labelling of benign anomalies can be done offline by the user or through user prompts in real time according to the environment and user preferences.

All filtered state transitions in the learning phase and their instance counts are stored in memory. Finally, only state transitions having instance counts greater than the environment specific threshold $Thresh_{Env}$ have a uniform probabilistic distribution in the model. All other state transitions probabilities are assigned null values to prevent unsafe situations. The details of the approach are outlined in Algorithm 1.

Algorithm 1: Learning Safe State Transitions

Input: Training Dataset TD, Environment Context Variables $\{T, I, Thresh_{env}\}$
Output: State Transition Probability Table P_{safe}
Initialize: $P_{safe}[:,:] = 0, Count[:,:] = 0$
$Mem \leftarrow Filter_{ANN}(TD)$
while $d \leftarrow 0 : SizeOf(Mem)$ **do**
  while $t \leftarrow 0 : T$ **do**
    $(S, A) \leftarrow get(Mem, d)$
    $Count[S, A] \leftarrow Count[S, A] + 1$
    $SafeMem \leftarrow (S, A, Count[S, A])$
    $t \leftarrow t + I$
  $d \leftarrow d + 1$
while $(S, A, Count) \in SafeMem$ **do**
  if $Count > Thresh_{Env}$ **then**
    $S \leftarrow \Delta(S, A)$
    $P_{safe}[S, S] \leftarrow 1$

3.3.2 Reward Function Estimation

To address the challenge of unknown reward function, we estimate the reward function using user input and previous experiences. The estimated (smart) reward function for environment state S_t, action A, and *time instance* t is defined as:

$$R_{smart}(S, A, t) = \sum_{\mathrm{j}=0}^{\kappa} (f_{\mathrm{j}}) F_{\mathrm{j}}(s, a, t) - \frac{I}{kT} \sum_{\mathrm{i}=0}^{k} \omega_{\mathrm{i}}(s_{\mathrm{i}}, a)(t - t)$$

where the user inputs define the κ individual functionality requirements in terms of normalized functionality reward functions F_{j}. Examples of functionality rewards are: energy consumed, electricity costs, difference in optimal and current temperature, network usage etc. f_{j} are the weights associated with each functionality reward according to the user. The weights instill the concept of machine 'smartness' into the system, where the system is able to learn hypothetical strategies according to a combination of requirements or goals of the user. The user can alter weights to give more preference to one goal over the other but the essential strategy choices are made by the system keeping in mind the entire environment.

The second part in the above expression is the sum of all the estimated dis-utility caused by each device according to the their state and their pre-defined dis-utility values in episodes with time period T and interval size I. By dis-utility, we mean the discomfort or waiting time that might be caused to the user. This is an estimate obtained from the past behavior of the user, i.e., higher dis-utility means that the current state change is highly different from previous user behavior and vice-versa. t is the closest preferred *time instance* for the state-action according to past behavior. The higher the difference in t and t, the higher the estimated dis-utility is. The normalized ω_{i} function corresponds to the cost of dis-utility for the specific device. The reason for including the dis-utility in the reward function is to make sure that the agent in the virtual environment does not take actions purely for functionality optimization. For example, the agent deciding not to operate any of the appliances when power conservation is a functionality goal. It would give maximum functionality results (0 power consumption) but at a high cost with respect to user convenience. The reward function parameters f_{j} and ω_{i} are chosen to balance utility and dis-utility rewards according to a utility-disutility ratio $\chi = \frac{kT \sum_{\mathrm{j}=0}^{\kappa} f_{\mathrm{i}}}{I \sum_{\mathrm{i}=0}^{k} \omega_{\mathrm{i}}}$. Ideally, the value of χ should be chosen according to user preferences and environment configuration.

3.3.3 Q Learning Algorithm

We use a RL based approach to find the optimal quality function for the environment in terms of the estimated reward function R_{smart} as detailed in Algorithm 2. The FSM of the IoT environment is used to build a simulated environment where an agent can run multiple episodes to find the optimal and safe device actions for upcoming episodes. The agent balances exploration and exploitation according to the exploration rate ϵ. The exploration of the agent is constrained by security and safety policies at each step by using the safe state transition table P_{safe} learnt from Algorithm 1. The $Max(Q, c)$ function returns the c highest quality action for the given state and time stamp. Random batches from the agents prior experiences are selected and replayed to learn cumulative rewards according to the discount factor γ and batch size $Bsize$. Finally, the random batch with cumulative rewards is used to further train the DNN in order learn optimal Q table values for each state action pair and timestamp of the episode.

It is important to note that in our current formulation of the MDP, optimal actions are chosen with respect to the current state and timestamp of the episode. It is possible that in complex IoT environments, a more sophisticated policy identification in terms of higher order temporal trajectories is required. In this case, a MDP model relying on the immediate previous state would be a limitation. However, such a limitation can be overcome by incorporating temporal parameters for the episode in the state definition of the model. Such an approach would result in more fine-grained `Jarvis`'s optimization policies but at the cost of a higher number of state spaces and computation cost. The identification of ideal temporal parameters and specifics of the DNN, like number of hidden layers, activation functions, optimizers, network organization (feed forward/recurrent/convolutional), are beyond the scope of this work and should be chosen according to the device configurations and user requirements in the specific IoT environment.

3.4 Instantiation for a Smart Home Environment

In this section, we instantiate `Jarvis` for a smart home environment. We implement a prototype of `Jarvis` on the popular Samsung SmartThings [64] IoT platform. In what

follows, we first provide design details about the key components of `Jarvis`. Next, we qualitatively analyze the benefits of `Jarvis` using a small smart home example.

3.4.1 Design Details

The main components of the architecture are discussed in what follows.

Logging System

To capture all device related events, this component uses a logger app (see Figure 3.2) which subscribes to all device capabilities and attribute changes associated with these capabilities. More specifically, an attribute change on the device results in the creation of an event which triggers the logger app because of its subscriptions, after which the logger app proceeds to store the log. The logs in JSON format are shown as follows:

(Event.date, Event.data, User.info, App.info, Group.info, Location.info, Device.label, Capability.name, Attribute.name, Attribute.value, Capability.command)

Log Parser

The logs are parsed through a normalization function to generate the state model of the environment. The normalization function quantifies the device *Attribute.Value* to discrete device states and *Capability.command* to discrete device actions. Capabilities and attributes for a device can be numbers, strings, vectors, enum values etc., so we have manually developed device specific normalization functions. For all devices in the logs, their unique device states and actions are modelled as the FSM of the environment. In our prototype, we choose the time period value T as 1 day, interval value I as 1 min, and learning phase time L as 1 week. We choose these values intuitively as typically users have periodic behavior in terms of days/weeks and generally do not require demand response times below a minute. During the learning period, state transitions are recorded as *learning episodes* according to T, I, L values.

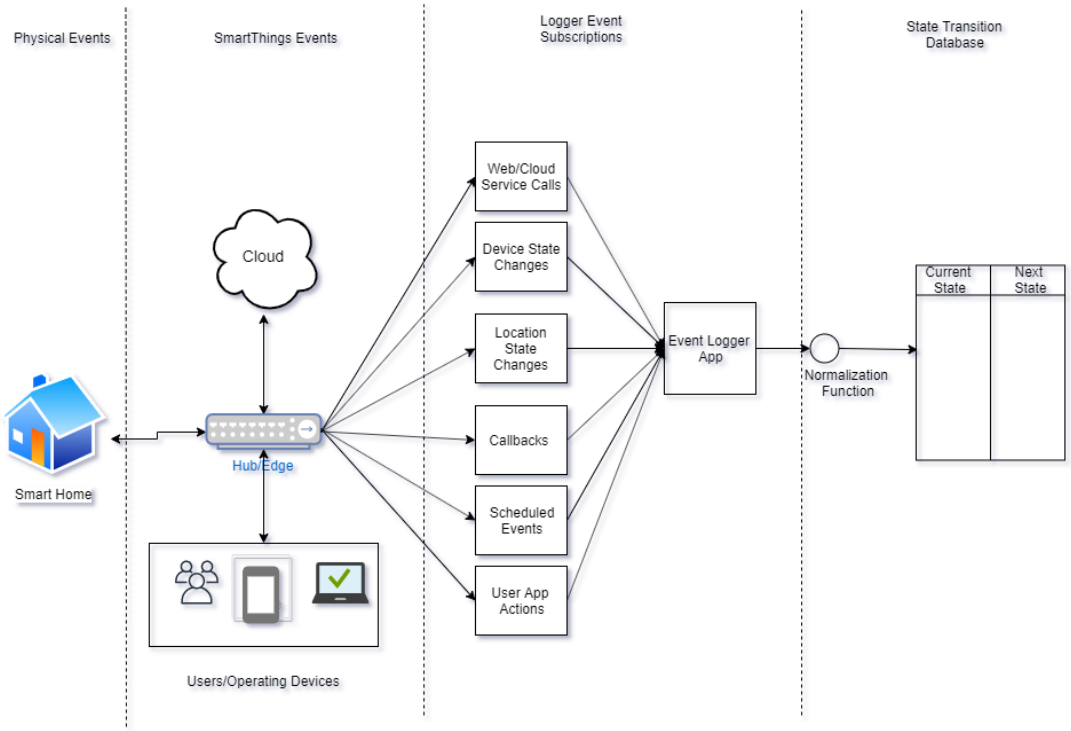


Figure 3.2. Logging and State Modelling for the SmartThings Architecture

Security Policy Learner

The security policy learner (SPL) is responsible for learning the safe state transitions table (P_{safe}) from *learning episodes* using Algorithm 1. By using results of user studies like [62] we manually generate labelled *benign anomalous* state transitions applicable to the smart home environment. Examples of such behavior, such as leaving fridge/oven door open, TV/oven on for short periods etc., are shown in section 3.6.1. The training set TD for the ANN is generated by randomly choosing state transitions from *learning episodes* and *benign anomalous* state transitions. The $Thresh_{env}$ should ideally be 0 as safety is critical in a smart home. A feed-forward multi-layer perceptron ANN with a single hidden layer is trained using back-propagation of TD samples.

Smart Reward Function

The smart reward function R_{smart} is estimated from environment specific normalized functionality reward functions F_{j}, the reward weights f_{j}, and the dis-utility functions ω_{i} provided by the user. The reward functions are generally scalar values available to the user in the environment through power meters, smart grids, and sensors. For example, if energy conservation is a functionality requirement, F_0 is directly proportional to power consumed in all device state transitions for the particular time interval which can be monitored by power meters. ω_{i}'s are chosen by the user according to the environment configurations. For example, appliances which require immediate action and do not consume a lot of power are devices with high dis-utility, like lights, door bells, locks etc., have very high values of ω_{i}. Low dis-utility devices, like HVACs, washing machines, dish washers etc., have high power costs and no immediate action requirements. The dis-utility value of R_{smart} for *time instance* t is generated from the delay in state transitions $t - t$, when compared to ideal *time instance* t in *learning episodes* and the device specific dis-utility ω_{i}. The definition and discretization of ω_{i} depend on the device and environment configurations, and thus we manually define ω_{i} values for each device. However, it is important to note this is just a one-time offline cost after which the devices can be used in other smart home contexts.

RL Environment

`Jarvis` dynamically builds a RL Q learning environment using the FSM of the smart home and the functionality requirements specified by the user. It builds a simulated virtual environment using the openAI gym[65]. We have developed a general environment in the openAI gym which can be used for various IoT setups and device configurations. An agent can explore all device states and take any action on any of the devices at every *time instance* with interval time I for an *episode* of time T.

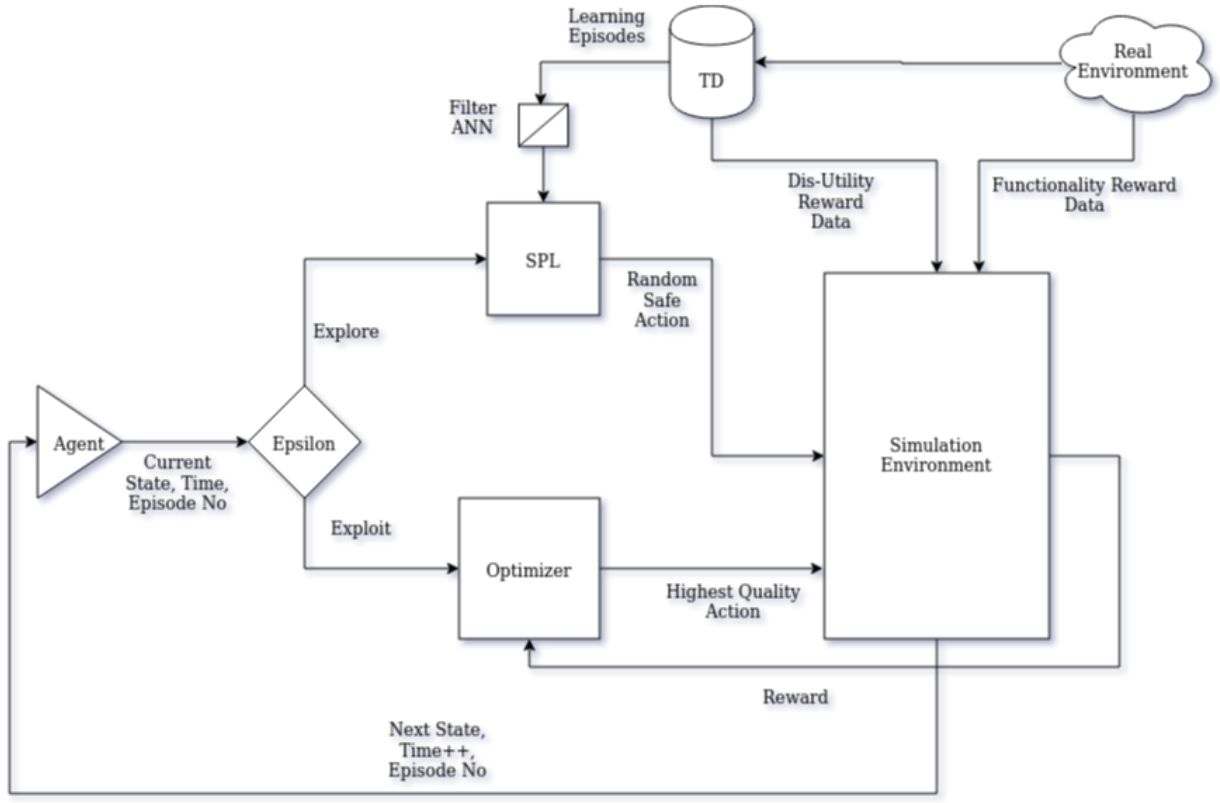


Figure 3.3. `Jarvis` RL Framework

Optimizer

The optimizer deploys a RL agent to traverse the RL environment in EP episodes to optimize the Q function according to Algorithm 3 (see Figure 3.3). A batch processing DNN, developed using TensorFlow [66] with two hidden layers and a learning rate of 0.001, is used to determine the optimal Q function values using first-order gradient-based optimization. The drawback of using the DNN arbitrarily is that it does not remember experiences from the previous episodes.

To address such problem, we defined the concept of 'replay' in Algorithm 3, where the agent remembers the actions and corresponding cumulative rewards, for all previous 'replays' of prior episodes and uses the cumulative rewards to optimize the Q function. The system stores all the experiences of the episode and 'exploits' this knowledge during a new episode

(see Figure 3.3). The agent uses a random batch from the past experiences to further train the neural network after every new episode.

Practical Deep Learning

The problem of action space explosion occurs when building the DNN as the action space increases exponentially. Consider an environment where each device can exist in only 2 states: on and off. Then even in this minimalist model, there can be a total of 2^k actions and we see that the action space grows exponentially with the number of devices. Maintaining a quality function of that magnitude for even small environments would be infeasible. To address this problem, we break up an action into mini-actions. The input to the DNN is the state of the environment. The output is an array of rewards for each 'mini-action' instead of a whole environment action. A 'mini-action' is defined as an intermediate action performed on only one device in an interval of the episode. In an asynchronous system where the state of the global system is characterized by the composition of the internal state of each individual device, a parallel execution can always be serialized into the execution of the individual device mini-actions. There can be at most k mini-actions possible in one interval of the episode, i.e one per each device. This allows the neural network to learn quality values efficiently for the mini-action space (grows linearly) rather than total action space (grows exponentially). There can only be k mini-actions for each trigger. So, in a single interval, the agent can take a maximum of k mini-actions to form a complete action.

3.4.2 Example: Analysis and Discussion

The example smart home consists of 5 devices: a smart lock, a door touch sensor, a smart light, a smart thermostat controller and a temperature sensor. The FSM of the environment is shown in Table 3.1.

Safety

To demonstrate the effectiveness of our approach in learning safe state transitions, we install five common IFTTT [58] apps in the example smart home (see Table 3.2). For

instance, App 1 opens the door when a user arrives home. In terms of T/A behavior, the action (A) here is 'opening the door' and the trigger (T) is 'the door touch sensor or camera identifying the authorized user at the door'. In our model, this trigger would be represented as the door sensor being in "Auth. User" or in p_{1_1} state.

If no safety policies were identified by `Jarvis`, the T/A behavior of the apps would be as described in columns 4-5 of Table 3.2. 'X' indicates that the device can be in any state (or take any action) and is not directly affected by the app. 'O' indicates that no action should be taken on the device. We see that these triggers and actions are specified without taking into account the environment context and only consider the individual app and the devices associated with it. They also do not take into account the security or safety of the environment. For example, in the context of App 1, it is not safe for the door to be unlocked when the user is not at home or sleeping. Similarly, turning the thermostat on when the user is not home or setting the temperature to unsafe high values in App 2's context. Turning off temperature and door sensors can also lead to safety issues. Next in columns 6-7, we show the safe T/A behavior learnt from Algorithm 1. We can see that `Jarvis` is able to learn the *natural progression* of device states, from which it automatically learns the security policies. The only exception occurs in the case of emergency situations raised by certain sensors like smoke alarms, fire sensors etc. The safe functioning of these devices cannot be determined from natural progression as such scenarios occur only in rare situations. So, we have to adjust our model to add security/safety policies for such devices manually as our security policy learner does not in these situations. In Section 3.5, we discuss using active learning approaches to learn *safe non-natural behavior* and make such adjustments autonomously.

Effectiveness of Constrained Exploration

We specify $k = 3$ example user required functionalities in the smart home environment: Energy Conservation, Electricity Cost Minimization, and House Temperature Optimization.

We compare the action quality learnt without any security constrains (unconstrained exploration) to action quality learnt with `Jarvis` (constrained exploration) in terms of the given functionalities in Table 3.3 using 8 common T/A behaviors. For instance, turning off

the lights and thermostat when the user leaves the house. An unconstrained optimizer with a goal of conserving energy would turn off all the devices which can cause safety concerns like turning off essential sensory devices like fire-alarms, door lock sensors and temperature sensors. In the case of the single goal of temperature optimization, the unconstrained exploration would lead to turning on the thermostat even when the user is not home which is a safety as well as energy waste issue. We see that unconstrained optimization of actions according to functionality leads to unsafe situations, which are avoided using the constrained exploration of the `Jarvis` RL environment.

3.4.3 Dis-utility vs Safety.

It is important to notice the difference between the dis-utility in the R_{smart} reward function and the learnt safe transitions P_{safe} from the SPL component. High dis-utility and unsafe behavior might share some common patterns but they have different definitions in our model. While both are probabilistic and are learnt during *learning episodes*, they differ in their enforcement: the former is 'strict' as it represents user safety in the model, while the latter is 'lapse' as it represents user inconvenience.

We illustrate the difference using the smart home example with energy conservation as the functionality goal. A user leaves his house meticulously at 8 am and locks the door during the learning phase. Then while exploring the RL environment, the agent learns that it gains a higher reward when the lights and thermostat are turned off exactly at 8am because of the low dis-utility in the reward function. So the agent learns that for the trigger T_1 ="locking the door at 8am", the action A_1="turning the thermostat and lights off" is the highest quality action. Now suppose that the user is not so meticulous and leaves sometimes between 6am-8am. Now the agent finds that the reward for taking A_1 is lower as the utility part is the same but the dis-utility part for this action is higher as the user does not leave exactly at 8am. So, we see that although the reward is lower, if the user were to leave the house at 6am instead of 8am, the optimizer will still predict the optimal action A_1 even if the reward is lower. A security policy on the other hand is strict and does not allow any unsafe action. In the same example, for the trigger T_1, the action A_2 = "power off the lock"

would cause a security concern. The SPL component learns that for T_1, the action A_2 must never taken place. So, in the RL environment this T/A behavior would never occur unlike the high dis-utility case seen before.

3.5 Evaluation

We evaluate the instantiation of `Jarvis` for a smart home environment quantitatively in terms of two metrics: security and functionality. We first describe the simulated testbed for our evaluation. Next, we evaluate the safety and security policies learnt by the SPL component. Then, we analyze the effectiveness of the Optimizer in terms of three user defined functionalities.

3.5.1 Testbed

We build a virtual testbed (see Figure 3.4). It consists of five users and two locations: Home A and Home B. We use the open smart home simulator (OpenSHS) [67] to build data sets for home A using simulated daily user activities [68]. The Home B's datasets are simulated from real world data collected by user studies [68]. We use 55,156 samples of user generated benign anomalies [62] for building the training dataset TD for the testbed.

3.5.2 Safety and Security

Building Malicious Data-sets: There has been significant research on identification and analysis of safety and security violations that occur in smart homes [1], [2], [69]. After reviewing prior work, we define the following six types of malicious activities:

1. Type 1: T/A safety violations
2. Type 2: Integrity/access control violations
3. Type 3: General security/conflicting actions/race condition violations
4. Type 4: Safety violations by malicious apps
5. Type 5: Insider attacks

We give examples of each type and details of how violations from prior work are reproduced for our testbed in Appendix 3.6.2. In total, we develop 214 security violation instances

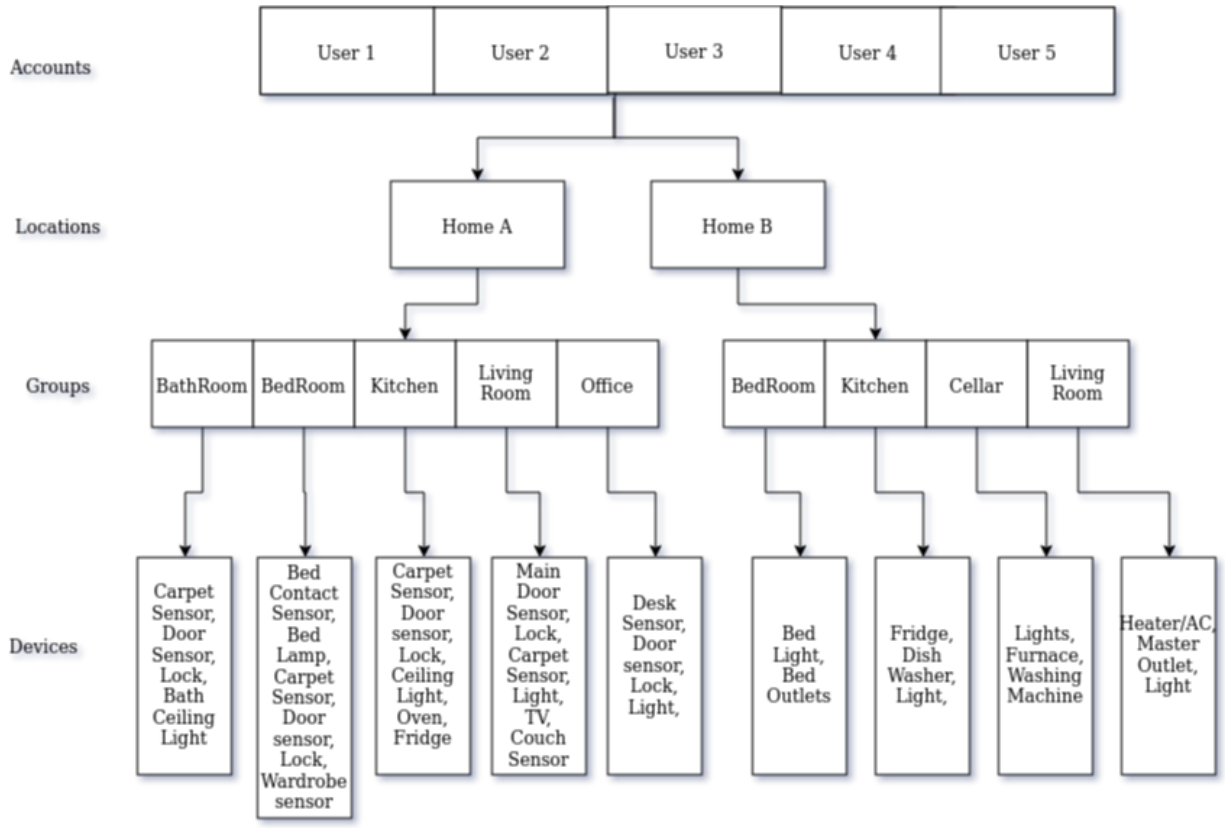


Figure 3.4. Overview of the evaluation setup

with a breakdown as follows: Type 1 (114), Type 2 (40), Type 3 (40), Type 4 (10), and Type 5 (10).

Security Analysis: To evaluate safety provided by the SPL, we manually engineer each of the 214 malicious state transitions in random episodes of the RL environment to generate 21,400 *malicious episodes*. We then play these malicious episodes in the RL environment after the *learning episodes* of the SPL component are complete. We find that the SPL is able to flag 100% of all the malicious state transitions in the *malicious episodes*.

3.5.3 False Positives

We evaluate the SPL in terms of false positives resulting from benign user anomalies that are classified as malicious state transitions. We use data-sets[62] that have samples of anomalous activity collected from user studies. The participants of the studies were asked

to define anomalous activities themselves and then perform additional simulations of these defined anomalies. These data-sets have activities for a period of one month in smart home A with interval size of one minute. We engineer instances of benign anomalous state transitions collected from [62] in random episodes after the *learning episodes* of the SPL are complete for smart home A to generate 18,120 *benign anomalous episodes*.

The benign anomalies are detected and filtered by the ANN used by the SPL. We find that 99.2% of the *benign anomalous episodes* are correctly classified by the ANN and the false positives are 0.8%. The ROC graph is shown in Figure 3.5.

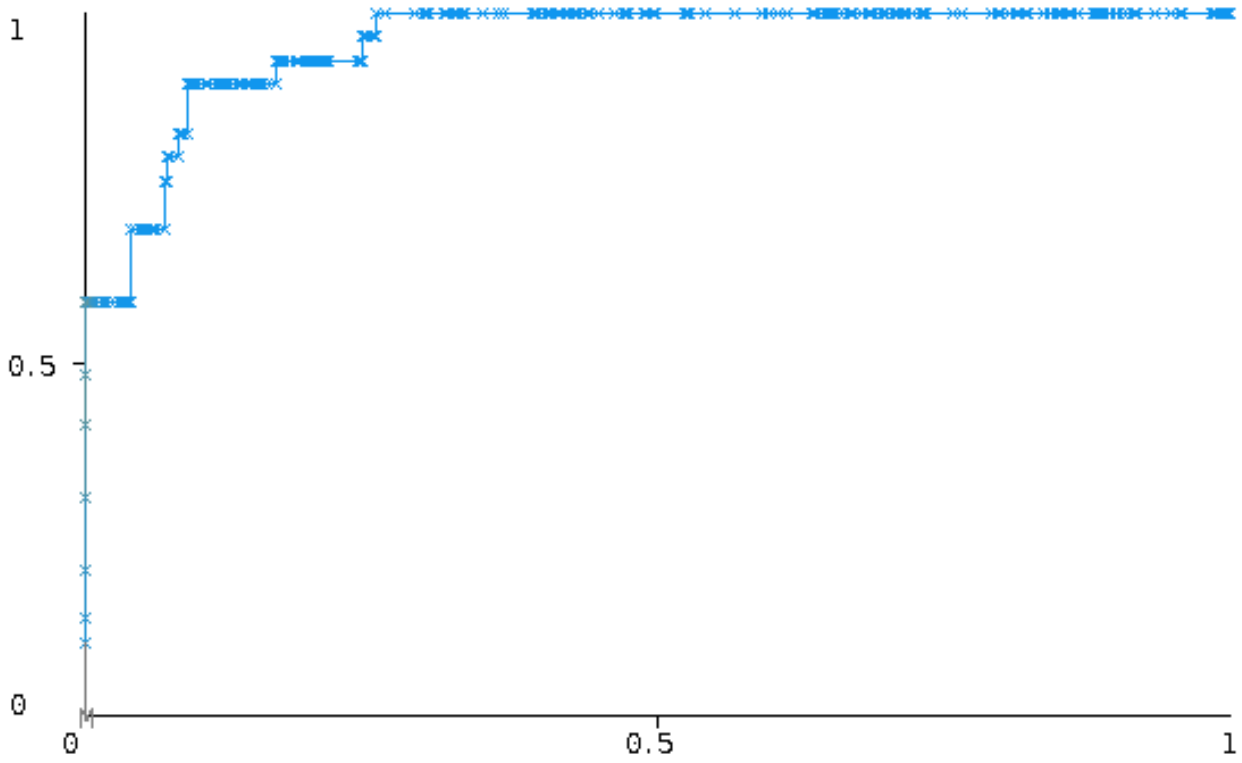


Figure 3.5. ROC curve for filtering accuracy of the SPL

3.5.4 Functionality

For analysing the functionality optimizer component, we take $\kappa = 3$ hypothetical goals of the user: energy optimization, energy cost minimization, and ideal temperature maintenance. We define normalized reward functions for each as: F_0 - meter readings of power

usage (energy usage), F_1 - electricity costs for power usage according to DAM (day-ahead-market) prices[70] (energy cost), and F_3 - temperature difference between day ahead forecasted temperature and HVAC readings. We manually define the values of the dis-utility reward functions (ω_{i}) for $k = 11$ devices such that total dis-utility and total utility rewards are equally balanced: $\chi = 1$ and $f_1 + f_2 + f_3 = 1$. This makes sure that the optimized actions never cause more dis-utility than functionality when traversing the environment.

We perform experiments for a range of rewards weights $f_1, f_2, f_3 \in [0.1, 0.9]$ for 30 random days from the dataset from [68]. For each day, `Jarvis` produces the optimal strategy or Q table based on the f_{j} values. We compare the normal user behavior with `Jarvis` optimized behavior; the results of the comparison are reported in Figures 3.6, 3.7, and 3.8 for each functionality, respectively. We can see that the optimized actions provide an advantage over normal behavior in the range from $f_{\text{j}} = 0.1$ to $f_{\text{j}} = 0.9$. The shaded region in the graphs is defined as the *safe benefit space* of the environment for each individual functionality. It is important to note the user may take some actions of the day manually and depend on `Jarvis` for other actions. In this case, `Jarvis` still suggests the best possible action from the *safe benefit space* for whichever state the environment has reached because of user actions.

3.5.5 Analysis of the Benefit Space

We can see that `Jarvis` learns to take 'smart' decisions by exploring the *safe benefit space*. With a variation of f_{j} and χ values, `Jarvis` is able to learn a variety of hypothetical environment specific ethics for state space exploration. For instance, configuration values $f_1 = 0.9, f_2 = 0.05, f_3 = 0.05$ result in `Jarvis` having highly energy usage conscious ethics. On the other hand, for values $f_1 = 0.2, f_2 = 0.2, f_3 = 0.6$, `Jarvis` learns more balanced ethics which emphasizes maintaining house temperature at the expense of energy usage and costs.

Learning hypothetical human objectives or ethics to model the reward function is a hard problem because of the intractability of value of information to the user for a given environment. It is the focus of various active learning schemes [71], [72]. `Jarvis` provides an effective way of producing hypotheticals along the *safe benefit space* for each functionality

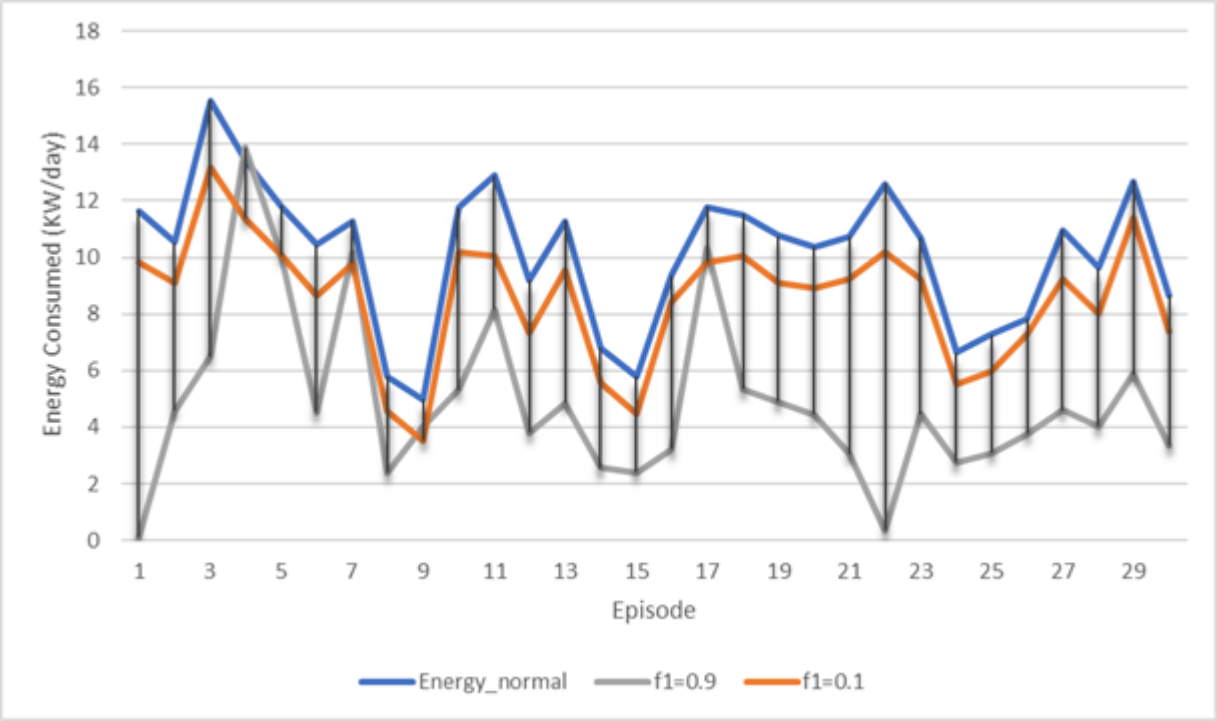


Figure 3.6. Energy Conservation

defined by the user. The reward function model can then be learnt from user feedback on these hypothetical behaviors.

3.5.6 Limitations of Unconstrained Exploration

In order to see the benefits and drawbacks of unconstrained exploration, we perform an experiment to compare the behavior of `Jarvis` with and without the SPL component. The results are reported in Figure 3.9. We can see that unconstrained exploration promises higher rewards but at the same time has an average of 32 safety violations per episode. We refer to the grey region of the graph as the overall *unsafe benefit space* of the environment (the overall *safe benefit space* is shown in orange). Since the SPL is probabilistic and prone to false positives, of the 32 violations, there might be some which are benign and should be allowed. In this regard, `Jarvis` has a bias towards 'safer' solutions based on Occam's razor [73]. This is a valid assumption in IoT environments especially in smart homes where user safety is paramount. However, there may be violations which are benign (i.e., false

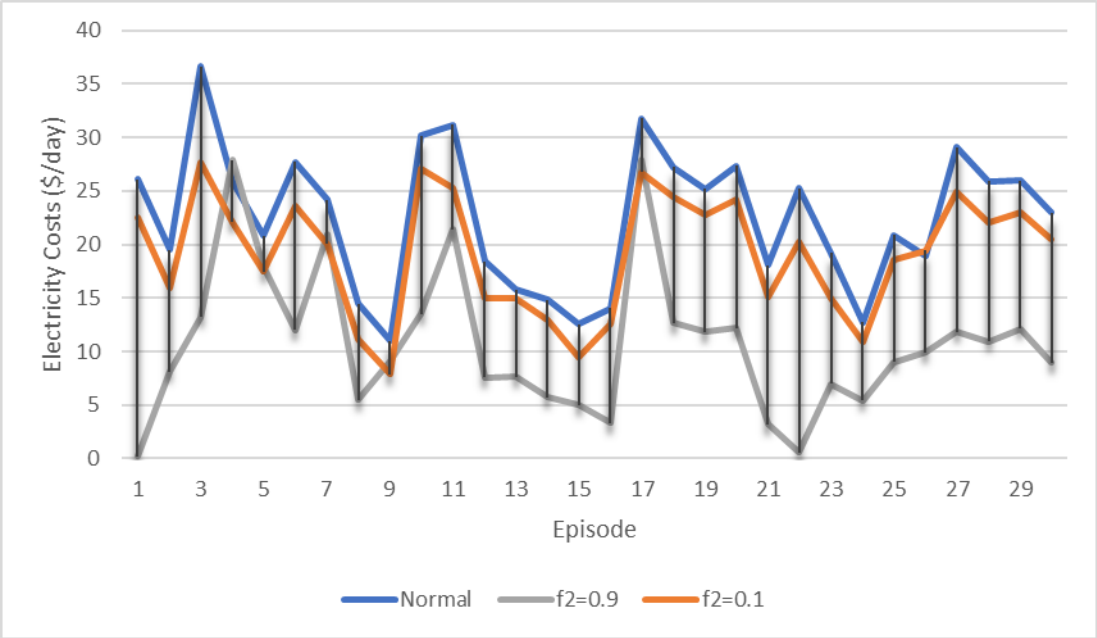


Figure 3.7. Energy Price Minimization

positives) or provide enough functionality benefits such that they are acceptable to the user in certain scenarios. User feedback on these actions in the *unsafe benefit space* can be used to correctly classify these actions as benign or malicious. Using such active learning methods to utilize the *unsafe benefit space* is a promising future direction of research.

3.6 Related Work

Approaches for securing IoT environments are based on vetting apps by building static or dynamic smart app models [1]–[3], [61], [74]–[76]. The models are generated using a combination of device capabilities, user prompts, app permissions, and event subscriptions. After which, some form of forward symbolic execution [77] is used to explore all feasible paths in the built graphs. In order to build multi-app models, a set of the apps which share device events and graph causality are merged together to form a union of the app's state models. Then a general purpose model checking, like SPIN [78], is used to detect policy violations in the generated app models. These app-based modelling approaches are prone to state space and path explosion problems especially with large numbers of heterogeneous apps in terms of functionalities, dependencies, and attribute values. These systems work for

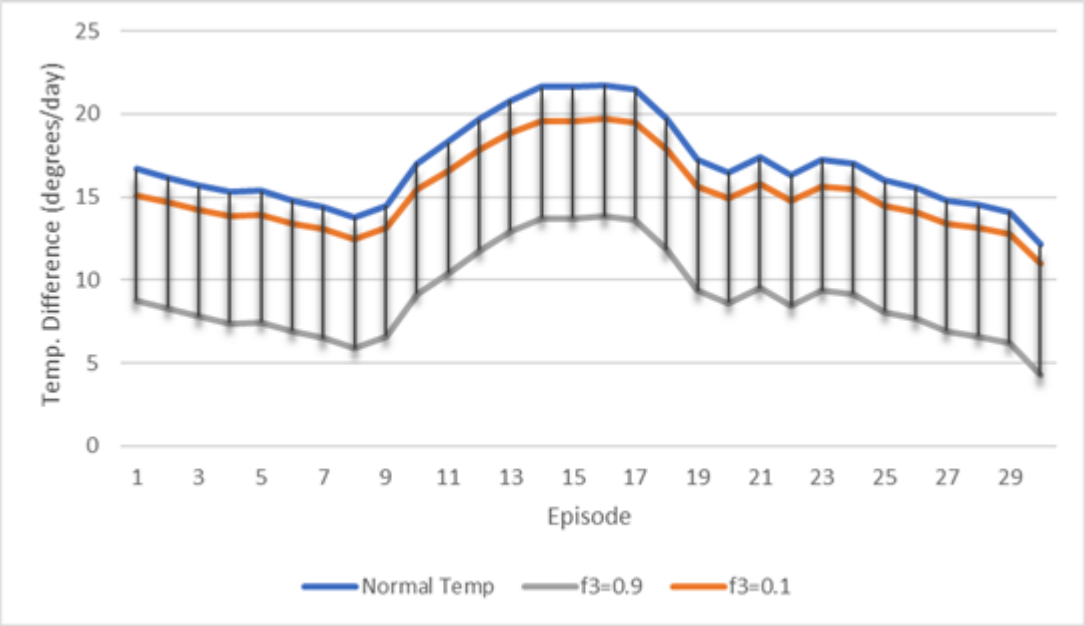


Figure 3.8. Temperature Difference Optimization

small dynamic models built by limiting app-interactions, interleaving or using just a specific set of apps and devices triggered in a particular order. On the other hand, the `Jarvis` state model is constructed only from device states and actions and hence not affected by the number or variety of apps running in the environment. Also, the safety and/or security of the decisions is assessed from pre-defined policies set by users [1], [2]. Defining such policies for different devices and app configurations requires significant human effort. An approach for the automated learning of such safety policies in a smart home by observing natural behavior of the devices is proposed in [79]. Our system is similar in this regard as it learns security policies by observing prior T-A behavior during the learning phase.

The other drawback of those approaches is that they only detect unsafe/insecure states and do not consider the functionality requirements or goals of the user. Approaches have been proposed for building optimization models for individual functionality requirements like energy usage [80], [81], electricity prices[9], [82], and thermal comfort[83], [84] for smart homes and smart grids using various statistical and machine learning tools. In [7], a RL based system is used to manage the energy in a residential setting with a smart grid. For smart homes, a deep RL based framework is developed using energy and user discomfort as

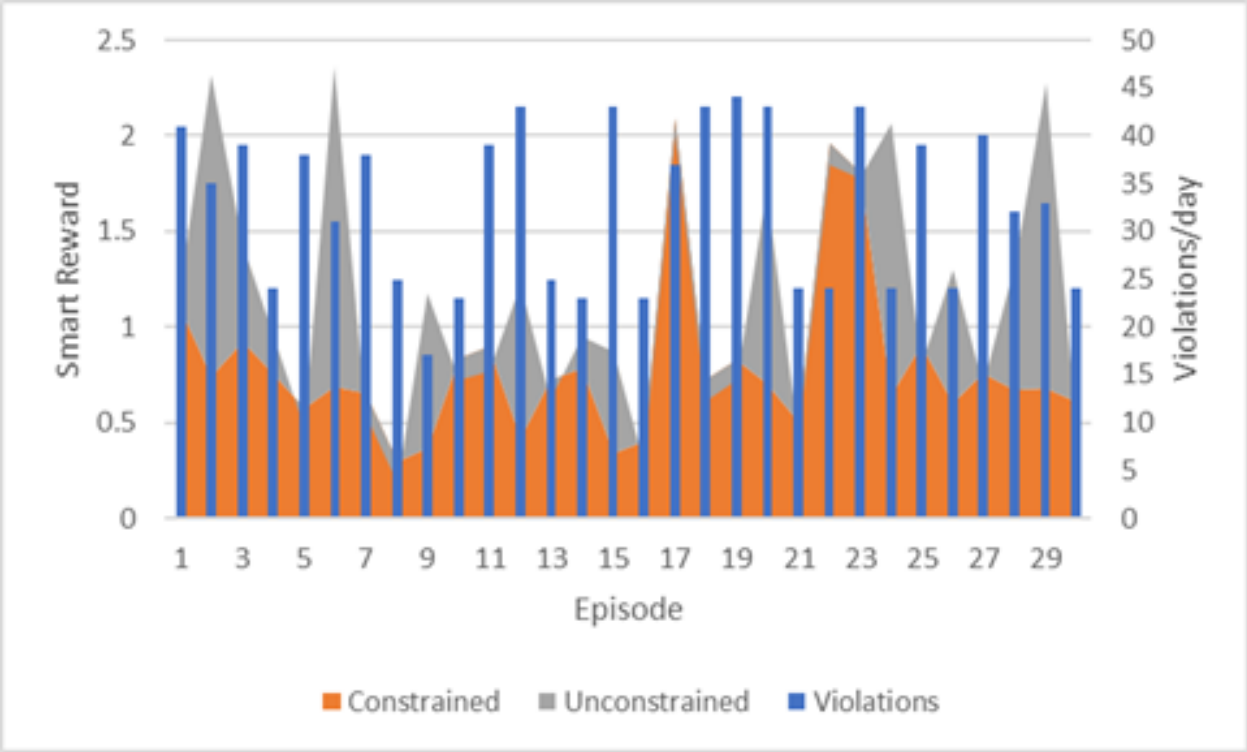


Figure 3.9. Unconstrained vs Constrained Exploration Benefit Space

opposing rewards [6]. Other learning systems are used to maintain the state changes of power intensive HVAC, washing machines, industrial appliances etc. [85]. However, these systems are designed to have simplistic state models with specific set of devices, thermal dynamics and simple environments. These cannot be directly used for the complex and heterogeneous IoT environment with multiple apps and devices. Also, those approaches do not address security and safety.

3.6.1 Benign User Anomaly Examples

1. Common benign anomalous user activities:
   (a) Leaving Fridge Door Open
   (b) Leaving oven on for a long time
   (c) Leaving Main Door open
   (d) Leaving lights on for a long time
   (e) Leaving TV on

(f) Leaving wardrobe open

3.6.2 Safety Violation Examples

1. Type 1: Trigger-Action Application specific Safety Violations
   (a) The main door is unlocked when user is not home
   (b) The main door is unlocked when user is sleeping
   (c) The lights of the room are off when motion is detected
   (d) The lights of the room are off when a user unlocks the door
   (e) The door is opened when there is no motion outside/inside the house
   (f) Lights are on when the user is sleeping
   (g) Appliances like oven,TV etc. are on when user is sleeping/away from home
   (h) Appliances like fridge/security system/fire alarm are off
   (i) Appliances interfering with each other eg: turning on heater when ac is on and vice-versa
2. Type 2: Integrity/Access Control Violations
   (a) Unauthorized Users performing actions
   (b) Users performing actions in unauthorized groups/locations
3. Type 3: General security/conflicting actions/race condition violations
   (a) A single user performing multiple actions repeatedly resulting in conflicting states on a device
   (b) Multiple users performing the same action on the device multiple times
   (c) Multiple users performing conflicting actions on the device
   (d) User performing an action which acts as a trigger to perform an action on the same device
4. Type 4: Malicious apps causing safety violations
   (a) App turns off lights when motion is detected (e.g., to prevent brightening of the path outside the house)
   (b) App turns lights/devices off after user specified time to save energy

(c) App turns lights to dim mode when user leaves home and door is unlocked after some time

(d) App turns on devices when user is home and turns them off when the user is not

(e) App 1 turns on lights when alarm sounds, App 2 assumes user is home when lights turn on and locks the door

(f) App turns off all devices when user is sleeping

5. Type 5: Insider attacks

(a) Heater is set to unsafe temperature and door is locked by insider

(b) Oven is left turned on for long time by insider when user is sleeping/not home

(c) Door is unlocked by an insider when no user is at home/sleeping

Constructing Malicious Data-sets: By simulating instances of these activities in smart home B, we generate *malicious* labelled state transitions. We build the malicious data-set for each type as follows. For type 1, we manually go through the trigger-action application specific security policies defined in [1], [2] and identify activities applicable to the devices in our testbed. For example, R.1 in [2] is applied to five devices in smart home A, R.14 is applied to one device in smart home B, and to sixteen devices in smart home A. In total, we identify 114 such violations by applying policies R1-R30 to our testbed. For type 2, we survey prior work on access control in IoT environments [74] to construct 40 unauthorized access instances which violate various security policies in smart homes A and B. For type 3, we construct violation instances from the general security (G1-G4) rules defined in [2]. For type 4, we use the malicious app interactions identified in [1] to build ten instances in smart home A. For type 5, we emulate common real world insider attacks resulting in thefts, breaches and safety issues to build ten instances.

3.6.3 State Space Explosion Mitigation.

In terms of the entire environment context, each 'X' value can exist in as many states the device could be in. So, for App 1 there can be a total of $2 * 3 * 4 = 24$ triggers as D_2, D_3, D_4 can exist in $2, 3, 4$ states respectively. Similarly, there can be $2*2*4*2 = 32$ actions possible. This would result in a total number of $24 * 32 = 768$ trigger action pairs for one app in a

simplistic smart home. Typically a number large number of apps are deployed in the smart home. Also, the number of triggers would depend on the number of 'X's and the number of states a device can exist in. It would then be the case that in a large environment it is infeasible to observe app based T/A behavior. This problem is mitigated by observing whole environment device T/A behavior rather than individual app T/A behavior. Also, constraining the state space exploration of the RF agent by only allowing safe trigger action further reduces the state space used.

3.7 Summary

In this chapter, we have proposed `Jarvis`, a RL framework for IoT environments that learns device actions to optimize user defined goals but whose exploration is constrained by dynamically identified safety and security policies. We have instantiated `Jarvis` for a smart home environment and have shown that it provides considerable benefits in terms of functionality and safety.

Algorithm 2: Learning Optimal State Action Quality

Input: Simulated Environment Env; Estimated Reward Function $R_{smart}() : \{SS, AS\} \rightarrow \mathbb{Z}$; Maximum Episodes EP; Exploration (Rate , Min, Decay) $= (\epsilon, \epsilon_{min}, \epsilon_{decay})$; Safe State Transition Table $P_{safe}[S, S]$; Batch Size $BSize$; Discount Rate γ; Preferable Loss L_p

Output: State Action Quality Function $Q : \{SS, n\} \rightarrow AS$

Initialize: $Q[\,:,:\,] = 0$

while $ep \leftarrow 0 : EP$ **do**
- $S_{curr} = Env.Init()$
- $S_{next} = S_{curr}$
- **while** $t \leftarrow 0 : T$ **do**
  - **if** $Random() \leq \epsilon$ **then**
    - ▷ Exploration: **while** $P_{safe}[S_{curr}, S_{next}] \neq 0$ **do**
      - $A_{curr} = Random(AS)$
      - $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$
  - **else**
    - ▷ Exploitation: $count = 0$
    - **while** $P_{safe}[S_{prev}, S_{curr}] \neq 0$ **do**
      - $A_{curr} = Max(Q[S_{curr}, t], c)$
      - $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$
      - $c \leftarrow c + 1$
  - $R_{curr} = R_{smart}(S_{curr}, A_{curr}, t)$
  - $Mem \leftarrow (S_{curr}, A_{curr}, R_{curr}, t)$
  - **if** $Mem > Bsize$ **then**
    - $Q = Replay(Bsize)$
  - $t \leftarrow t + I$
  - $S_{curr} \leftarrow S_{next}$
- $ep \leftarrow ep + 1$

procedure REPLAY($Bsize$, L_p)
- $MiniBatch \leftarrow Sample(Mem, Bsize)$
- **while** $(S, A, R, t) \in MiniBatch$ **do**
  - $S_{next} \leftarrow \Delta(S, A)$
- $count = 0$
- **while** $P_{smart}[S_{next}, \Delta(S_{next}, A_{next})] \neq 0$ **do**
  - $A_{next} = Max(Q[S_{next}, t + 1], count)$
  - $count \leftarrow count + 1$
  - $R_{cum} = R + \gamma R_{smart}(S_{next}, A_{next}, t)$
- $MiniBatch[(S, A, R, t)] \leftarrow (S, A, R_{cum}, t)$
- $Loss \leftarrow DNN_{Train}(Q, MiniBatch)$
- **if** $\epsilon \geq \epsilon_{min}$ *AND* $Loss \leq Loss_p$ **then**
  - $\epsilon \leftarrow \epsilon * \epsilon_{decay}$
  - return Q

end procedure

Table 3.1. Smart Home Environment FSM

Device Type	Device	p_{i_0}	p_{i_1}	p_{i_2}	p_{i_3}	a_{i_0}	a_{i_1}	a_{i_2}	a_{i_3}
Lock	D_0	locked(outside)	unlocked	off	locked(inside)	Lock	Unlock	Power Off	Power On
Door Sensor	D_1	Sensing	Auth. User	Unauth. user	-	Power Off	Power On	-	-
Light	D_2	off	On	-	-	Power Off	Power On	-	-
Thermostat	D_3	Heat	Cool	Off	-	Increase Temp.	Decrease Temp.	Power Off	Power On
Temp. Sensor	D_4	Above Opt. Temp	Below Opt. Temp	Optimum	Fire Alarm	Power Off	Power On	-	-

Table 3.2. Comparison of normal vs safe T/A behavior

App	Description	Devices Involved	Trigger	Action	Safe Triggers	Safe Actions
1	Door unlocks when authenticated user arrives at the door	D_0, D_1	$(p_{0_0}, p_{1_1}, X, X, X)$	(a_{0_1}, X, X, X, X)	$(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_0})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_1})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_2})$	$(a_{0_1}, O, a_{2_1}, X, X)$
2	Maintain optimal temperature in the house	D_3, D_4	(X, X, X, X, p_{4_0}), (X, X, X, X, p_{4_1})	(X, X, X, a_{3_1}, X), (X, X, X, a_{3_0}, X)	$(p_{0_1}, p_{1_0}, X, X, p_{4_0})$ $(p_{0_1}, p_{1_0}, X, X, p_{4_1})$	(O, O, O, a_{3_1}, X), (O, O, O, a_{3_0}, X)
3	Lights turn on when user arrives home	D_0, D_1, D_2	$(p_{0_0}, p_{1_1}, X, X, X)$	(X, X, a_{2_1}, X, X)	$(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_0})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_1})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_2})$	$(a_{0_1}, O, a_{2_1}, X, O)$
4	Door is opened/lights turned on when fire alarm is raised	D_0, D_2, D_4	(X, X, X, X, p_{4_3})	$(a_{0_1}, X, a_{2_1}, X, X)$	-	-
5	Thermostat/lights turned off when user leaves the house	D_0, D_1, D_3	$(p_{0_0}, p_{1_0}, X, X, X)$	$(X, X, a_{2_0}, a_{3_2}, X)$	$(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_0})$ $(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_1})$ $(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_2})$	$(O, O, a_{2_0}, a_{3_2}, O)$

Table 3.3. Comparison of action quality for Unconstrained vs Constrained Exploration

Function	Trigger	Trigger Description	High Quality Action	High Quality Safe Action	Action Description (quality vs safety)
Energy Conservation	$(p_{0_0}, p_{1_0}, X, X, X), t$	User leaves the house and locks the door	$(a_{0_2}, a_{1_0}, a_{2_0}, a_{3_2}, a_{4_0}), t$	$(O, O, a_{2_0}, a_{3_2}, O), t$	Turn off the lights and thermostat only not all appliances
Energy Conservation	$(X, X, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn the thermostat off
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_0}), t$	Temperature drops below optimum and user at home	$(O, O, O, a_{3_0}, O), t_p$	$(O, O, O, a_{3_0}, O), t$	Power on the heater at closest off peak hour t instead of waiting for optimal non-peak hour t_p
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_1}), t$	Temperature goes above optimum and user at home	$(O, O, O, a_{3_1}, O), t_p$	$(O, O, O, a_{3_1}, O), t$	Power on the cooler at closest off peak hour t instead of waiting for optimal non-peak hour t_p
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn the thermostat off
Temperature Optimization	$(X, X, X, X, p_{4_0}), t$	Temperature drops below optimum	$(O, O, O, a_{3_0}, O), t$	$(O, O, O, a_{3_0}, O), t$	Power on the heater to optimize the temperature but only at t when the user arrives home
Temperature Optimization	$(X, X, X, X, p_{4_1}), t$	Temperature goes above optimum	$(O, O, O, a_{3_0}, O), t$	$(O, O, O, a_{3_0}, O), t$	Power on the cooler to optimize the temperature but only at t when the user arrives home
Temperature Optimization	$(X, X, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn thermostat off

4. JARVIS-SDN: SECURITY CONSTRAINED RL FOR RATE CONTROL IN SDN ENVIRONMENTS

The use of machine learning (ML) techniques in the control plane of software defined networks (SDNs) provides enhanced approaches to traffic engineering, such as maximizing quality of service (QoS). Generally, QoS is determined by the interplay within various network functionalities such as rate control, routing, load balancing, and resource management. This interplay can become very complex. The benefit of ML techniques is that they can model complexity given sufficient representative data to train upon. However, the diversity and scale of current networks together with the diversity of traffic behavior hinder the task of gathering data that captures enough sets of behaviors for training. This poses a challenge to classical ML. Reinforcement Learning (RL), on the other hand, relies on learning optimal policies online based on system state using a model-free approach. These policies are more likely to transfer over to a new environment, and these characteristics make them more suitable for network control. RL based frameworks have thus already been proposed for specific functions within networks, such as for controlling routing [13], traffic rate control [14] and load balancing [15].

Network control solutions require optimization across multiple functionalities, not just a single one. Current uses of RL for network control focus on optimizing a single functionality, which makes these existing solutions difficult to deploy in real networks. For example, learning a policy which maximizes the throughput of the network (functionality 1: optimal routing) can come at the cost of unfair bandwidth consumption by a set of users (functionality 2: per user bandwidth fairness). Perhaps even more critical is the case of security policies. For example, learning a policy which maximizes the throughput of the network (functionality 1: optimal rate control for QoS) can unknowingly facilitate the propagation of a high throughput Denial of Service (DoS) attack.

To address those issues, we propose `Jarvis-SDN`, an adaptation of our constrained RL framework for IoT [86] to SDNs. In `Jarvis-SDN`, a RL based agent using Deep Q-Learning learns optimal policies for a SDN controller to optimize across multiple network functionalities while maintaining security. Examples of such functionalities include optimal rate

control (measured by per user throughput), routing optimization (measured by a metric like latency), availability of device resources, or path quality (metrics such as loss rate, or jitter). The basic idea is to define the reward to the agent as a weighted combination of individual functionality performance metrics, including a metric for security behavior of the system. A key challenge in applying our previous framework [86] to SDNs is that it is not obvious how to define performance metrics for security (see [87], [88] for proposals and discussions regarding security metrics). Our approach for quantifying security is to measure the ability to protect against known attacks. We first build offline 'attack signatures' from packet captures of previously seen attacks using different ML techniques: Decision Trees, Random Forests, Deep Neural Networks (DNN) and Deep Q-Networks (DQN). These attack signatures are then used by the RL agent to determine a quality value for the network state depending on the perceived threat a network flow has on the current and near future states of the network.

To summarize, we make the following contributions:

1. A context independent RL framework, `Jarvis-SDN`, constrained by security policies for SDN controllers.
2. A novel approach to build quantifiable security metrics (attack signatures) for network flows using DQN.
3. Extensive evaluation of different ML techniques to build attack signatures using the CICIDS dataset [89] consisting of network attacks, such as DoS, DDoS, Brute-Force and Web based attacks. We find that DQN based signatures perform better than other ML techniques.
4. Through an instantiation of `Jarvis-SDN` for a SDN controller with the goal of optimal rate control, we show that the RL agent learns desirable behavior for malicious and benign flows.

The rest of the chapter is organized as follows. In Section 5.1, we give some background on Deep Q-Learning. Next, in Section 4.2, we formally define the system model and discuss key challenges. In Section 4.3, we define and analyze our RL based DQN approach for building attack signatures. In Section 4.4, we instantiate the `Jarvis-SDN` framework in a simulated network for optimal rate control and analyze its effectiveness. Finally, we discuss

related work in Section 5.6 and in Section 5.7 we conclude and outline directions for future work.

4.1 Background on Deep Q-Learning

A Reinforcement learning (RL) framework [63] (see Figure 5.1) is a probabilistic state transition environment where state transitions are caused by actions executed by an agent and every state-action pair (s, a) is assigned a reward value r given by a reward function $R(s, a)$. A RL agent traverses the environment according to a policy $\boldsymbol{\pi}$ that selects an action to execute in a given state for policy parameters θ and receives the rewards accrued over all the state transitions that happened according to θ. In a Q learning framework, the goal is to find the optimal policy which maximizes the cumulative reward through a function $Q(s, a)$ that estimates the expected cumulative reward the agent will get at the end of an episode if the current state is s, the agent executes a and follows learnt policy $\boldsymbol{\pi}_\theta$.

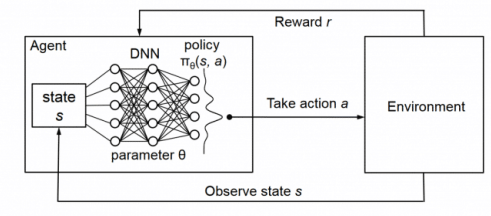


Figure 4.1. Deep Q Learning Environment

In a deep Q learning system, a deep neural network, referred to as DQN, is used to determine the optimal Q function using a temporal difference equation defined as follows:

$$Q_t(s, a) = Q_{t-1}(s, a) + \alpha[R(s, a)$$
$$+\gamma Max_a\{Q_t(s, a)\} - Q_{t-1}(s, a)]$$

where Q_t is the current estimation of the Q function, Q_{t-1} is the previous estimation of Q after $t-1$ steps of training. The estimated next state and action are denoted by s and a, respectively. The learning rate (α) determines to what extent newly acquired information

overrides old information. The discount factor (γ) determines the importance of future rewards.

4.2 System Model and Problem Formulation

The need for intelligent SDN controllers arises in several scenarios. One scenario is the rate control of traffic from different users within a data center or cloud network environment, where the behaviour is controlled by means of a centralized controller, as typical in SDN architectures. The controller needs to dynamically determine how to handle each traffic flow, and determine the per-flow rate limits according to the server capacity and QoS requirements. Another scenario is related to a telecommunications or Internet service provider. The service provider is responsible for accepting packets from connected clients, and route them to the appropriate egress points. The bandwidth rates per user are dynamically determined according to the user subscription service level agreement and to maximize overall network throughput. A third scenario is related to the 5G cellular networks, where network protocols are designed to implement their control and data planes using SDN technologies. In these cases, the control plane needs to determine the rate of traffic from a cellular device, depending on various factors like user channel quality, whether a mobile edge computing server is being leveraged, or packets are being forwarded to an alternative location for processing. While the protocols are different than in a data center or the Internet, the operational paradigm is very similar. In most of these environments, the SDN controller needs to take decisions based on the inspection of the headers within the IP network packets. Therefore, we focus on approaches that learn optimal flow rate-control policies based on the inspection of features extracted from network packet headers.

In general terms, the problem we want to address in this chapter using ML techniques is the design of an intelligent security-aware SDN controller. The controller is responsible for dealing with network flows from several users. Depending on the amount of active flows, anticipated traffic in the near future, and the properties of the current flows (such as potential attacks), the controller configures the optimal rate limits for each flow. In what follows, we

first define the state model underlying our approach. We then formulate our problem and discuss key challenges.

4.2.1 System Model

State Model: We define the state of the SDN environment as a tuple $S_t = {s_1}^t, {s_2}^t, ..., {s_n}^t$ of network flows at time t. The SDN controller manages n network flows from η users. In the current implementation, we assume that $n = \eta$ at all times for ease of execution in the simulation. The flow state ${s_\mathrm{i}}^t$ of user i at time t is defined in terms of k network flow parameters ${s_\mathrm{i}}^t = {P_{1_\mathrm{i}}}^t, {P_{2_\mathrm{i}}}^t, ...{P_{k_\mathrm{i}}}^t$. These parameters can take numeric values, such as packet counts, packet length, packet rate, inter arrival times, or categorical values, such as whether SYN/FIN/ACK flags are set or not, and protocol type.

State Transition Model: We monitor state transitions in the network in terms of "episodes". We define two configuration parameters for an episode: time period T and interval I. The state transitions occur every I time-units until the timestamp reaches T time units, after which the state is reset to the initial state and marks the end of an episode. An episode basically consists of T/I *time instances* at which the state transitions of the environment are recorded. For example, in our current implementation, for $\{T, I\} = \{60, 1\}$, the episodes are one minute long with state transitions every second.

Action Space: After each interval, the SDN controller can take various actions corresponding to different rules used to manipulate the flow table as specified in the action set of the Openflow protocol [90]. For example, in our current prototype, the actions correspond to allow/drop a percentage of the packets for each flow. More specifically, an action A_t at time t, is defined as $A_t = {a_1}^t, {a_2}^t, ..., {a_n}^t$. Here, ${a_\mathrm{i}}^t$ is the action taken on user flow i at time t. Each action $a_\mathrm{i} \in [0, 1]$ corresponds to the percentage of packets to drop while allowing the rest. 0 corresponds to allowing all traffic to pass and 1 corresponds to dropping all traffic. It is important to note that dropping the packets is a crude mechanism of rate control and is a commonly implemented method by network switches. However, new techniques which enforce a rate cap by smoothly delaying packets rather than discarding them can be easily

incorporated in our system model. So, in the rest of the chapter, we use the terms dropping and delaying packets interchangeably.

4.2.2 Problem Definition

We formulate the functionality optimization goal of `Jarvis-SDN` as a Markovian decision process (MDP). A MDP is a sequential decision making problem where outcomes are under the control of an agent. The agent's goal in this case is to maximize the functionality as specified by the user/application by choosing a sequence of actions for the upcoming episode of the environment. In our model, functionality requirements defined by the user/application are measured through a reward function. The specified functionality requirement determines the utility ($F()$) gained by the user/application in the environment, which is one part of the reward function. The other part, derives from the security metric of the environment ($D()$). The general structure of the reward function is defined as follows:

$R(S_t, A_t) = (1 - \delta)F(S_t, A_t) + \delta D(S_t, A_t)$

where S_t and A_t are the current state and action at time instance t in the environment. We analyze the effect that manipulating δ has on the overall functionality goals and security of the environment in Section 4.4. The goal of `Jarvis-SDN` is to maximize the cumulative reward at the end of the episode which is a MDP problem defined formally as follows.

Definition 4.2.1. *A MDP consists of a tuple (R, T, I, S_0). R is the reward function; T is the time period; I is the interval; and S_0 is the initial state of the environment. The goal of the agent is to find an execution strategy of actions, which maximizes the total value of R of the next upcoming episode, for the environment in state S_{i} where $0 \leq \mathrm{i} \leq \lceil T/I \rceil$.*

4.2.3 Challenges

There are two key challenges when applying the system model to a SDN environment.

1. Unknown Security Metric $D()$: Quantifying the security metric accurately in a complex and dynamic environment is not trivial. We address this issue in Section 4.3 by developing an efficient security metric using RL based attack signatures. Notice that other security metrics can also be incorporated into our framework.

2. Unknown Security Ratio δ: The value of the ratio to optimally balance functionality vs security depends on the environment and user/application requirements. Finding this value accurately is not trivial. In Section 4.4, we analyse tuning the security ratio δ as a hyper parameter for `Jarvis-SDN` instantiation with the functionality goal of optimal rate control.

4.3 Building Attack Signatures

In this section, we address the first challenge in building accurate security metrics, referred to as 'attack signatures'. Deep packet inspection at wire speed seems impractical with the growing amount of data and size of networks. Instead, flow based features, which can be maintained using counters and meters, and are easily available through standard SDN protocols such as Openflow [90], are less expensive to monitor. Most supervised attack signatures are built from flow parameters similar to the network flow parameters in our states $P_1, P_2, ..P_k$. These parameters are collected from IDS datasets like NSL-KDD [91] and CICIDS[89]. However, these datasets contain features collected for the entirety of the network flows and not for intervals of the flows as in our framework. This inherently puts the IDS trained on those datasets at a disadvantage since it delays detection time because the flows can only be classified as malicious or benign after the attack ends or enough packets of the malicious flow have reached the controller. In contrast, we build and analyze 'partial attack signatures' using features collected after every interval I in the episode of length T rather than the entire flow. In this respect, our attack signatures are only a partial representation of the more comprehensive traditional attack signatures.

4.3.1 Design of an IDS Based on Partial Attack Signatures

We compare four different ML techniques for building attack signatures in order to determine the most suitable one for our framework: Decision Trees (DT), Random Forests (RF), Deep Neural Networks (DNN), and Deep Q-Networks (DQN). The first three are feature-based classification models that predict benign or malicious flows. The DQN approach, on the other hand, learns optimal quality values using deep Q-Learning on replayed episodes. We model this DQN framework in the same state-action model of our optimization problem.

However, the reward function is modified such that the agent receives negative rewards for allowing malicious traffic and positive rewards for allowing benign traffic depending on the hyper parameters $\beta, \mu \in [0, 1]$. Here, μ and β represent the importance of reducing false positives (Type 2 errors) and false negatives (Type 1 errors), respectively. These rewards are proportional to the amount of traffic let through. The state-action-reward model is defined as follows:

State Model: The state of a user i at time instance t, ${s_\mathrm{i}}^t = {P_{1\mathrm{i}}}^t, {P_{2\mathrm{i}}}^t, {P_{3\mathrm{i}}}^t, {P_{4\mathrm{i}}}^t$. Here, P_{1-4} represent the following features, respectively: P_1: #packets sent from user to server, P_2: #bytes sent from user to server, P_3: #packets sent from server to user, and P_4: #bytes sent from server to user.

Actions: After every interval, the controller can take 11 possible actions $a_\mathrm{i} \in \{0, 0.1, 0.2, .., 1\}$ corresponding to percentage of packets to drop while allowing the rest. 0 corresponds to allowing all traffic to pass and 1 corresponds to dropping all traffic. We discretize the action space to allow better convergence of the DQN.

Rewards: The reward function is defined as follows.

$$R({s_\mathrm{i}}^t, {a_\mathrm{i}}^t) = \begin{cases} (1 - {a_\mathrm{i}}^t) * ({P_{2\mathrm{i}}}^t) * \beta & \text{if benign episode} \\ -(1 - {a_\mathrm{i}}^t) * ({P_{2\mathrm{i}}}^t) * \mu & \text{if malicious episode} \end{cases}$$

We build a simulation environment using the OpenAI gym [65]. For training the DQN, we replay 20177 malicious and 126714 benign episodes from the CICIDS dataset [89]. During replay of the episodes, we simulate packet drops by using a state transition function Δ, manually configured according to the network flow features used in the model. The exploration factor of $\epsilon = 1$ with a decay rate 0.995 is used to balance exploration (random actions) and exploitation (using learnt policy) while training. A discount factor $\gamma = 0.95$ and learning rate $\alpha = 0.001$ are used. The deep neural network used has 2 fully connected hidden layers comprising of 64 neurons each.

Table 4.1. Attack Taxonomy

Attack Type	Total Packet Capture Size (Gb)	Categories
Brute Force	11	FTP-Patator, SSH-Patator
DoS	13.4	SlowLoris, Hulk, GoldenEye, SlowHTTPtest
DDoS	8.8	DDoS LOIT
Web	8.3	XSS, SQL Injection, Brute Force

4.3.2 Evaluation Metrics

For our analysis, we consider four common attacks, Brute Force, DoS, DDoS (Distributed Denial of Service) and Web-based (see Table 5.2), taken from the CICIDS dataset. This dataset has full packet captures of attacks and benign behavior recorded over five days. We define four key performance metrics as follows.

Naive Accuracy: It measures the accuracy of detecting known attacks similar to the ones in the training dataset.

Robustness: Features collected from networks flows in real time tend to be noisy because of network errors, packet drops, jitter, throughput throttling, user behavior etc. Introduction of noise or perturbations in the immutable features is a typical obfuscation technique employed in adversarial ML attacks. The attack signatures must be robust enough to deal with noisy data. So, white Gaussian noise is introduced into the testing dataset. The resulting accuracy is measured as a function of the noise introduced.

Adaptability: It measures the ability of the system to detect new types of attacks. We consider two forms of adaptability: (1) known attacks with minor modifications, e.g. payload differences (string changes, varying malware bytecode), and packet fragmentation variations; (2) new attacks employing similar concepts. Specifically, we exclude FTP-Patator (brute force), SlowLoris (DoS) and SlowHTTPTest (DoS) attacks during training and use them for testing. The resulting accuracy in these scenarios is used to represent the adaptability metric for the attack signatures.

Episode metrics: Instead of detecting malicious traffic on a per interval basis, we attempt to determine whether any malicious interval exists over the range of the episode for unknown attacks under noisy conditions. We define three *episode metrics*: (1) *Episode Accuracy*: Overall accuracy; (2) *Episode True Positive Rate (TPR)*: Rate of correctly identifying malicious episodes; and (3) *Episode False Positive Rate (FPR)*: Rate of incorrectly identifying benign episodes as malicious.

Table 4.2. Attack Signature Analysis

Signature Type	Naive Accuracy	Robustness Noise(low-high)	Adaptability	Episode Metrics (Accuracy, TPR, FPR)			Quality Value
DT	99.92	70.58-26.60	53.22	44.88	100	99.2	No
RF	99.93	80.14-24.83	53.40	44.88	100	99.2	No
DNN	97.22	95.56-80.24	70.28	59.55	100	72.8	Yes
DQN	88.27	79.53-71.21	59.48	73.77	94	42.4	Yes

4.3.3 Comparison to other ML Techniques

Results from our evaluation are shown in Table 4.2. We make the following observations on those results.

Naive Accuracy: All signatures work well in terms of naive accuracy for known attacks. The tree based signatures (DT/RF) perform slightly better than neural network based signatures (DNN/DQN).

Robustness: Performance of all signatures degrades with the amount of noise added. Comparatively, neural network based signatures are less affected.

Adaptability: All algorithms have decreased accuracy when trying to identify unknown and modified attacks. The highest accuracy, 70.28%, is achieved by DNN based signatures.

As we observe, all the attack signatures perform poorly when dealing with unknown or zero-day attacks in terms of per-interval accuracy. We can infer that these unknown malicious flows resemble behavior of both known malicious and benign flows in many intervals of an episode. So, building attack signatures based on episodes rather than individual intervals is a more suitable approach as we see when analyzing the *episode metrics*. It is important to note here that unsupervised learning techniques, like anomaly detection on benign behavior

(clustering, SVMs), could be a better approach for identifying unknown attacks. Although these techniques could be incorporated into our framework, in this work, we only focus on supervised learning and leave that as future work.

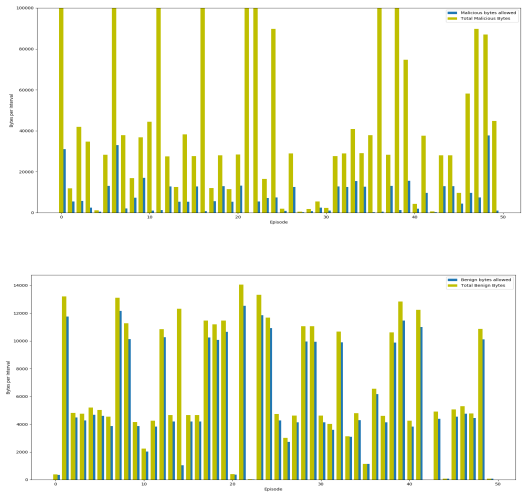


Figure 4.2. Bytes allowed: Malicious (top) and Benign (bottom)

Episode metrics: DQN attack signatures outperform the other signatures in terms of *Episode* metrics even though they perform poorly in terms of per interval accuracy. One reason is that the DQN signatures are learnt from a cumulative reward by replaying attack ranges rather than individual intervals. The other reason is the structure of the reward function, which gives a higher negative reward for false positives than false negatives over the episode range.

The classification based signatures (DT/RF) perform badly for the episode metrics because they learn optimal discrete values (benign/malicious) rather than smooth quality values. We can see that, although these signatures are able to correctly identify all the malicious episodes (*episode TPR* = 100%), they also result in a high *FPR = 99.2%*. Such overly strict signatures are unacceptable as a high percentage of benign flows end up being dropped. On the other hand, the DQN agent learns quality values $Q(S_t, A_t)$ for each interval within the episode. We can then build an intelligent policy based on quality values that minimizes the *episode FPR* and simultaneously maintains a good *episode TPR* by tuning the hyperparameters β and μ. We infer that DQN based agents perform better for new attacks in terms of false positives. Note that DNN signatures can also generate quality values using

state values $V(S_t)$ from a final softmax layer. However, these quality values do not incorporate the state and actions dynamics of the model. Therefore, using these quality values as metrics in a RL framework for functionality optimization results in random quality values being assigned to new actions. This results in poor performance of the RL agent (see Section 4.4.6).

4.3.4 Evaluation and Analysis

To analyse the effectiveness of RL based (DQN) attack signatures, we randomly choose 1000 malicious and benign episodes, and apply the policy learnt to them. We monitor the amount of malicious bytes let through in case of malicious episodes and the amount of benign bytes let through for benign episodes as shown in Figure 4.2. We see that the attack signatures show the desired behavior in all the episodes. It is important to note that these attack signatures have been built using a minimal set of observable states at the SDN controller. With the recent advances in programmable switches and data plane programming languages, like P4 [92], detailed information about packets and their headers is accessible to the controller to make even better security decisions.

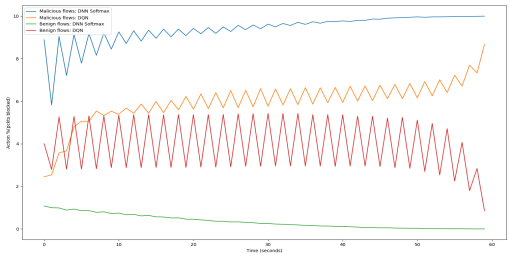


Figure 4.3. Intra-Episode Actions: DQN and DNN Softmax

To further investigate the behavior of the RL based attack signatures, we analyze the actions taken by the RL agent in the episode itself. Figure 4.3 shows the behavior averaged over 1000 malicious and benign episodes. This is desirable behavior as malicious flows get throttled while benign flows are not. Also, we see that at the beginning of the episodes, the RL agent is not sure whether the flow is malicious or not. In such situations, we see that the RL agent exhibits the following desirable behavior. Malicious looking flows are throttled until the agent is sure they are an attack and then are blocked completely. Benign looking

flows are throttled until the agent is sure they are benign and then are allowed. We compare the behavior with the policy based on state values obtained from the softmax layer of the DNN classifier. Surprisingly, the softmax layer approach also shows these same desirable properties and performs better than the RL based signatures. This could be attributed to the fact that since no other functionality optimizations ($F() = 0$) are required during this analysis, the agent does not fully explore the action space and thus the state-action dynamics of the model have little impact on the optimal policy. However, as we see later in Section 4.4.6, the softmax based signatures do not perform well when other functionality optimizations are needed.

4.4 Instantiation of `Jarvis-SDN`

In this section, we instantiate our framework with the goal of optimal rate control in our simulated environment.

4.4.1 Implementation Details

We have built a prototype of `Jarvis-SDN` and integrated it with a network emulated in Mininet [93]. The emulated network includes benign and malicious users accessing a web server. `Jarvis-SDN` sits in the control plane of the SDN controller. It is able to access counters maintained for each flow in the flow table of the SDN controller similar to architectures like [94]. Information from these counters is then used to monitor the state of each flow in terms of the flow parameters $P_1 - P_4$. Similar to approaches discussed in [95], the actions specified by the RL agent are converted into per flow bandwidth limits.

4.4.2 Simulation Environment

The simulated environment consists of n users accessing a web server in a SDN enabled environment. `Jarvis-SDN` running at the SDN controller is responsible for taking optimal rate control actions on each flow so as to maximize two functionalities: ❶ Rate control for the server according to a maximum allowed server load threshold: $SLT \sim \mathcal{N}(\mu_1, {\sigma_1}^2)$; ❷ User throughput fairness according to user service level agreements: $SLA \sim \mathcal{N}(\mu_2, {\sigma_2}^2)$. We use

standard Gaussian distributions to model the server load threshold and user SLA per time interval. The mean μ_1, μ_2 and standard deviation σ_1, σ_2 are chosen manually by observing the server threshold and benign flows in the simulation. Along with this, we maintain δ (or d), f_1 and f_2 as hyper parameters to incorporate security metrics, rate control and user fairness weights in the objective function, respectively.

4.4.3 System Model

The state and action model of `Jarvis-SDN` are the same as the ones defined before for building attack signatures. However, the reward function is modified to include optimal rate control functionality metrics and security metrics from RL based attack signatures. The reward function is defined as follows.

$$R(S_t, A_t) = (1 - \delta)[f_1 F_1(S_t, A_t) + f_2 F_2(S_t, A_t)] + \delta D(S_t, A_t)$$

The normalized functionality and security metrics used in the reward function are defined as follows. We observe that, unlike the case when building attack signatures, here the rewards are expressed as expectations over all the users flows in the network.

Rate Control: It gives a positive reward proportional to the amount of traffic allowed when the current load is less than the threshold and a large negative reward C_1 for overloading the server. The positive rewards are proportional to the amount of traffic let through, which encourages higher throughput.

$$\begin{aligned} F_1(S_t, A_t) &= \mathop{\mathbb{E}}_{\mathrm{i} \in \eta} \left[F_1({s_\mathrm{i}}^t, {a_\mathrm{i}}^t) \right] \\ &= \begin{cases} \mathbb{E}_{\mathrm{i} \in \eta} \left[(1 - {a_\mathrm{i}}^t) * {P_{2\mathrm{i}}}^t \right] & \{SLT(t) > (1 - {a_\mathrm{i}}^t) * {P_{2\mathrm{i}}}^t\} \forall \mathrm{i} \in \eta \\ -C_1 & \textit{otherwise.} \end{cases} \end{aligned}$$

User Fairness: It gives a positive reward proportional to the amount of traffic allowed when the service level agreement is upheld and a large negative C_2 otherwise. The values of parameters C_1 and C_2 are chosen to be greater than the maximum throughput values possible in the environment.

$$F_2(S_t, A_t) = \mathop{\mathbb{E}}_{\mathrm{i}\in\eta}\left[F_2({s_\mathrm{i}}^t, {a_\mathrm{i}}^t)\right]$$

$$= \begin{cases} \mathbb{E}_{\mathrm{i}\in\eta}\left[(1-{a_\mathrm{i}}^t) * {P_{2_\mathrm{i}}}^t\right] & \{SLA(t) \leq (1-{a_\mathrm{i}}^t) * {P_{2_\mathrm{i}}}^t\}\forall \mathrm{i} \in \eta \\ -C_2 & \text{otherwise.} \end{cases}$$

Security: It is the quality value generated by RL based attack signatures built before. Here Q is the learnt Q function using the DQN approach. It gives a positive reward when the flow matches a benign flow and a negative reward when it matches a malicious flow.

$$D(S_t, A_t) = \mathop{\mathbb{E}}_{\mathrm{i}\in\eta}\left[D({s_\mathrm{i}}^t, {a_\mathrm{i}}^t)\right] \approx \mathop{\mathbb{E}}_{\mathrm{i}\in\eta}\left[Q({s_\mathrm{i}}^t, {a_\mathrm{i}}^t)\right]$$

4.4.4 Training

We use a DQN approach to learn the optimal policy similar to the attack signature generation model. We conduct experiments with δ values ranging from 0 (no security guarantees) to 0.99 (high security guarantees). For our experiments, we configure the functionality weights $f_1, f_2 = 0.5$ as constants. We refer the reader to our previous work [86] for more details about functionality weights.

4.4.5 Evaluation and Analysis

Figure 4.4 shows the actions (averaged over 1000 episodes) taken in a single malicious and benign episode. We observe that higher values of δ result in better security guarantees for malicious episodes but at the cost of more false positives per interval during benign episodes. We further analyze the action space of the environment in terms of range of δ. We represent the action space (red region) below $\delta = 0$ as the *unsafe action space* of the model. Similarly, the blue region above $\delta = 1$ represents the *safe action space*. The green region represents the *region of interest* with a range of $\delta = [0, 1]$. The optimal action policy of the environment lies in this region.

The analysis of the *region of interest* in the action space allows one to build ethics or logical hypotheticals for exploration of the environment by the RL agent. For example, the RL agent can explore the action space with $\delta > 0.8$, for highly security conscious or

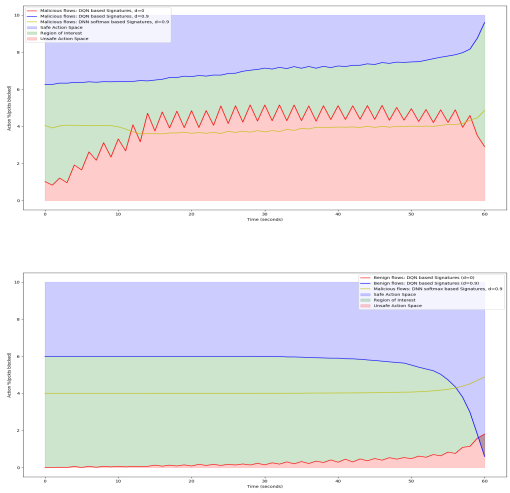


Figure 4.4. Action Space: Malicious (top) and Benign (bottom)

risk-averse ethics. Similarly, with $\delta < 0.2$, the RL agent can explore actions with more adventurous high risk high reward ethics. Learning hypothetical human objectives or ethics to model the reward function or the exploration process is a hard problem because of the intractability of value of information (especially security information) to the user for a given environment. This is the focus of various active learning schemes [71], [72]. The exploration strategy can be dynamically altered based on user/application feedback on these hypothetical behaviors. `Jarvis-SDN` provides an effective way of altering the exploration process by using such hypotheticals along the *region of interest*.

4.4.6 Comparison to Traditional IDS Metrics

Figure 4.4 also shows the actions taken by the RL agent using DNN softmax based security metrics. We see that for both malicious and benign episodes, the RL agent converges to similar actions. This means that these security metrics perform poorly in the detection of malicious flows. This confirms our intuition that when the optimization of multiple functionalities is required, RL agents using softmax based DNNs or traditional IDS security metrics do not converge to secure policies.

4.5 Related Work

As part of previous work we designed and evaluated a framework to constrain RL agents by security and safety policies for IoT-based smart homes [86]. However in a smart home scenario, 'attack signatures' are well defined in terms of device states and can be learnt efficiently by observing anomalies against naturally occurring behavior of users. Such an approach does not work for detecting attacks in networks based on network flows. The reason is that in networks, there are numerous applications for which it is difficult to build an accurate baseline of benign behavior. So, an anomaly based detection is inherently prone to false positives. To address this issue, in this chapter we have used a semi-supervised learning approach to build attack signatures which provide quantifiable security metrics. The other reason is that the state space of a network environment is not limited to discrete device states like in smart homes and thus suffers from the state explosion problem.

Attempts have been made to define quantifiable security metrics [87], [88], [96]. Proposed approaches include building metrics from user or host vulnerabilities, defense strengths, attack (or threat) severity and situation understanding of the environment. To the best of our knowledge, security metrics or attack signatures based on network flow parameters have not been explored. In terms of encoding security metrics in RL frameworks, there are two main approaches: (1) transformation of the optimization criterion, and (2) modification of the exploration process [97]. Our approach falls under the first category as we modify the objective/reward function using the security metrics. But additionally, we also analyze the action space according to the hyper parameter δ, which can be used to inject external knowledge or advice to guide the exploration process of the RL framework. Due to space limitations, we leave the integration of such an approach into Jarvis-SDN as our future work.

4.6 Conclusion and Future Work

In this chapter we have designed and evaluated `Jarvis-SDN`, a RL framework for optimizing network functionalities constrained using RL-based network flow attack signatures. While our initial results show that our framework represents significant progress, we plan to

carry out additional experiments to assess its performance for different network conditions, network functionalities like routing and RL algorithms like policy gradient approaches.

5. JARVIS-SDN: SECURITY CONSTRAINED RL FOR ROUTING IN SDN ENVIRONMENTS

The Traffic Engineering (TE) problem is a fundamental networking problem which deals with performance evaluation and optimization of operational IP networks. However, as networks are growing at an exponential rate in terms of scale and heterogeneity in terms of new functionalities, like internet of things (IoT), protocols, hardware/software platforms etc., the TE problem has become even more challenging. On the other hand, new networking paradigms, like network function virtualization (NFVs) and software defined networks (SDNs), allow for more flexibility and efficiency in learning and deploying intelligent TE policies.

Such additional capabilities further increase the complexity of network management tasks. Therefore approaches have been proposed that leverage machine learning (ML) techniques to automate network management. Such approaches build models of the network architecture and user[1] behavior, and then learn optimal policies using these models. However, such static model-based strategies do not work for dynamic network conditions and requirements. The network policies must evolve on-the-fly to keep up with these dynamic requirements and changes. To this end, several model-free reinforcement learning (RL) methods have been proposed [10]–[12], which are able to learn optimal policies in an online experience-based manner.

In terms of security, the zero trust policy paradigm for enterprises has been recently introduced [98], [99]. Zero trust assumes that there is no implicit trust granted to assets or user accounts based solely on their physical or network location (i.e., local area networks versus the Internet) or based on asset ownership (enterprise or personally owned). Zero trust thus requires that authentication, authorization and other security services be executed before or during a session accessing an enterprise resource [98]. Under such a security paradigm, it is essential for the security policies along with networking policies to be self-evolving also based on the dynamic network conditions and requirements. It is also important to mention that, as modern networks are plagued with various network based attacks like DDoS, DoS, web based attacks etc., intrusion detection and prevention systems (IDS/IPS) (see Ta-

[1]↑By user, we refer to any party communicating over the network, such as applications and end-users.

ble 5.1) have been developed to detect such attacks. In SDN environments, IDS/IPSs are deployed as either physical middle-boxes or virtualized network functions NFVs [100] running on commercial-off the-shelf (COTS) servers. Therefore, the combined use of network IDS/IPS, zero trust architectures, and NFVs should substantially enhance network security in modern SDN environments.

However, existing model-free RL based approaches do not consider network security when determining routing policies. This is an inherent flaw of"intelligent" network controllers. For example, ideally a suspicious network flow, which could be a DDoS attack, should be routed through nodes[2] running a DDoS IDS/IPS on the network. To address such requirement, we design `STE-SDN`, a Smart TE framework enabling optimal and secure routing policies using model-free deep DRL. Specifically, `STE-SDN` routes networks flows intelligently based on two goals: (1) Network functionalities: minimize overall network latency, bandwidth, etc.; and (2) Network security: route suspicious flows through various NFV IDS/IPSs to minimize network security risks. Like previous DRL approaches, performance metrics, used as reward in the environment, for network functionalities can be defined in terms of latency, bandwidth usage, etc. However, metrics for security are not trivial to define. To address such an issue, we define a value system for secure routing in SDN environments based on the placement and type of security services (IDS/IPS) in the network. This value system allows network administrators to encode their domain knowledge of security services into a reward function, which determines the "intelligence" of the DRL routing model. We demonstrate the efficacy of the value system by developing a taxonomy of IDS/IPSs for SDN environments and encoding this domain knowledge into a 'smart' reward function.

To summarize, we make the following contributions:

1. We develop a model-free DRL framework for security aware intelligent routing
2. We define a security metric (value system) that can be used to translate domain knowledge about security services in SDN environments into a reward function for DRL frameworks. We build a taxonomy of IDS/IPSs for SDN environments and encode this knowledge into a 'smart' reward function using the defined value system.

[2]↑By node, we refer to a gateway/access point of an autonomous system (AS) or VLAN network segment

3. We build a simulation environment for testing DRL based routing algorithms on the openai gym[65] using networks flows collected from the CICIDS dataset [89] to model user behavior and network architecture generated using the Inet network topology generator [101].
4. We implement and test a DRL-based routing system for different regions of a 5000 node randomly generated network in the simulation environment for three attack classes: DDoS, Web and Brute-Force.

The rest of the chapter is organized as follows. In Section 5.1, we give some background on DRL, specifically Deep Q-Learning. Next, in Section 5.2, we formally define the system model and discuss key challenges. In Section 5.3, we provide details of our RL based solution. In Section 5.4, we discuss the design details of an instantiation of our framework in a simulated network of 5000 nodes with the goal of minimizing overall network latency and maintaining security. In Section 5.5, we analyze the effectiveness of our framework with respect to performance and security in our simulated network. Finally, we discuss related work in Section 5.6 and in Section 5.7, we outline conclusions and directions for future work.

5.1 Background

A Reinforcement learning (RL) framework [63] (see Figure 5.1) is a probabilistic state transition environment where state transitions are caused by actions executed by an agent and every state-action pair (s, a) is assigned a reward value r given by a reward function $R(s, a)$. A RL agent traverses the environment according to a policy $\boldsymbol{\pi}$ that selects an action to execute in a given state for policy parameters θ and receives the rewards accrued over all the state transitions that hschoappened according to θ. In a Q learning framework, the goal is to find the optimal policy which maximizes the cumulative reward through a function $Q(s, a)$ that estimates the expected cumulative reward the agent will get at the end of an episode if the current state is s, the agent executes a and follows the learnt policy $\boldsymbol{\pi}_\theta$.

In a deep Q RL system, a deep neural network, referred to as DQN, is used to determine the optimal Q function using a temporal difference equation defined as follows:

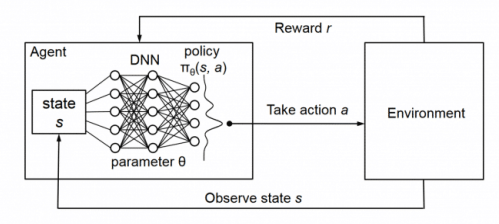


Figure 5.1. Deep Q Learning Environment

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a)$$
$$+\gamma Max_a\{Q_t(s,a)\} - Q_{t-1}(s,a)]$$

where Q_t is the current estimation of the Q function, Q_{t-1} is the previous estimation of Q after $t-1$ steps of training. The estimated next state and action are denoted by s and a, respectively. The learning rate (α) determines to what extent newly acquired information overrides old information. The discount factor (γ) determines the importance of future rewards.

5.2 System Model and Problem Formulation

In general terms, the problem we address in this chapter is the design of an intelligent security-aware SDN controller. The controller is responsible for dealing with network flows between several users. Depending on the network architecture, placement of security services (or NFVs) like IDS/IPSs in the network, and the properties of the network flows (such as potential attacks), the controller dynamically configures the optimal routes for each flow. In what follows, we first define the state model underlying our approach; we then formulate our problem and discuss key challenges.

5.2.1 System Model

State Model: The state $S_t = [P_t, E_t, srcid, destid]$ of our environment consists of 4 components: (1) P_t: flow based features of the flow to be routed at time t; (2) E_t: the current network edge weights along all the edges of the network at time t; (3) source node identi-

fier $srcid$ = [source IP address, source port]; and (4) destination node identifier $destid$ = [destination IP address, destination port].

The state of the flow, P_t, to be routed in the network at time t is defined in terms of k network flow parameters $P_t = {p_1}^t, {p_2}^t, ...{p_k}^t$. These parameters can take numeric values, like packet counts, packet length, packet rate, inter arrival times, etc., or categorical values, like SYN/FIN/ACK flag set or not, protocol type, etc. In our current implementation, we use 80 statistical network traffic features, including duration, number of packets, number of bytes, length of packets, which are defined in the CICIDS-17 dataset[89]. The edge weights $E_t = {E_1}^t, {E_2}^t, ...{E_e}^t$ are defined for each of the e edges in the network. These weights represent the cost of sending packets (in terms of latency) along each edge of the network at time t. The source and destination ids uniquely identify the source and destination points of the new flow to be routed in the network.

State Transition Model: We monitor state transitions in the network in terms of "episodes". We define two configuration parameters for an episode: time period T and interval I. The state transitions occur every I time-units until the timestamp reaches T time units, after which the state is reset to the initial state and marks the end of an episode. During every state transition, a new flow is added in the network between two nodes in the network. The SDN controller manages n network flows from η users. In the current implementation, we assume that $n = \eta$ at all times (one flow per user) for ease of execution in the simulation. An episode basically consists of T/I *time instances* at which the state transitions of the environment are recorded or η number of flows added in the network. For example, in our current implementation, for $\{T, I\} = \{60, 1\}$, the episodes are one minute long with state transitions every second. It essentially means that in an episode of 60 seconds, every second a new flow is added in the network.

Action Space: After each interval, the SDN controller can take various actions corresponding to different routing rules used to manipulate the flow table as specified in the action set of the Openflow protocol [90]. For example, in our current prototype, the actions correspond to first determining a set of candidate paths for each new flow added, and then choosing the optimal one among the candidate paths available based on the edges and their weights in the network. More specifically, given c candidate paths, an action A_t at time

t, takes value in the range $[1, c]$, where 1 corresponds to choosing the first candidate path, 2 corresponds to choosing the second candidate path and so on. It is important to note that choosing a single candidate path is a naive routing mechanism, which is a commonly implemented method by network switches. However, techniques, which allow one to send a particular flow packet through various candidate paths, are also possible. In this case, our model can be easily extended by altering the action space to a continuous vector space such as $A_t = a_0, a_1, ..., a_c$ where $a_{\text{i}} \in [0, 1]$ and $\sum_{\text{i}=0}^{c} a_{\text{i}} = 1$. We define the action space AS as all the possible actions $0 - c$ for discrete actions and $[0, 1] * c$ for continuous actions.

Definition 5.2.1. *An episode is defined as a tuple (N, S_0, T, I), where: $N = \{S_0, S_1, ., S_t, ..S_{T/I}\}$ is an ordered list of states reached in the episode where for each state S_t, action A_t is taken to reach S_{t+1} where $0 < t \leq T/I$; S_0 is the initial state of the episode; T is the time period; and I is the interval.*

5.2.2 Problem Definition

We formulate the functionality optimization goal of `STE-SDN` as a Markovian decision process (MDP). A MDP is a sequential decision making problem where outcomes are under the control of an agent. The agent's goal in this case is to maximize the functionality as specified by the user by choosing a sequence of actions for the upcoming episode of the environment. In our model, functionality requirements defined by the user are measured through a reward function. The specified functionality requirement determines the utility ($F()$) gained by the user in the environment, which is one part of the reward function. The other part derives from the security metric of the environment ($D()$). The general structure of the reward function is defined as follows:

$R(S_t, A_t) = (1 - \delta)F(S_t, A_t) + \delta D(S_t, A_t)$ where S_t and A_t are the current state and action at time instance t in the environment. In Section 5.3, we define the functions $F()$ and $D()$ in more details. The goal of `STE-SDN` is to maximize the cumulative reward at the end of the episode which is a MDP problem defined formally as follows.

Definition 5.2.2. *A MDP consists of a tuple (R, T, I, S_0), where: R is the reward function; T is the time period; I is the interval; and S_0 is the initial state of the environment. The*

goal of the agent is to find an execution strategy of actions, which maximizes the total value of R of the next upcoming episode, for the environment in state S_{i} where $0 \leq \mathrm{i} \leq \lceil T/I \rceil$.

5.2.3 Challenges

There are two key challenges when applying our system model to a SDN environment.
1. State Explosion Problem: We observe that the state of the environment includes the edge weights E_t of the network. The number of edge weights grows at a rate of $O(n^2)$ where n is the number of nodes in the network. For a large network, working with a large state model will likely result in the state explosion problem and thus be computationally infeasible. In this work, we address this issue by modifying the state to include only candidate path weights instead of the edge weights. Along with this, we minimize computational costs by running the candidate path search algorithm in an offline manner instead of a per episode basis. Although this could result in sub-optimal policies being learnt due to non-availability (leakage) of information about some key edge weights (not part of candidate paths), our experimental results reported in Section 5.5.2 show that in practice the learnt policies are close to optimal.
2. Unknown Security Metrics $D()$: Quantifying security metrics accurately in a complex and dynamic environment is not trivial. This problem is compounded by the fact that alerts issued by security services are not quantifiable without proper domain knowledge of the attacks. To address this issue we develop an efficient framework for building such security metrics based on the type of security service and its placement in the network (see Section 5.3.2). Such a framework allows network administrators to incorporate their security domain knowledge into a value system, which is then used to build the dynamic reward function. This reward function acts as a black-box between network users and network administrator's but at the same time allows both of them to configure the value system of the routing environment. To illustrate how the value system would work in practice, we build a taxonomy of security services for SDN environments and then encode this domain knowledge into our value system.

5.3 RL based Solution

In this section, we define our RL based solution to solve our MDP problem. We first discuss how we address the challenges discussed in the previous section and then describe in detail the overall Deep Q-Learning based algorithm.

5.3.1 Localized State Model

To address the state explosion problem, we define the state of the environment as a localized portion of the network rather than for the whole network. That is, instead of processing the weights of all the edges in the entire network, we process only the weights of the edges involved in the candidate paths for each new flow added in the network. Furthermore, we perform the expensive path computations in an offline manner and only use the total path weights (sum of all the edge weights in the path) instead of individual edge weights. c candidate paths are determined a-priori for each node pair in the network using any path finding algorithm before the model is initialized. Specifically, for all the node pairs in the environment, a set of c candidate paths are pre-computed before the learning process begins. So, instead of $E_t = {E_1}^t, {E_2}^t, ...{E_\epsilon}^t$ in the state definition, we use $CP_c = {CP_1}^t, {CP_2}^t, ...{CP_c}^t$. So, the overall size of the state space, SS, decreases from $\epsilon \approx O(n^2)$ to $c \approx O(1)$. Our approach mitigates the state explosion problem and at the same time gives enough localized information to the RL agent to take optimal decisions. It is important to note that choosing a high enough value for c is crucial in order to make sure that the localized view of the network still gives a close to optimal decision. In other words, c should be chosen such that there is enough diversity in the available candidate paths that one of them is close to optimal.

In our current implementation, we use a naive recursive shortest k paths which results in $O(n^k)$ exponential time complexity. Our approach could be improved to $O(n)$ by using a more sophisticated K shortest path algorithm [102]. However, as all these computations are performed offline before the environment is initialized, they do not affect the overall performance of our approach. Also, candidate path finding is not the focus of this work and we argue that any such algorithm can be employed in our framework. Candidate path

finding can be performed in an online manner but this would require a computation cost in ($O(n)$ using [102]) for each episode run, which is why in our implementation all candidate paths are found in an offline manner.

5.3.2 Security Metric Value System

We define a security metric (value system) for DRL frameworks by assigning values to different types of network attack and the corresponding security services that can accurately detect them. This value system enables domain knowledge about security services and the attacks that these services are able to stop to be efficiently translated into a reward function.

We define the value system for κ security services denoted by $SC_1, SC_2, ..., SC_\kappa$. These security services are usually deployed in-line with all the m flows going through the gateway or access point of the autonomous system (AS). If an attack is detected by the security service SC_i for a flow $\text{j} \in [0, m)$, in general an alert $A_{SC_\text{i},\text{j}}$ is generated by the security service. The alert $A_{SC_\text{i},\text{j}}$ from the security service SC_i could either be a binary (attack/benign) flag ($A_{SC_\text{i},\text{j}} \in \{1, 0\}$), or in more sophisticated systems a confidence value ($A_{SC_\text{i},\text{j}} \in [0, 1]$). By using these generated alerts for a particular flow, a dynamic smart reward function R_{smart} is defined to generate a reward value which can then be used by the RL agent for learning.

For each type of security service SC_i, we define a value system parameter $v_\text{i} = \{\omega_1 p_\text{i} + \omega_2 cv_\text{i}\}/2$ which is a weighted sum of two key hyper-parameters: (1) Priority $p_\text{i} \in [0, 1]$: it determines the importance of the alert for that corresponding type of attack in the environment; and (2) confidence $cv_\text{i} \in [0, 1]$: it is the confidence value for the reported alert by the security service. For example, detecting a DDoS attack might be at higher priority in the environment than a passive sniffing attack. On some networks, a DoS or DDoS attack could have higher priority than Web based or FTP brute force attacks as the web/FTP servers are already hardened against such attacks. The confidence values cv_i can highly vary and depend on the design of the security service SC_i (see Table 5.1). Both these values require fine-tuning by network administrators based on the requirements of the environment. However, the confidence value is generally pre-defined for the security service in advance, so it stays constant across different environments and does not require much tuning.

Table 5.1. Taxonomy of Security Services in SDN Environments

Security Service (SC_i)	Confidence (cv_i)	Attacks Covered	Alert Type (A_{SC})	Detection Method	Application Area/ Key Features
[103]	Low (0.1-0.3)	DoS, Port-Scanning	Binary (1/0)	Anomaly-Based	SOHO home/office networks
[104]	Low (0.1-0.2)	DDoS, Flooding Attacks	Binary (1/0)	Network-Flow Statistics based using unsupervised self organizing maps (SOMs)	Lightweight on NoX controller
[105]	Medium (0.2-0.5)	DDoS	Binary (1/0)	Anomaly-Based using Deep Learning	SDN environments
[106]	Medium (0.2-0.5)	DDoS	Binary (1/0)	Statistics based using stacked autoencoder (SAE)	SDN environments
[107]	Medium(0.3-0.7)	DDoS, DoS, Probe, U2R, U2L, Web based attacks	Confidence Value	Statistics based Advanced ANN using SOM, M-SOM, Learning Vector Quantization (LVQ1), HLVQ1	SDN Opendaylight Controller
Kalis [108]	High (0.7-1)	Network Attacks: Selective Forwarding, wormhole, blackhole, jamming	Signature Confidence Value	Signature/Rule Based	IoT Environments
Heimdall [26]	Medium (0.4-0.8)	DDoS, Web based attacks, DNS spoofing	Binary (1/0)	White-List based (IP, Domain name)	IoT Environments
[109]	High (0.7-1)	DDoS, IP Spoofing, Host Location Hijacking, Flow Table Overloading, Flooding Attacks	Confidence Value [0,1] 5-Tuple from 5 layers	ML based 5 Layers: User Authentication, Deep RL based game theory approach, Shannon Entropy, GM-SOM, Location based analysis	5G Environment with NFVs
[110]	High (0.5-1)	network scanning, OpenFlow flooding, switch compromised attacks, ARP attacks in both data plane and control plane	Confidence Value [0,1]	Multi Observation Hidden Markov Model (HMM) on SDN environment features	SDN Environment with Ryu Controller and Openflow Switch
[111]	Medium (0.5-0.8)	DDoS	Binary (1/0)	Kohonen neural network updated with fuzzy logic rules (FSOMDM)	SDN based Cloud Computing Environment
Snort, Bro, L7-Linux-Filter	Broad (0.1-1)	DoS, DDoS, Brute-Force, Web based etc.	Priority Alert (1-10)	Signature/Rule based	Classical and SDN Environments
[112]	Low (0.1-0.4)	Insider Attacks	Trust Value [0,1]	Trust Computation using Bayesian Inference	Health Care/IoT based SDN Environment
[113]	High (0.7-1)	DoS, DDoS, Brute-Force, Web based etc.	Priority of Attack Signature [0,1]	Signature based Parallelized Deep Packet Inspection (DPI) Module	SDN Environment
[114]	High (0.6-0.9)	DoS	Priority of Attack Signature [0,1]	Selective DPI	Sensory Modules and modular design reduced cost of operation in SDN Environments
[115]	High (0.8-1)	Application Based DoS, DDoS, Brute-Force, Web based etc.	Priority of Attack Signature [0,1]	DPI of application Signature data	Analysis of JSONs of user and application flows in SDN Environments
BlindBox, DPIEnc, [116]	High (0.7-1)	Application Based DoS, DDoS, Brute-Force, Web based etc.	Priority of Attack Signature [0,1]	DPI on Encrypted Traffic	Obfuscated Rule Encryption and Probable Cause Decryption on SDN Environments

To better illustrate the value system, we provide in Table 5.1 a taxonomy of network security services proposed in the research literature. We assign both a qualitative and quantitative confidence range to those services in the context of SDN environments based on our domain knowledge of these services. For instance, signature based IDS/IPS have a higher confidence value than anomaly based ones as the latter are more prone to false positives. Services which employ deep packet inspection (DPI) have a higher confidence value as compared to ones using statistical or machine learning techniques over network flows. Such a confidence range would serve as a guideline to network administrators for choosing an appropriate confidence value for each security service and fine-tuning it according to network requirements. We also observe that some services provide a binary alert while others provide a more fine-grained alert priority/confidence value. It is important to note that such confidence values for alerts A_{SC} are specific to each service and are different from the other parameters defined for the value system. Notice that our value system is parametric with

respect to the considered services and the confidence and priority values assigned to these services. Therefore, our framework can be customized to specific settings by including specific services and fine-tuning their confidence values. Finally, for a flow j, routed through a path which contains a set of q security services $SC_{\mathrm{i}1}, SC_{\mathrm{i}2}, ..., SC_{\mathrm{i}q}$ on one of the intermediary ASs, the security metric $D(\mathrm{j})$ is defined as:

$$D(\mathrm{j}) = A_{SC_{\mathrm{i}1},\mathrm{j}} v_{\mathrm{i}1} + A_{SC_{\mathrm{i}2},\mathrm{j}} v_{\mathrm{i}2} + ... + A_{SC_{\mathrm{i}q},\mathrm{j}} v_{\mathrm{i}q}$$

The functionality metric of the environment $F(\mathrm{j})$ can be defined in terms of overall latency, bandwidth, user satisfaction etc. in the network. In our current implementation, we define $F(\mathrm{j})$ as the time taken or latency (scaled to 1) for the completion of each new flow j added in the network. Finally, by combining both the functionality and security metrics, we define the smart reward function of the framework for each new flow added as follows:

$$R_{smart}(\mathrm{j}) = (1 - \delta)F(\mathrm{j}) + \delta D(\mathrm{j})$$

It is important to note that we do no consider the computational or communication cost of the operation of these security services in the reward function in terms of latency, bandwidth, etc. This could be introduced as another hyper-parameter to account for operation costs similar to priority and confidence values. However, in our current framework, we maintain such operation cost parameters as trainable parameters rather than hyper-parameters. Note that the functionality metrics part of the rewards provides enough information about functionality gains such as latency, bandwidth etc. for these trainable parameters to learn about cost of operation of the security services.

5.3.3 Q-Learning Algorithm

We now describe our RL based Deep Q-Learning approach to find the optimal quality function for the environment in terms of the reward function R_{smart} as detailed in Algorithm 3. We build a simulated environment where an agent can run multiple episodes to find the optimal and most secure routing path for upcoming episodes. The agent balances

exploration and exploitation according to the exploration rate ϵ. The $ArgMax(Q)$ function returns the highest quality action or candidate path for the given state. Random batches from the agents prior experiences are selected and replayed to learn cumulative rewards according to the discount factor γ and batch size $Bsize$. Finally, the random batch with cumulative rewards is used to train a Deep Neural Network (DNN) in order learn optimal Q function values for each state action pair of the episode. The DNN used has 2 fully connected hidden layers comprising of 128 neurons each. The exploration factor of $\epsilon = 1$ with a decay rate 0.995 is used to balance exploration (random actions) and exploitation (using learnt policy) while training. A discount factor $\gamma = 0.95$ and learning rate $\alpha = 0.001$ are used.

It is important to note that our RL framework can employ any based RL based algorithm like DDPG [117], Dueling DQN [118], DDQN [119], A3C [120], etc. In our prototype, we use the naive DQN algorithm as it is the fastest to train and gives close to optimal results in our environment. Also, the RL algorithm itself is not the focus of our work; rather our goal is to build a robust RL framework that can effectively employ any of these algorithms depending on the requirements and constraints of the environment/network.

5.4 Design Details

In this section, we describe how we instantiate our RL framework to solve the traffic engineering problem in a SDN. To that end, we design a simulation environment on the OpenAI Gym [65]. In order to simulate a realistic environment, we define three design goals, namely:

1. Realistic network topology.
2. Networking patterns similar to malicious and benign traffic.
3. Realistic user behavior/mobility patterns.

We design a modular framework consisting of 3 modules, each with a different function. The first module is the the *network topology generator module*, responsible for generating a realistic network topology. The second module is the *network flow database*, essentially a store of pre-collected network flows that realistically emulate typical networking patterns for both malicious and benign traffic. The third module is the *user behavior module* that deter-

mines the policies to simulate realistic user behavior and mobility patterns. These modules interact with each other through a central *Episode Module* that is responsible for effectively combining information from them to generate episodes in the simulation environment.

In what follows we describe each module in more detail and then describe the overall workflow of the RL framework.

5.4.1 Network Topology Generator Module

We define the network topology of the environment in terms of a directed weighted graph where the nodes represent autonomous systems (AS) and the weights represent the cost of packet transfer (dependent on latency, bandwidth, etc.) on that edge. The network topology for the simulated environment can be generated using two approaches. The first is to use a known network topology, such as ARPANET [121], NSF Network [122] etc. The problem with this approach is that these topologies are static, meaning that we cannot carry out multiple experiments with different network configurations. Also, these topologies are outdated and unreliable. The other approach, which we adopt, is to use synthetic topology generation tools, like Inet [101] and BRITE [123]. This approach allows us to carry multiple experiments on different network configurations, which are more reliable and up to date.

Specifically, in our current prototype, we use the Inet [101] topology generator, which generates an AS level representation of the Internet. It generates random networks with characteristics similar to those of the Internet from November 1997 to Feb 2002, and beyond. We use it to generate a network of 5000 nodes, which is the approximate number of ASs on the Internet. In terms of security services, we randomly distribute 100 different security services from our taxonomy (see Table 5.1) across different ASs in the network. We then perform experiments on randomly chosen localized regions in this large network, to represent different parts of the Internet (see Section 5.5 for more details). Although we used Inet based network topologies for our experiments, it is important to note that any static known network topology or some other topology generator tools can be used for topology generation in our framework.

5.4.2 Network Flow Database

There are numerous public datasets available of network flow features for both malicious and benign traffic, like NSL-KDD [91], CICIDS17-18 [89], UNB-ISCX [124] etc. So, the network flow database used in our framework collects flows from these existing datasets in an offline manner. Also, flow features can be collected in an online manner in the network by using a network sniffer at gateways. The online approach is useful for collecting unique benign flows in the network. The offline approach collects both benign and malicious flows. For our experiments, we use the CICIDS-17 dataset, which profiles the abstract behavior of human interactions and generates naturalistic benign background traffic of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols [89]. For malicious flows, we consider three common attacks, that is, Brute Force, DDoS (Distributed Denial of Service) and Web-based (see Table 5.2).

Table 5.2. Attack Taxonomy

Attack Type	Total Packet Capture Size (Gb)	Categories
Brute Force	11	FTP-Patator, SSH-Patator
DDoS	8.8	DDoS LOIT, ARES, Port Scans
Web	8.3	XSS, SQL Injection, Brute Force, Heartbleed

5.4.3 User Behavior Module

By user behavior, we mean we refer to which user communicates with who and when in the network. To this end, several prior research efforts [125] have tried to build realistic communication patterns in networks modelled using recorded user communication patterns in real networks. One approach could have been to use one of these communication models in our framework. However, the goal of our work is not only to accommodate existing communication patterns but to also show that our framework is 'intelligent' enough to deal with any new communication pattern that might occur. To this end, we use a randomized net-

work communication model. Specifically, two nodes (source and destination) are randomly chosen in the network and then a network flow is randomly chosen from the network flow database between these two nodes. We use this procedure for each *instance* of the episode.

The episode module is responsible for collecting T/I *instances* using the user behavior module. Since the source and destination nodes of the flows are randomized, the candidate paths for every source-destination pair must be pre-computed offline as discussed in Subsection 5.3.1. To do this, we maintain the offline *candidate path generator module* which pre-determines the candidate paths for each source-destination pair in the network.

5.4.4 Simulation Environment

Using the data from the episode module, the simulation environment is built where at every I times, a new flow is added in the environment for a time period T as defined in Subsection 5.2.1. At every instance I, the controller takes a routing decision based on the routing policy. All *(state, action, next state, reward, next state)* records are stored in the replay buffer which is used by the RL learning algorithm to optimize the routing policy. Figure 5.2 provides an overview of all components of the simulation environment.

5.5 Evaluation Results and Analysis

In this section, we analyze the security and performance of our framework for three attack scenarios: brute force, DDoS and web based from the CICIDS-17 dataset.

5.5.1 Security Analysis

For the security analysis, we compare the overall detection loss (scaled to 1) for random episodes in the simulated environment with `STE-SDN` and without. The results are reported in Fig. 5.3, 5.4 and 5.4 for each scenario, respectively. By detection loss, we refer to the risk metric part ($D()$) of the reward function, which is proportional to the amount of malicious packets that were routed through some part of the network which has no security services for this type of attack or has security services that are less suited for this kind of attack. The goal of `STE-SDN` is to minimize the detection loss in the network and we see that this is

the case for all three scenarios as shown by the results in the figures. We infer that `STE-SDN` is security aware and reduces security risk on average in all three scenarios.

5.5.2 Performance Analysis

For performance analysis, we compare the overall latency (scaled to 1) for training episodes in the simulated environment with `STE-SDN` and without. The results are reported in Fig. 5.6, 5.7 and 5.8 for each scenario, respectively. By latency loss, we refer to the functionality gain part of the reward function ($F()$), which is proportional to the latency taken for routing all the flows in the episode.

We first observe that the latency loss reaches the optimal value in around 50 training episodes. We see that when using `STE-SDN`, the optimal latency loss achieved is higher than the case when `STE-SDN` is not used. This is the cost we need to pay for better security in the network. However, we see that for many of the episodes this cost is not that high as alternate secure candidate paths found by the RL agent have similar latency to the optimal insecure candidate paths. We can infer that, as the functionality reward ($F()$) is also part of the reward function, `STE-SDN` is able to "intelligently" optimize the latency or functionality for these episodes but not at the cost of security.

We also see that there exists a reward differential (green space between orange and blue plots) in both functionality gain (latency loss) and security risk mitigation (detection loss) when `STE-SDN` is used and when it is not. We refer to the state-action space of these reward differentials as *unsafe performance state-action space* and *unsafe risk state-action space*, respectively (similar to [86], [126]). By exploring these *unsafe state-action space*, the risk (security risk) and functionality gain (network latency) trade-off can be managed by adjusting the reward function hyper-parameters (see Subsection 5.3.2). The values of these parameters directly impact how the "intelligence" of the network controllers is defined.

We also observe that the *unsafe risk state-action space* is clearly defined for most episodes, while the *unsafe performance state-action space* is less clear (for some episodes where orange and blue plots converge, it does not exist). From this we can infer that exploring the *unsafe risk state-action space* is more feasible in our framework because the rewards are more

structured due to our value system (see Subsection 5.3.2). Through careful exploration of this space and by fine-tuning the parameters in the value system, network administrators would be able to develop intelligent and specialized TE policies in SDN environments.

5.6 Related Work

The TE problem has been widely investigated and several approaches to solve it have been proposed. The most common approach is to use link state routing protocols, like Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), etc. These approaches are more geared towards simplicity rather than optimality. Later, Xu et al. [127] proposed PEFT, a link state routing with traffic splitting over multiple paths and an exponential penalty on longer paths to achieve optimal routing solutions. In the context of SDN, Jain and et al. [128] proposed a TE solution using a bandwidth function for data transmissions among Google's data centers. Agarwal et al. [129] proposed approximate TE solutions for SDN environments. Due to the scale and dynamic nature of modern day networks, these model based solutions are not effective. Recently, deep model-free RL based experience driven approaches [10]–[12] have been proposed for dynamic and large scale networks. However in contrast to our work, these DRL models do not consider network security.

To the best of our knowledge, ours is the first RL based TE framework that uses knowledge about location and details of security services in the network. Approaches [130]–[132] have been proposed to address the problem of security services placement in networks to minimize security risks. These approaches are orthogonal to our work. Whereas their goal is to find an optimal placement strategy for security services, our goal is to find an optimal TE strategy to optimize functionality and minimize risk using knowledge about security services available and their placements in the network.

Approaches have also been proposed to deal with the congestion control aspect of TE, such as the one by Li et al. [133] that deals with managing sensitive/insensitive traffic by adding explicit delay terms to the utility function measuring QoS. More recently, deep RL based approaches, like the one by Huang et al. [14], have been proposed to solve the

congestion control problem. Previous work [126] on rate control focuses on encoding security metrics into the RL framework to deal with network based attacks while at the same time optimizing rates. However, in contrast to our work, these approaches do not consider the routing aspect of the TE problem.

5.7 Conclusion and Future Work

In this chapter, we have proposed a DRL based "intelligent" TE framework for SDNs able to optimize both network functionality and security. Our framework focuses primarily on the routing aspect of TE. As part of future work, we intend to integrate techniques [126] for optimal network rate control with the techniques presented in this chapter for end-to-end routing, in order to build a complete DRL framework for optimal and secure TE for SDN environments.

Algorithm 3: Learning Optimal State Action Quality

Input: Simulated Environment Env with n nodes; Estimated Reward Function $R_{smart}() : \{SS, AS\} \rightarrow \mathbb{Z}$; Maximum Episodes EP; Exploration (Rate , Min, Decay) $= (\epsilon, \epsilon_{min}, \epsilon_{decay})$; Batch Size $BSize$; Discount Rate γ; Preferable Loss L_p

Output: State Action Quality Function $Q : \{SS\} \rightarrow AS$

Initialize: $Q[:,:] = 0$

Offline Candidate Path Algo:

$\forall (src, dst) \in [0, n) \rightarrow \{{CP_1}^t, {CP_2}^t, ...{CP_c}^t\}_{(src,dst)}$; Encode candidate paths into SS and AS

while $ep \leftarrow 0 : EP$ **do**

- $S_{curr} = Env.Init()$
- $S_{next} = S_{curr}$
- **while** $t \leftarrow 0 : T$ **do**
  - **if** $Random() \leq \epsilon$ **then**
    - $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$ ▷ Exploration: $A_{curr} = Random(AS)$
  - **else**
    - $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$ ▷ Exploitation: $A_{curr} = ArgMax(Q[S_{curr}])$
  - $R_{curr} = R_{smart}(S_{curr}, A_{curr}, t)$
  - $Mem \leftarrow (S_{curr}, A_{curr}, R_{curr}, t)$
  - **if** $Mem > Bsize$ **then**
    - $Q = Replay(Bsize)$
  - $t \leftarrow t + I$
  - $S_{curr} \leftarrow S_{next}$
- $ep \leftarrow ep + 1$

procedure REPLAY($Bsize$, L_p)

- $MiniBatch \leftarrow Sample(Mem, Bsize)$
- **while** $(S, A, R, t) \in MiniBatch$ **do**
  - $S_{next} \leftarrow \Delta(S, A)$
  - $A_{next} = Max(Q[S_{next}])$
  - $R_{cum} = R + \gamma R_{smart}(S_{next}, A_{next}, t)$
  - $MiniBatch[(S, A, R, t)] \leftarrow (S, A, R_{cum}, t)$
  - $Loss \leftarrow DNN_{Train}(Q, MiniBatch)$
- **if** $\epsilon \geq \epsilon_{min}$ AND $Loss \leq L_p$ **then**
  - $\epsilon \leftarrow \epsilon * \epsilon_{decay}$
  - return Q

end procedure

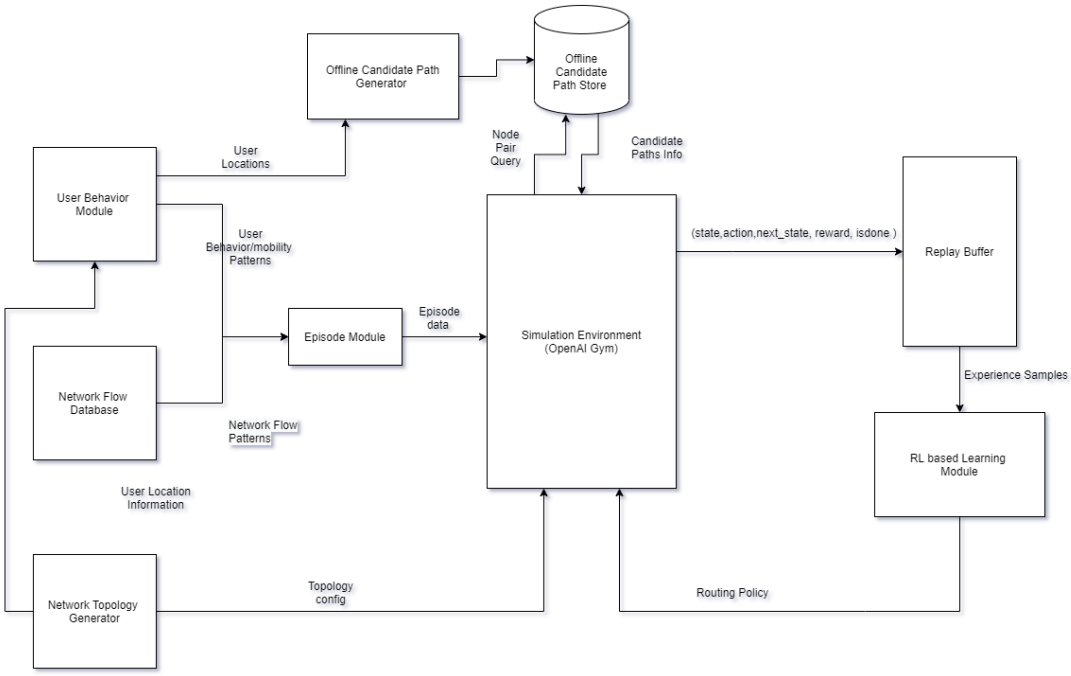


Figure 5.2. `STE-SDN` Framework

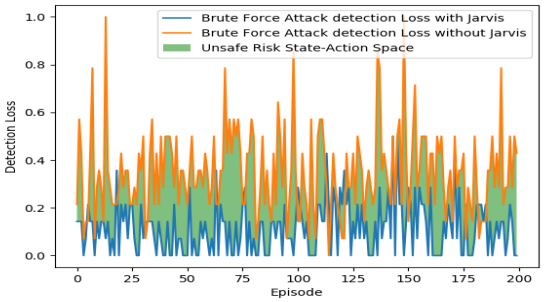


Figure 5.3. Detection Loss: Brute Force Attacks

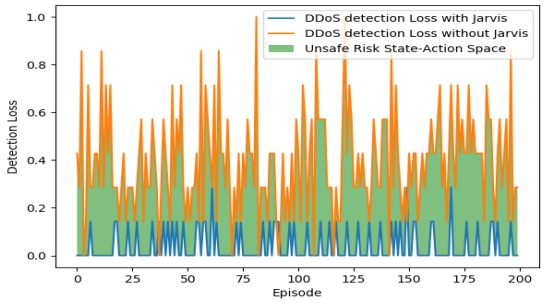


Figure 5.4. Detection Loss: DDoS Attacks

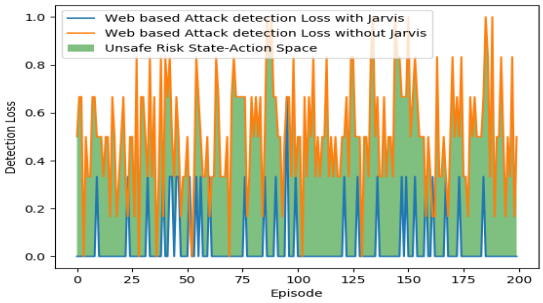


Figure 5.5. Detection Loss: Web Based Attacks

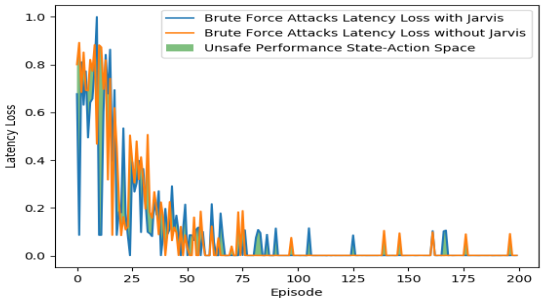


Figure 5.6. Latency Loss: Brute Force Attacks

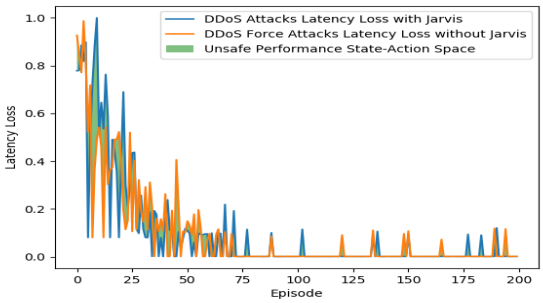


Figure 5.7. Latency Loss: DDoS Attacks

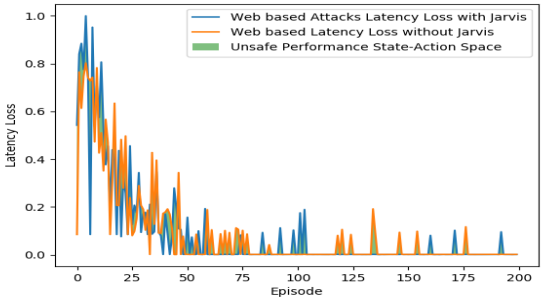


Figure 5.8. Latency Loss: Web Based Attacks

6. ACCELERATING RL LEARNING USING TRANSFER LEARNING

To begin, we introduce the Software Defined Coalitions (SDC) architecture that has been proposed by the DAIS ITA (ITA) team and discuss the fragmentation scenarios of SDC to be studied in this work. In addition, we briefly present RL techniques as network control policies for the SDC and transfer learning for the environment under consideration.

6.1 Background

6.1.1 Software Defined Coalitions (SDC)

To attain the vision of distributed brain and intelligence for joint missions of coalition forces, we need to understand how different infrastructure components in the coalition can be composed together to form an efficient, unified, agile infrastructure, in spite of substantial resource constraints, high levels of dynamicity and local policies restricting coalition participation. The key challenge is to develop the fundamental design principles and techniques by which we can obtain such infrastructure composition for distributed intelligence and analytics for coalition operations.

Toward this goal of enabling a new level of agility and dynamism, the DAIS ITA Alliance (https://dais-ita.org/pub) has introduced the new architecture called Software Defined Coalitions (SDC), to realize many benefits including programmable coalition management, easy reconfiguration, on-demand resource allocation, and rapid response to network anomaly/failures. The notion of SDC represents a major extension of the existing Software Defined Networking (SDN) concept to include all types of coalition resources such as communication, storage, computation, databases, sensors and other forms of resources. Realizing an SDC infrastructure can lead to a major advancement to support overall coalition needs, for diverse settings from combat operations to intelligent operations to humanity operations.

The core idea of a Software Defined Network (SDN) is to separate control logic of the communication network from the data plane of switches and links to support communications services. An SDN typically consists of multiple network domains, each of which has a

single controller responsible for managing switches and links within the domain. Switches and communication links are connected across the domains to form a data plane for transfer of user data, while domain controllers are also interconnected to form a control plane for exchanging information for network control purposes. Since the software implementation of control logic residing within controllers can be readily re-programmed and changed, such software-defined capability enables centralized management, rapid configuration and adjustment of communication resources to achieve higher-level goals.

SDC applies the similar concept of separation of control logic from the data plane as in the SDN. As an illustrative example in Figure 6.1, an SDC is composed of multiple domains, where each domain represents a connected sub-network of a coalition partner. Each domain contains one domain controller and controllers of various domains are connected through the control plane for the exchange of control information. It is important to note that the SDC is a logical or virtual architecture. That is, multiple SDCs can actually be running on and supported by the same set of physical resources owned by coalition partners and each of such SDCs is referred to as an SDC slice. The example shown in Figure 1 has two SDC slices denoted as slice 1 and 2. Each SDC slice has a slice controller, which connects with all domain controllers associated with the slice. These domain and slice controllers, which form a hierarchical control structure, coordinate to utilize all infrastructure resources to support services for achieving mission objectives and satisfying coalition policies for the given SDCs.

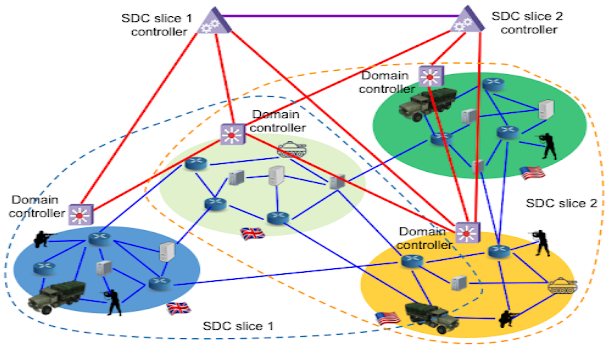


Figure 6.1. Software Defined Coalitions (SDC) for armed forces

Despite the potential advantages of SDC, major challenges need to be addressed before the SDC capability can be realized in tactical environments. SDC does not consider only switches and links, but other resources such as computational servers, storage, databases,

sensors and other forms of resources must also be considered. Given such a diversified set of resources, the control logic for maintaining and sharing resources across domains is more complicated than those for SDN. The infrastructure and service dynamicity in the tactical environments further complicate algorithms and techniques required to support the SDC capability. The DAIS ITA team has addressed these challenges. For example, the fundamental understanding of controller synchronization has been studied in [1], while techniques for controller synchronization has been developed in [2], [3] and [4]. A hybrid control architecture for SDN and ad-hoc networks to combine the advantages of central and distributed control mechanisms is proposed in [5] and efficient techniques for sharing of coalition resources across domains in SDC are investigated in [6] and [7]. Although various SDC issues have been addressed, one area that has not received much attention is how the SDC architecture will respond and behave when the infrastructure, which is formed by "joining up" various domains of resources owned by multiple coalition partners, becomes fragmented due to unexpected failures of communication links or other system components as well as planned fragmentation and re-joining for mission purposes. This is the focus in the rest of this chapter.

6.1.2 SDC Fragmentation

The main idea of SDC is to facilitate dynamic, near real-time configuration and reconfiguration of multiple domains of various resources belonging to different coalition partners to form a single infrastructure for supporting coalition missions. The severe tactical environments may cause network components and communication links in particular to fail. Such component failures can lead to a variety of conditions, including disconnection of assets and devices from their domain controllers, disconnection between domain controllers, as well as disconnection of communication links and gateways between domains. Relatively speaking, connections among resources, and between resources and their controller within a domain are more reliable and robust than the connections across domains. It is so because each domain is typically well designed and maintained as it belongs to a coalition partner or a branch of armed forces. In contrast, SDC is formed by dynamically "connecting" multiple

domains from different owners, which may use heterogeneous sets of technologies and standards. In this work, we focus on disconnection of domains in the SDC, which is referred to as SDC fragmentation here. While SDC fragmentation can be caused by unexpected failures of network components such as communication links and gateways, fragmentation and re-joining of SDC domains can also be planned in order to meet the mission objectives. That is, for the mission requirements, a subset of domains can be disconnected (and thus fragmented) from and re-join the rest of the SDC infrastructure at a later time. This is referred to as planned SDC fragmentation, in contrast to the unplanned fragmentation due to unexpected situations such as component failures. Regardless of the types of fragmentation, the SDC runs as a single infrastructure while connected, and is divided up into a set of disconnected networks during fragmentation. Naturally, control algorithms for resource allocation and service provisioning are highly affected by whether the SDC is connected or fragmented. It is important to ensure the dynamic configuration and re-configuration of resources and services can be carried out efficiently in presence of possible domain fragmentation and re-joining. Since RL techniques (e.g., [3, 4]) have been commonly used to control infrastructures, it is desirable to understand and improve operations of such learning techniques when the SDC can change suddenly from being connected to fragmented, and vice versa. In particular, as SDC fragmentation and re-joining of domains represent sudden changes of operating environment, the learning-based control algorithms in use are expected to quickly adapt to such rapid changes while ensuring satisfactory performance and robustness. The control algorithms should efficiently allocate resources, provide services and continue to learn the conditions in the fragmented networks so that the algorithms can operate efficiently when the fragmented networks are re-connected to form a single SDC infrastructure again. Essentially, the challenge here is to enhance the RL techniques to be adaptive to the fragmentation and re-connection in SDC, as to be elaborated in the following.

6.1.3 RL for Developing Network Control Policy

Network control, like many other system control problems, usually involves solving complex decision-making and optimization problems. Traditional optimisation techniques, when

employed for solving network control problems, usually require various simplifications of the original problem and strict assumptions being met. Given that SDC networks are usually large and heterogeneous systems, accurately modelling such systems and formulating tractable optimization problems for SDC is a big challenge in the first place. In addition, the traditional modelling of network control problems often cannot take full advantage of the potential large amount of network operation data, which are made available through the control plane of the SDC infrastructure. To address the shortcomings of traditional network control based on the optimisation paradigms, novel reinforcement learning (RL)-based [11] approaches have been applied to designing control policies for the SDC networks [3-4]. At the high level, RL-based techniques aim at finding the optimal strategy for solving the serial decision-making problems, in order to maximize the long-term objective of the formulated decision-making problems. RL algorithms achieve this goal by learning from its past experiences to enforce "good" behaviours and avoid "bad" ones. Therefore, RL algorithms essentially offer a way for developing network control strategies by learning from the past control decisions. In the context of SDC control , we formulate the process of control policy design for SDC as a serial decision-making problem, where existing control decisions and network operation data are the past "behaviours" and "experiences", from which RL algorithms can assist in learning the optimized control policies.

6.1.4 Transfer Learning

Transfer learning (TL) allows machine-learning (ML) algorithms to repurpose the knowledge gained by learning one or more source tasks into learning a target task. TL is useful in scenarios when there is not a lot of training data available in a particular domain, but there is a lot of existing training data available in a different but similar domain. TL has been shown to be effective for ML applications like image classification, text processing, and speech recognition.

In any TL scenario, the source and target may either differ in domain (i.e., the feature space and the feature distribution of the datasets) or task (i.e., the label space and the objective predictive function learnt from the training data). Domain adaptation (DA) is a

special case of transductive TL; that is, the source and target tasks are the same, and the domains are related but different. Most DA methods try to align the source and target data distributions to learn a domain-invariant mapping of the datasets.

Singla et al. [8] use the concept of adversarial domain adaptation by using a generative adversarial network (GAN) to train a network intrusion detection (NID) classifier. The NID classifier is trained to detect a new family of attacks (target domain) by leveraging a source dataset that contains packet data for attack from existing attack families and benign samples (source domain). An overview of the GAN architecture is shown in Fig. 2. The architecture consists of two generators which can have shared weights in some layers or all layers, which take as input the source and target datasets and train to create a domain invariant mapping so that the discriminator cannot tell whether a sample is from the source dataset or the target dataset. The discriminator trains to get better at making the domain prediction, i.e. telling whether the sample belongs to the source dataset or the target dataset. In addition, the classifier trains to be better at predicting the class of the sample - in the case of the NID scenario, the classifier predicts whether the sample is an attack sample or a benign sample. All these components learn simultaneously to achieve their respective optimization objectives. After enough training the generator along with the classifier can be taken out to act as a classifier which can classify samples from both the source and target datasets.

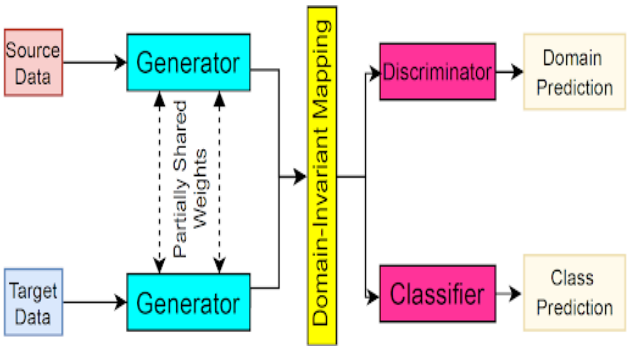


Figure 6.2. Architecture of the generic framework for adversarial DA using GANs.

6.2 Problem Formulation

6.2.1 The Analytic-Service and Fragmentation Scenario

In this section, we describe a simple but representative networking scenario as the basis for illustrating the application of the combination of RL and TL techniques for designing the network control policy. In particular, we consider a SDC consisting of two domains, i.e., the US and the UK domains. Each domain is equipped with a domain controller. The domain controller has full knowledge of the status information of all network elements (i.e., resources) in its own domain, domain controllers synchronize with each other to update each other about the status information of network elements residing in their own domains. The purpose of controller synchronization is to facilitate inter-domain networking tasks.

In this chapter, the networking elements of interest are specialized data servers that are placed at various locations throughout the domains which host network services. Multiple types of services (applications) are supported, although each data server can support and process service requests (i.e., analytic tasks) of one type of service. It is assumed that all services are available and supported by servers in both US and UK domains. Each server has an infinite buffer to temporarily store requests waiting for service and process requests on a FIFO basis. Users residing in the two domains submit requests for services to the corresponding domain controller. To realize the advantages of SDC, a service request received in one domain can be routed and processed by a server in the other domain. Specifically, each domain controller, based on its knowledge of the infrastructure status, forwards every received request for processing by an appropriate server in the local or remote domain. The purpose of such server selection for each request by the receiving domain controller is to optimise the performance metric of the SDC infrastructure as defined below.

The whole SDC of the US and UK domains is represented by a discrete-time model where service requests depart from the system upon processing completion by the servers, and arrive at the domain controllers at the time epoch immediately before and precisely when a time slot begins, respectively. New processing requests for each type of service arrive at each controller according to a Bernoulli process (see Table 1 for details). At the beginning of each time slot, each controller selects a server for each newly arrived request and forwards

the latter to the selected server for processing. Each server can process requests of one type of service. It is assumed that data servers hosting the same service can have different processing speeds (power), but each of the service requests allocated to a given server requires a fixed amount of time to complete its processing.

The status information of the servers (i.e., the multi-dimensional state description of the system) is defined as the amount of unfinished work on all servers immediately after the current time slot begins (i.e., new service requests have just arrived and been assigned to the selected servers). The amount of unfinished work on a given server is equal to the total number of time slots required to complete the current request in processing as well as all requests pending in the server's buffer. For simplicity at this point, we assume a service request forwarded from one domain to a server in the other domain does not incur additional delay or processing overhead. When the two domains are connected, their domain controllers can regularly synchronize and exchange the up-to-date server status information. During the control-plane fragmentation, domain controllers cannot exchange such information with each other, although each controller continues to know the status of its servers in the local domain. It should be noted that we assume both domains can reach each other through the data plane at all times, i.e., fragmentation only occurs to the control plane, but not the data plane.

6.2.2 Network Control Objectives and Problem Statement

Based on the scenario described above, the control task of domain controllers is to make request-forwarding decisions, such that the overall delay in satisfying all requests received are minimized. In addition, we can identify two states of the SDC, i.e., (i) the SDC's control plane is connected and (ii) the SDC' control plane is fragmented. In the former case, since domain controllers can synchronize with each other, there is effectively only one logical controller in the network and global control decisions can be made based on information obtained from all domains in the SDC network. Whereas in the latter case, each domain controller can make control decisions only based on network status information within its local domain. Therefore, the control objective of domain controllers can be further defined as to develop request-forwarding strategies by using the available network status information,

such that (i) the overall service request delays across the whole SDC are minimized when the control plane is connected, and (ii) the overall service request delays in their respective domains are minimized when the control plane is fragmented.

The request-forwarding strategies are developed based on the network status information available to the domain controllers. Since the availability of network status information will be impacted by whether or not the control plane is fragmented, the request-forwarding strategies will have to adjust accordingly. Therefore, the second network control objective of domain controllers is to be able to quickly adapt their request-forwarding strategies when the status of the control plane, being fragmented or connected, changes.

Without loss of generality, in this chapter we focus our investigation on the scenario where the control plane emerges from fragmentation. In particular, the problem we aim at addressing is two-folded: (i) develop request-forwarding strategies for domain controllers when the control plane is fragmented, and (ii) quickly adjust the individual request-forwarding strategies developed during fragmentation into a unified one when the control plane becomes reconnected.

6.3 RL for Learning Control Policy in SDC Domains

We employ RL-based methods for developing request-forwarding strategies in SDC domains, which enable domain controllers to take advantage of the large amounts of past network operation data it gathers in the SDC. These RL-based methods can be applied for both connected and fragmented control planes, as the main difference between the two cases is only the amount of data available to the domain controllers. To this end, we first formulate the problem of allocating requests to servers in the time-slotted system as an Markov Decision Process (MDP) [11], which can be described as a 3-tuple as follows.

State: The state is the amount of unfinished work (measured in time slots) of all data servers, which is the total amount of time (in unit of time slot) required to complete the request in service and all pending requests in the server buffer. Action: To determine the percentage of requests of each service type received in a time slot that are assigned to the servers (in both domains) supporting the service. If service requests from two domains are

assigned to the same server, it is assumed that the intra-domain requests will be served prior to those from the remote domain. Reward: For a time slot, the reward is calculated as the reciprocal of the average waiting time (i.e., from the arrival until the processing completion for a request) for the requests assigned during this time slot according to the action taken.

Then, the past state-action-reward tuples are stored and used by RL-based algorithms for the development of request-forwarding strategy. In particular, we leverage two off-policy and offline RL algorithms, i.e., the Deep Deterministic Policy Gradient (DDPG) [9] and the State Action Separable Reinforcement Learning sasRL [10] algorithms, to accomplish this. The DDPG is a state-of-the-art model-free and off-policy RL algorithm, which is based on the deterministic policy-gradient theorem [9]. The DDPG employs several techniques to improve data usage efficiency and to stabilize the DRL training process, such as replay buffers and the soft parameter update procedure. In addition, since our goal is to quickly develop these forwarding strategies, we also employ the sasRL algorithm, which is a new RL paradigm we recently developed for reducing the complexity of RL problems. As the design of RL algorithm itself is not tightly coupled with the main theme of this chapter, readers are referred to [9] and [10] for detailed descriptions of the mechanism behind the DDPG and the sasRL.

6.3.1 RL-TL based Control Policy Design for Fragmented SDC

We consider the following TL approach for transferring local knowledge (gained during fragmentation) from the US and UK domains to the global agent after SDC fragmentation ends. The general idea is to combine the local knowledge from the domains using various combination strategies like zero-weighted, naïve concatenation and full combination to form the source dataset (SD). These combination strategies essentially aim to conform the state of the local domains (SD) into the state of the global domain (TD). New knowledge gained in the SDC after fragmentation ends forms the target dataset (TD). Finally, a general adversarial network (GAN) is used to minimize the "domain loss" between SD and TD to improve convergence of the global RL agent. The details are given as follows.

6.3.2 Reward Knowledge Transfer

The SD is generated by combining exploration data from both UK and US domains. The TD is generated from new explorations in the whole SDC. We use a GAN (see Figure 3) to generate augmented or synthetic data (i.e., the enhanced exploration data) by minimizing the "domain loss" between SD and TD. Then we train the global RL agent using this synthetic data. A key environment specific parameter in this setting is the bias introduced while training the global agent. For the optimal convergence, the new exploration samples (TD) should be favoured over stale local domain knowledge (SD). This approach requires two learning or training steps, one to train the GAN and one to train the global RL agent.

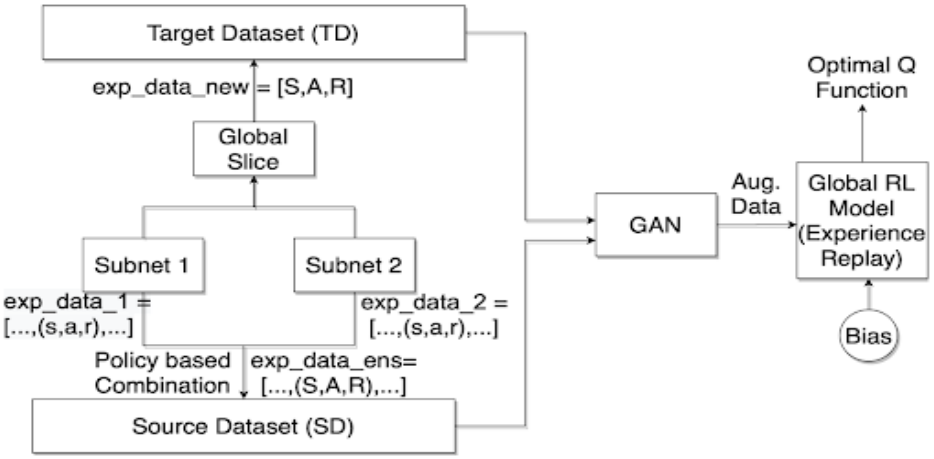


Figure 6.3. General work-flow of our TL-assisted RL approach.

6.3.3 General Work-Flow

Exploration samples are collected by respective local domain controllers required for generating their own local RL models during fragmentation. Once fragmentation ends and domains re-combine, these local explorations are exchanged by the controllers to generate the SD. The controller then proceeds to perform new explorations in the combined SDC to generate the TD. The controller samples explorations from the SD and the TD biased towards the TD using a hyperparameter b to feed into the GAN. The GAN then generates augmented exploration samples using the domain adaptation technique discussed earlier.

Finally, the RL model for the combined SDC is learnt from these augmented exploration samples. We use the DDPG [9] and the sasRL [10] as the RL algorithms for optimizing the policies, but it is important to note that our technique can be used by any general RL algorithm, which employs an experience-replay buffer for sampling.

6.4 Experiments

6.4.1 The Simulated SDC Environment

As shown in Figure 4 below, the simulated SDC environment in our experiments consists of 2 domains, i.e., the US domain and the UK domain. Moreover, we consider 3 types of analytical service. Each of the US and UK domains has at least one server supporting each of these services, and every server only provides one type of service.

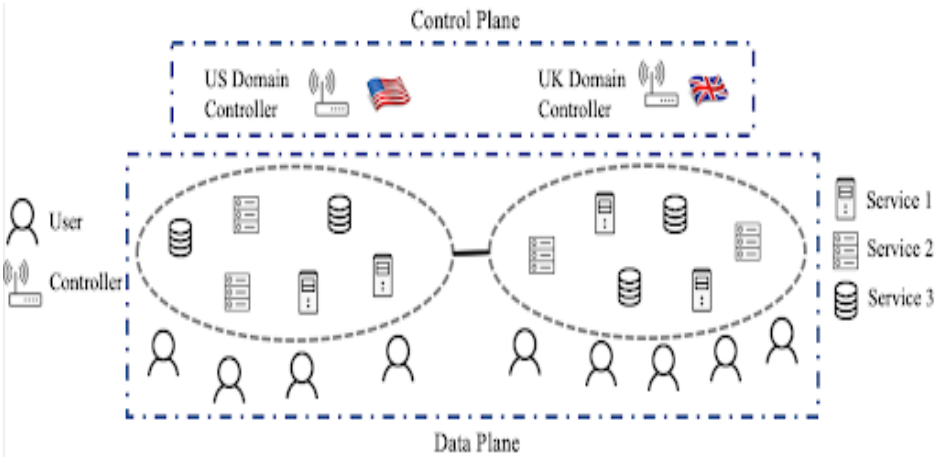


Figure 6.4. The simulated SDC environment.

We assume that the processing power of servers is abstracted by the number of time slots required for processing a unit job in the time-slotted system. The distribution of server processing power is assumed to be uniform. In addition, in the experiments we use Bernoulli distribution for modelling request arrival for the 3 network services in the SDC network. Table **??** summarizes the settings for server processing power and request arrivals in our experiments.

Table 6.1. Environment Configurations

	US domain (6 servers)	UK domain (6 servers)
Request processing time for Service 1	1st server: 1, 2 or 3 time slots, uniformly distributed 2nd server: 3, 4 or 5 time slots, uniformly distributed	1st server: 1, 2 or 3 time slots, uniformly distributed 2nd server: 3, 4 or 5 time slots, uniformly distributed
Request processing time for Service 2	3rd server: 2, 3 or 4 time slots, uniformly distributed 4th server: 5, 6 or 7 time slots, uniformly distributed	3rd server: 2, 3, or 4 time slots, uniformly distributed 4th server: 5, 6 or 7 time slots, uniformly distributed
Request processing time for Service 3	5th server: 4, 5 or 6 time slots, uniformly distributed 6th server: 9, 10 or 11 time slots, uniformly distributed	5th server: 4, 5 or 6 time slots, uniformly distributed 6th server: 9, 10 or 11 time slots, uniformly distributed
Arrivals of new requests for 3 types of services per time slot	Bernoulli arrivals: With prob $p_{\mathrm{i}}{}^{us}$, one request arrives for service i in a time slot for i=1 to 3 services; and prob (1-$p_{\mathrm{i}}{}^{us}$), no arrival	Bernoulli arrivals: With prob $p_{\mathrm{i}}{}^{uk}$, one request arrives for service i in a time slot for i=1 to 3 services; and prob (1-$p_{\mathrm{i}}{}^{uk}$), no arrival

6.4.2 RL Settings

For the DDPG algorithm, we implement the actor and critic networks as multilayer perceptrons (MLP), which consists of two hidden layers with 256 and 128 units, respectively. The discount factor and learning rate are set to 0.99 and 0.0005, respectively. In addition, we choose the conservative soft update for both the actor and critic networks, which update the parameters for both networks by 1

For the sasRL algorithm, we also implement it using the actor-critic architecture. The actor has one input layer, two hidden layers, and one output layer. The number of units in both the input layer and the output layer is the same as the dimension of the state vector. There are 64 units in both hidden layers. The critic has two input layers, three hidden layers, and one output layer. The input layer takes the current and the next state vectors as inputs, it then concatenates the two inputs to be fed to the hidden layers. Therefore, the number of units in both input layers equals the dimension of the state vector. There are 128, 64, and 32 units in three hidden layers, respectively. The transition model also has two input

layers, two hidden layers, and one output layer. The input layer takes the current and the next state vectors as inputs. Therefore, the number of units in the input layer is twice the dimension of the state vector. There are 64 and 32 units in two hidden layers, respectively. Readers are referred to [10] for more details on the sasRL architecture.

For both algorithms, we select the rectified linear unit (ReLU) and linear functions as the activation functions for all neurons in hidden layers and the neurons in the output layers, respectively. We use the Keras deep learning library for building and training the MLPs, which represent the actor, the critic, and the transition model described above. In particular, we use mean square error (MSE) and the Adam optimizer for estimating and minimizing training losses. The settings for the Adam optimizer are as follows: lr = 0.001, $_1$ = 0.9, $_2$ = 0.999, clipnorm = 1.0. The training procedure uses stochastic gradient descent operating on mini-batches of data (the minibatch size is 32) for gradient updates.

6.4.3 TL Settings

For the GAN implementation, we implement the generator and discriminator as neural networks, each of which consists of two hidden layers with 64 units each. The learning rate is set to 0.01. Each of the hidden layers uses a leaky relu activation function with slope of the negative curve set as (= 0.2). Each hidden layer is also followed by a batch normalization layer with momentum 0.8. The output of the discriminator network is a single neuron sigmoid activation and is optimized using a binary cross-entropy loss function. The generator network output dimensions are the same as the dimension of the combined SDC state. When generating augmented data from the generator, a sampling bias factor (discussed before) of 0.8 is used to bias the generated output towards the new explorations (TD).

6.4.4 Experiment Settings and Results

In our experiments, we assume that the control plane just emerged from fragmentation and domain controllers resume synchronizing with each other. Then, we test the proposed RL+TL approach under 3 scenarios with varying amount of new global data: (1) “real

exploration" with 10,000 real state-action-reward tuples, which are collected after control plane becomes reconnected; (2) "augmented exploration" with 100 real state-action-reward tuples collected after the control plane reconnection, and 9,900 augmented ones using the TL technique; (3) "limited exploration" with only 100 state-action-reward tuples collected after control plane reconnection. For updating parameters at each training iteration, we randomly pull the same number of samples from the respective sample pools in 3 scenarios. Therefore, we ensure that the same number of gradient updates are carried out for all 3 scenarios for a fair comparison. We conduct our experiments using both DDPG and sasRL as the chosen RL algorithm. Experiment results are shown in Figures 5 and 6 for DDPG and sasRL-based algorithms, respectively. In these plots, the vertical axis shows the average of accumulated rewards over a fixed number of time slots, whereas the horizontal axis corresponds to the number of training iterations. It is clear from the plots that the more state-action-reward data points we have, the higher the average accumulated reward. More importantly, by comparing the curves between "augmented exploration" and "limited exploration", we can see that although they both only use 100 real state-action-reward data points, the 9,900 augmented data points obtained via the TL technique help to produce results much closer to those obtained by using 10,000 real data points. Therefore, the TL technique we employ significantly accelerates the RL-based control policy generation by providing a large amount of synthetic data as an augmentation to the small amount of real data collected during a short period of time. In this case, the "augmented exploration" scenario can in theory start the RL process and learn control policy 100x faster than the "real exploration" scenario, and still achieve comparable results.

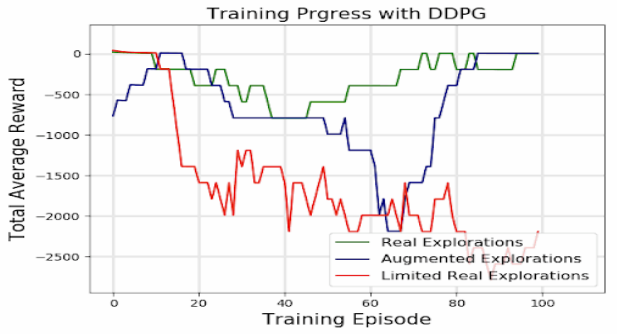


Figure 6.5. Training curves with DDPG.

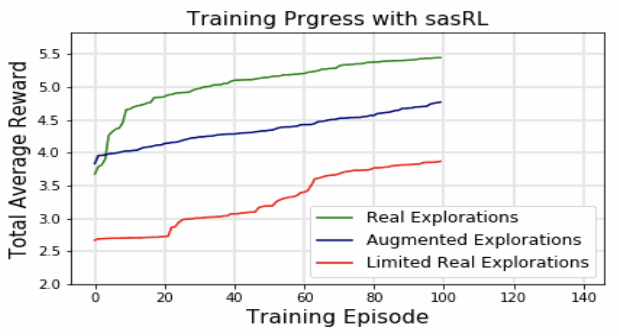


Figure 6.6. Training curves with sasRL.

6.5 Conclusion

In this chapter, we have explored the idea of applying TL techniques for accelerating RL-based network control in SDC. In particular, we consider the scenario where the control plane transitions from fragmentation to reconnection, and the domain controllers try to learn new control policies based on newly collected data after fragmentation ends. By employing the GAN-based TL technique, large amounts of synthetic data are generated, which are then used to assist the development of RL-based control policy. Experimental results show that the RL algorithms that use only 1% of real data and 99% of augmented (obtained through TL) data can achieve comparable performance to the scenario where 100% of data used are real. Therefore, we show that TL techniques can significantly accelerate the adaptation of the RL-based control policy in fragmented SDC. In essence, our results reveal that the combined RL-TL techniques can provide a rapid and adaptive response to abrupt changes of operating environments due to the SDC fragmentation. In effect, the RL-TL techniques enhance the robustness of the SDC architecture in supporting distributed analytic services despite the possible infrastructure fragmentation.

7. CONCLUSION AND FUTURE WORK

In this work, we have developed a constrained RL framework `Jarvis`, an autonomous control system deployed at the network edge able to support applications/users, by providing optimal device actions at the application and network layers to maximize QoS for the users in terms of required functionalities but at the same time maintaining safety and security the monitored IoT environment. We address various challenges of constraining DRL frameworks with safety and security policies for both application and network layers. We also address the single point of failure of network controllers by developing `Jarvis-SDN`, specifically for optimizing and securing core networking functionalities like routing and rate control. Along with this, we have designed the `Jarvis` models to be robust and flexible, so that they are applicable across different networking and environment contexts like IoT smart homes and SDN environments.

Currently, the `Jarvis` framework functions as a single RL agent framework which focuses on individual application layer functionalities, like energy saving and electricity cost minimization, and network layer functionalities, like optimal routing and rate control. One area of future work is to combine these functionalities into a single optimization goal using multiple RL agents by weighing each functionality reward using a user-defined reward function. This would be similar to the idea of encoding ethics through reward weight distribution in the smart reward function discussed in this work; however these weights would be across different RL agents and thus would avoid single RL agents getting stuck in a local optima.

REFERENCES

[1] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," pp. 147–158, 2018.

[2] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot.," 2019.

[3] S. Bauer and D. Schreckling, "Data provenance in the internet of things," 2013.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," pp. 13–16, 2012.

[5] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," pp. 685–695, 2015.

[6] D. O'Neill, M. Levorato, A. Goldsmith, and U. Mitra, "Residential demand response using reinforcement learning," pp. 409–414, 2010.

[7] L. Yu, W. Xie, D. Xie, Y. Zou, D. Zhang, Z. Sun, L. Zhang, and T. Jiang, "Deep reinforcement learning for smart home energy management," *arXiv preprint arXiv:1909.10165*, 2019.

[8] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for iot content-centric services," *Applied Soft Computing*, vol. 70, pp. 12–21, 2018.

[9] K. M. Tsui and S.-C. Chan, "Demand response optimization for smart home scheduling under real-time pricing," *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 1812–1821, 2012.

[10] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 1871–1879.

[11] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "Cfr-rl: Traffic engineering with reinforcement learning in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2249–2259, 2020.

[12] E. Einhorn and A. Mitschele-Thiel, "Rlte: Reinforcement learning for traffic-engineering," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*, Springer, 2008, pp. 120–133.

[13] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[14] X. Huang, T. Yuan, G. Qiao, and Y. Ren, "Deep reinforcement learning for multimedia traffic control in software defined networking," *IEEE Network*, vol. 32, no. 6, pp. 35–41, 2018.

[15] P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, "Marvel: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning," *Computer Networks*, p. 107 230, 2020.

[16] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[17] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[18] C. Cimpanu, *New hakai iot botnet takes aim at d-link, huawei, and realtek routers*, Sep. 2018. [Online]. Available: https://www.zdnet.com/article/new-hakai-iot-botnet-takes-aim-at-d-link-huawei-and-realtek-routers/.

[19] A. Greenberg, *The reaper botnet has already infected a million networks*, Oct. 2017. [Online]. Available: https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/.

[20] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding fileless attacks on linux-based iot devices with honeycloud," pp. 482–493, 2019.

[21] E. Bertino, "Data security and privacy in the iot," pp. 1–3, 2016.

[22] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks.," vol. 99, no. 1, pp. 229–238, 1999.

[23] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[24] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[25] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis a system for knowledge driven adaptable intrusion detection for the internet of things," pp. 656–666, 2017.

[26] J. Habibi, D. Midi, A. Mudgerikar, and E. Bertino, "Heimdall: Mitigating the internet of insecure things," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 968–978, 2017.

[27] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," pp. 1093–1110, 2017.

[28] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," pp. 519–524, 2016.

[29] J. Wallen, *Five nightmarish attacks that show the risks of iot security*, Jun. 2017. [Online]. Available: https://www.zdnet.com/article/5-nightmarish-attacks-that-show-the-risks-of-iot-security/.

[30] S. Cobb, *Rot: Ransomware of things*, 2017.

[31] K. Salah, J. M. A. Calero, S. Zeadally, S. Al-Mulla, and M. Alzaabi, "Using cloud computing to implement a security overlay network," *IEEE security & privacy*, vol. 11, no. 1, pp. 44–53, 2012.

[32] P. Calyam, S. Rajagopalan, S. Seetharam, A. Selvadhurai, K. Salah, and R. Ramnath, "Vdc-analyst: Design and verification of virtual desktop cloud resource allocations," *Computer Networks*, vol. 68, pp. 110–122, 2014.

[33] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources," vol. 2, pp. 256–261, 2013.

[34] A. Mudgerikar, P. Sharma, and E. Bertino, "E-spion: A system-level intrusion detection system for iot devices," pp. 493–500, 2019.

[35] F. Bellard, "Qemu, a fast and portable dynamic translator.," vol. 41, p. 46, 2005.

[36] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for linux-based embedded firmware.," 2016.

[37] [Online]. Available: https://linux.die.net/man/1/sha256sum.

[38] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya, "Merkle-damgård revisited: How to construct a hash function," pp. 430–448, 2005.

[39] C. Hall, *Survey shows linux the top operating system for internet of things devices*, May 2018. [Online]. Available: https://www.itprotoday.com/iot/survey-shows-linux-top-operating-system-internet-things-devices.

[40] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[41] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," *EMU*, vol. 9, p. 1, 2015.

[42] VirusTotal, [Online]. Available: https://www.virustotal.com.

[43] O. Malware, [Online]. Available: http://openmalware.org.

[44] E. J. Cho, J. H. Kim, and C. S. Hong, "Attack model and detection scheme for botnet on 6lowpan," pp. 515–518, 2009.

[45] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the rpl-based internet of things," *International Journal of Distributed Sensor Networks*, vol. 9, no. 8, p. 794 326, 2013.

[46] D. Oh, D. Kim, and W. W. Ro, "A malicious pattern detection engine for embedded security systems in the internet of things," *Sensors*, vol. 14, no. 12, pp. 24 188–24 211, 2014.

[47] T.-H. Lee, C.-H. Wen, L.-H. Chang, H.-S. Chiang, and M.-C. Hsieh, "A lightweight intrusion detection scheme based on energy consumption analysis in 6lowpan," in *Advanced Technologies, Embedded and Multimedia for Human-centric Computing*, Springer, 2014, pp. 1205–1213.

[48] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things.," pp. 606–611, 2015.

[49] P. Pongle and G. Chavan, "Real time intrusion and wormhole attack detection in internet of things," *International Journal of Computer Applications*, vol. 121, no. 9, 2015.

[50] N. K. Thanigaivelan, E. Nigussie, R. K. Kanth, S. Virtanen, and J. Isoaho, "Distributed internal anomaly detection system for internet-of-things," pp. 319–320, 2016.

[51] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, and M. A. Spirito, "An ids framework for internet of things empowered by 6lowpan," pp. 1337–1340, 2013.

[52] C. Liu, J. Yang, R. Chen, Y. Zhang, and J. Zeng, "Research on immunity-based intrusion detection technology for the internet of things," vol. 1, pp. 212–216, 2011.

[53] T. R. Chavez, "A look at linux audit," 2006.

[54] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems.," pp. 43–56, 2006.

[55] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," pp. 101–120, 2012.

[56] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting.," 2016.

[57] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," pp. 377–390, 2017.

[58] Ifttt, *Ifttt*. [Online]. Available: https://ifttt.com/.

[59] Zapier, *The easiest way to automate your work*. [Online]. Available: https://zapier.com/.

[60] *Hybrid integration platform (hip)*. [Online]. Available: https://apiant.com/.

[61] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman, "Iotsat: A formal framework for security analysis of the internet of things (iot)," pp. 180–188, 2016.

[62] T. Alshammari, N. Alshammari, M. Sedky, and C. Howard, "Simadl: Simulated activities of daily living dataset," *Data*, vol. 3, no. 2, p. 11, 2018.

[63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[64] *Smartthings*. [Online]. Available: https://www.smartthings.com/.

[65] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[66] Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," pp. 265–283, 2016.

[67] N. Alshammari, T. Alshammari, M. Sedky, J. Champion, and C. Bauer, "Openshs: Open smart home simulator," *Sensors*, vol. 17, no. 5, p. 1003, 2017.

[68] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, J. Albrecht, *et al.*, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, no. 112, p. 108, 2012.

[69] W. Ding and H. Hu, "On the safety of iot device physical interaction control," pp. 832–846, 2018.

[70] *Day-ahead market*. [Online]. Available: http://www.ercot.com/mktinfo/dam.

[71] S. Reddy, A. D. Dragan, S. Levine, S. Legg, and J. Leike, "Learning human objectives by evaluating hypothetical behavior," *arXiv preprint arXiv:1912.05652*, 2019.

[72] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[73] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information processing letters*, vol. 24, no. 6, pp. 377–380, 1987.

[74] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms.," 2017.

[75] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," pp. 361–378, 2017.

[76] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," pp. 636–654, 2016.

[77] S. Khurshid, C. S. Păsăreanu, and W. Visser, "Generalized symbolic execution for model checking and testing," pp. 553–568, 2003.

[78] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.

[79] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Helion: Enabling a natural perspective of home automation," *arXiv preprint arXiv:1907.00124*, 2019.

[80] V. Pilloni, A. Floris, A. Meloni, and L. Atzori, "Smart home energy management including renewable sources: A qoe-driven approach," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 2006–2018, 2016.

[81] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied energy*, vol. 235, pp. 1072–1089, 2019.

[82] F. De Angelis, M. Boaro, D. Fuselli, S. Squartini, F. Piazza, and Q. Wei, "Optimal home energy management under dynamic electrical and thermal constraints," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1518–1527, 2012.

[83] L. Yu, T. Jiang, and Y. Zou, "Online energy management for a sustainable smart home with an hvac load and random occupancy," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 1646–1659, 2017.

[84] M. Franceschelli, A. Pilloni, and A. Gaspam, "A heuristic approach for online distributed optimization of multi-agent networks of smart sockets and thermostatically controlled loads based on dynamic average consensus," pp. 2541–2548, 2018.

[85] M. Collotta and G. Pau, "An innovative approach for forecasting of energy requirements to improve a smart home management system based on ble," *IEEE Transactions on Green Communications and Networking*, vol. 1, no. 1, pp. 112–120, 2017.

[86] A. Mudgerikar and E. Bertino, "Jarvis: Moving towards a smarter internet of things," in *40th IEEE International Conference on Distributed Computing Systems*, IEEE, 2020.

[87] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Computing Surveys*, vol. 49, no. 4, 2017.

[88] W. H. Sanders, "Quantitative security metrics: Unattainable holy grail or a vital breakthrough within our reach?" *IEEE Security & Privacy*, vol. 12, no. 2, pp. 67–69, 2014.

[89] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP*, 2018, pp. 108–116.

[90] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[91] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*.

[92] P. Bosshart and et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[93] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[94] D. J. Hamad, K. G. Yalda, and I. T. Okumus, "Getting traffic statistics from network devices in an sdn environment using openflow," *Information Technology and Systems*, pp. 951–956, 2015.

[95] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[96] A. Ramos, M. Lazar, R. Holanda Filho, and J. J. Rodrigues, "Model-based quantitative network security metrics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2704–2734, 2017.

[97] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[98] S. W. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," 2020.

[99] C. Katsis, F. Cicala, D. Thomsen, N. Ringo, and E. B. Bertino, "Can i reach you? do i need to? new semantics in security policy specification and testing," in *SACMAT '21: The 26th ACM Symposium on Access Control Models and Technologies*, ACM, 2021, pp. 165–174.

[100] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia, "An sdn/nfv-enabled enterprise network architecture offering fine-grained security policy enforcement," *IEEE communications magazine*, vol. 55, no. 3, pp. 217–223, 2017.

[101] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator," 2000.

[102] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 4, 45–es, 2007.

[103] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International workshop on recent advances in intrusion detection*, Springer, 2011, pp. 161–180.

[104] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*, IEEE, 2010, pp. 408–415.

[105] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 international conference on wireless networks and mobile communications (WINCOM)*, IEEE, 2016, pp. 258–263.

[106] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based ddos detection system in software-defined networking (sdn)," *arXiv preprint arXiv:1611.07400*, 2016.

[107] D. Jankowski and M. Amanowicz, "On efficiency of selected machine learning algorithms for intrusion detection in software defined networks," *International Journal of Electronics and Telecommunications*, vol. 62, no. 3, pp. 247–252, 2016.

[108] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, IEEE, 2017, pp. 656–666.

[109] I. H. Abdulqadder, S. Zhou, D. Zou, I. T. Aziz, and S. M. A. Akber, "Multi-layered intrusion detection and prevention in the sdn/nfv enabled cloud of 5g networks using ai-based defense mechanisms," *Computer Networks*, vol. 179, p. 107 364, 2020.

[110] Z. Fan, Y. Xiao, A. Nayak, and C. Tan, "An improved network security situation assessment approach in software defined networks," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 295–309, 2019.

[111] H. Pillutla and A. Arjunan, "Fuzzy self organizing maps-based ddos mitigation mechanism for software defined networking in cloud computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 4, pp. 1547–1559, 2019.

[112] W. Meng, K.-K. R. Choo, S. Furnell, A. V. Vasilakos, and C. W. Probst, "Towards bayesian-based trust management for insider attacks in healthcare software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 761–773, 2018.

[113] Y. Li and R. Fu, "An parallelized deep packet inspection design in software defined network," in *Proceedings of 2nd International Conference on Information Technology and Electronic Commerce*, IEEE, 2014, pp. 6–10.

[114] T. Chin, X. Mountrouidou, X. Li, and K. Xiong, "Selective packet inspection to detect dos flooding using software defined networking (sdn)," in *2015 IEEE 35th international conference on distributed computing systems workshops*, IEEE, 2015, pp. 95–99.

[115] B. Renukadevi and S. D. M. Raja, "Deep packet inspection management application in sdn," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, IEEE, 2017, pp. 256–259.

[116] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 213–226.

[117] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, "A novel ddpg method with prioritized experience replay," in *2017 IEEE international conference on systems, man, and cybernetics (SMC)*, IEEE, 2017, pp. 316–321.

[118] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[119] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1995–2003.

[120] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

[121] J. McQuillan, I. Richer, and E. Rosen, "The new routing algorithm for the arpanet," *IEEE transactions on communications*, vol. 28, no. 5, pp. 711–719, 1980.

[122] D. L. Mills and H.-W. Braun, "The nsfnet backbone network," in *Proceedings of the ACM workshop on Frontiers in computer communications technology*, 1987, pp. 191–196.

[123] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, 2001, pp. 346–353.

[124] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[125] B. Chandrasekaran, "Survey of network traffic models," *Waschington University in St. Louis CSE*, vol. 567, 2009.

[126] A. Mudgerikar, E. Bertino, J. Lobo, and D. Verma, "A security-constrained reinforcement learningframework for software defined networks," in *International Conference on Communications*, IEEE, 2021.

[127] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on networking*, vol. 19, no. 6, pp. 1717–1730, 2011.

[128] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[129] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*, IEEE, 2013, pp. 2211–2219.

[130] S. Yang, L. Cui, Z. Chen, and W. Xiao, "An efficient approach to robust sdn controller placement for security," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1669–1682, 2020.

[131] D. Santos, A. de Sousa, and C. M. Machuca, "Robust sdn controller placement to malicious node attacks," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, 2018, pp. 1–8.

[132] L. Dridi and M. F. Zhani, "A holistic approach to mitigating dos attacks in sdn networks," *International Journal of Network Management*, vol. 28, no. 1, e1996, 2018.

[133] Y. Li, A. Papachristodoulou, M. Chiang, and A. R. Calderbank, "Congestion control and its stability in networks with delay sensitive traffic," *Computer Networks*, vol. 55, no. 1, pp. 20–32, 2011.

VITA

I am Anand Mudgerikar, a PhD student at Purdue University under the supervision of Dr. Elisa Bertino. My research interests include Information Security, Cryptography, Computer Networks and Machine Learning. My current focus is on developing 'safe' reinforcement learning frameworks for optimizing user QoS requirements and network functionalities in IoT environments.

I am also part of the DAIS-ITA project which is a collaborative arrangement between U.S. and UK governments led by IBM for research in distributed analytics. We are currently working on using transfer learning in conjugation with reinforcement learning to dynamically determine optimal policies in distributed network coalitions. I spent my summers of 2017 and 2018 at HPE Labs under the supervision of Dr. Puneet Sharma where I was involved in development of 'E-Spion: A system level intrusion detection system' and 'DITO: A honeypot service for IoT Devices'. Before pursuing my PhD, I also completed my Masters from the Center for Education and Research in Information Assurance and Security (CERIAS) with Dr. Elisa Bertino and Dr. Ioannis Papapanagiotou. My research and masters thesis were focused on development of hardware accelerated cryptographic libraries using GPUs for authentication in IoT environments.

Before coming to the states for my higher education, I completed my Bachelors in Information and Communication Technology from DA-IICT, India. For my Bachelor's thesis, I worked on improving the IPSec standard by incorporating secure multicast functionality using multi-party key computation under the guidance of Dr. Manik Lal Das. I was also working with TIFAC, Dept. of Science and Technology, Govt. of India with Dr. Prabhat Ranjan where I was responsible for conducting a research study to analyze the security threats in 3rd party applications and ensuring network security to prevent any unauthorized access.

PUBLICATIONS

Related Publications:

- **A. Mudgerikar**, E. Bertino **Intelligent Security Aware Routing: Using Model-Free Reinforcement Learning**, INFOCOMM 2021 (under submission)
- **A. Mudgerikar**, E. Bertino, J. Lobo and D. Verma, **A Security-Constrained Reinforcement Learning Framework for Software Defined Networks**, ICC 2021 - IEEE International Conference on Communications, 2021, pp. 1-6
- Ziyao Zhang, **Anand Mudgerikar**, Ankush Singla, Kin K. Leung, Elisa Bertino, Dinesh Verma, Kevin Chan, John Melrose, Jeremy Tucker, **Reinforcement and transfer learning for distributed analytics in fragmented software defined coalitions**, Proc. SPIE 11746, Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III, 117461W (12 April 2021)
- **Mudgerikar A.** and Bertino E., **Jarvis: Moving towards a smarter IoT using RL**, ICDCS 2020
- **Mudgerikar A.**, Bertino E. (2021) **IoT Attacks and Malware.**, In: Chen X., Susilo W., Bertino E. (eds) Cyber Security Meets Machine Learning. Springer, Singapore.
- **Mudgerikar A.**, Sharma P. and Bertino E., **Edge based Intrusion Detection for the Internet Of Things**, ACM Transactions on Management Information Systems, 'Special Issue on Analytics for Cybersecurity and Privacy', 2020.
- **Mudgerikar A.**, Sharma P. and Bertino E., **E-Spion: System Level Intrusion Detection for the Internet Of Things**, AsiaCCS 2019.
- Midi D., **Mudgerikar A.**, Rullo N. and Bertino E., **Kalis - A System for Knowledge-driven Adaptable Intrusion Detection for the Internet of Things**, ICDCS 2017
- Midi D., Habibi J., **Mudgerikar A.** and Bertino E., **Heimdall: Mitigating the Internet of Insecure Things**, IEEE Internet of Things Journal 2017

Related Patent

- Patent Disclosure, **Detecting attacks on computing devices** , US20190238567A1

Other Publications

- Hyunwoo Lee, **Anand Mudgerikar**, Ashish Kundu, Ninghui Li and Elisa Bertino,**IoTEDef: An Infection-Identifying and Self-Evolving Network Intrusion Detection System for IoT Early Defense**, CODASPY 2021
- Mehnaz S., **Mudgerikar A.** and Bertino E., **RWGuard: An Approach for Detection and Recovery of Cryptographic RansomWare**, RAID 2018, pp. 114-136.
- Yavuz A., **Mudgerikar A.**, Singla A., I. Papapanagiotou and Bertino E.,**Real-Time Digital Signatures for Time-Critical Networks** IEEE Transactions on Information Forensics and Security 2017
- Singla A., **Mudgerikar A.**, Papapanagiotou I. and Yavuz A., **HAA: Hardware-Accelerated Authentication for Internet of Things in Mission Critical Vehicular Networks**, MILCOMM 2015
- **Mudgerikar A.** and Das M.L., **Secure multicast using IPsec and multi-party key computation**, Int. J. Internet Technology and Secured Transactions, Inderscience Publications, Vol. 5, No. 2, pp.149-162.

Other Patent

- Patent Disclosure, **Hardware Accelerated Priority Based Message Authentication for Vehicular Networks**, OTC:2015-PAPA-67164