

# Autonomous Cyber Defense for Resilient Cyber-Physical Systems

Qisheng Zhang

Dissertation submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Applications

Jin-Hee Cho, Chair

Ing-Ray Chen

Chang-Tien Lu

Terrence J. Moore

Feng Chen

December 11, 2023

Falls Church, Virginia

Keywords: Cyber-physical systems, network resilience, network security, network adaptation, software diversity, deep reinforcement learning, fault-tolerance, recoverability, intrusion prevention, intrusion response

Copyright 2024, Qisheng Zhang

# Autonomous Cyber Defense for Resilient Cyber-Physical Systems

Qisheng Zhang

(ABSTRACT)

In this dissertation research, we design and analyze resilient cyber-physical systems (CPSs) under high network dynamics, adversarial attacks, and various uncertainties. We focus on three key system attributes to build resilient CPSs by developing a suite of the autonomous cyber defense mechanisms. First, we consider *network adaptability* to achieve the resilience of a CPS. Network adaptability represents the network ability to maintain its security and connectivity level when faced with incoming attacks. We address this by network topology adaptation. *Network topology adaptation* can contribute to quickly identifying and updating the network topology to confuse attacks by changing attack paths. We leverage *deep reinforcement learning* (DRL) to develop CPSs using network topology adaptation. Second, we consider the *fault-tolerance* of a CPS as another attribute to ensure system resilience. We aim to build a resilient CPS under severe resource constraints, adversarial attacks, and various uncertainties. We chose a solar sensor-based smart farm as one example of the CPS applications and develop a resource-aware monitoring system for the smart farms. We leverage DRL and uncertainty quantification using a belief theory, called Subjective Logic, to optimize critical tradeoffs between system performance and security under the contested CPS environments. Lastly, we study system resilience in terms of system *recoverability*. The system recoverability refers to the system's ability to recover from performance degradation or failure. In this task, we mainly focus on developing an automated intrusion response system (IRS) for CPSs. We aim to design the IRS with effective and efficient responses by reducing a false alarm rate and defense cost, respectively. Specifically, We build a lightweight

IRS for an in-vehicle controller area network (CAN) bus system operating with DRL-based autonomous driving.

# Acknowledgments

The research is partly supported by the Army Research Office under Grant Contract Numbers W91NF-20-2-014, The Commonwealth Cyber Initiative (CCI), NSF Grants 2106987 and 2107450, and Virginia Tech's ICTAS EFO Opportunity Seed Investment Grant.

# Contents

<b>List of Figures</b>	<b>xii</b>
<b>List of Tables</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research Goal . . . . .	3
1.3 Contribution . . . . .	5
1.4 Structure of the Dissertation . . . . .	6
<b>2 Literature Review</b>	<b>8</b>
2.1 Network Resilience . . . . .	8
2.1.1 Network Resilience in Network Science . . . . .	9
2.1.2 Network Resilience in Computer Science . . . . .	10
2.2 Shuffling-based Network Topology Adaptations for Resilient Networks . . . . .	12
2.3 Deep Reinforcement Learning-based Approaches for Resilient Cyber-Physical Systems . . . . .	13
2.3.1 DRL-based Network Adaptations . . . . .	13
2.3.2 DRL-based System Optimization for Wireless Sensor Networks . . . . .	14

2.4	Intrusion Detection Systems and Intrusion Response Systems . . . . .	17
2.4.1	Intrusion Detection Systems . . . . .	17
2.4.2	Intrusion Response Systems . . . . .	19
<b>3</b>	<b>Vulnerability-Aware, Software Diversity-based Network Adaptation for Resilient Software-Defined Networks</b>	<b>21</b>
3.1	Motivation & Research Goal . . . . .	21
3.2	Problem Statement . . . . .	23
3.3	Key Contributions . . . . .	24
3.4	System Model . . . . .	26
3.4.1	Network Model . . . . .	26
3.4.2	Node Model . . . . .	28
3.4.3	Attack Model . . . . .	31
3.5	SDA Algorithm Design . . . . .	32
3.5.1	Software Diversity Metric . . . . .	32
3.5.2	Software Diversity-based Bond Percolation for Network Adaptation . . . . .	33
3.5.3	Practical Operations of the SDA Algorithm . . . . .	38
3.6	Experimental Setup . . . . .	40
3.6.1	Performance Metrics . . . . .	40
3.6.2	Counterpart Baseline Schemes for Performance Comparison . . . . .	42
3.6.3	Environment Setup . . . . .	44

3.7	Numerical Results & Analysis . . . . .	47
3.7.1	Comparative Performance Analysis under a Dense Network . . . . .	47
3.7.2	Comparative Performance Analysis Under a Medium Network . . . . .	50
3.7.3	Comparative Performance Analysis Under a Sparse Network . . . . .	52
3.8	Key Findings . . . . .	55
<b>4</b>	<b>Deep Reinforcement Learning-based Vulnerability-Aware Network Adaptations for Resilient Software-Defined Networks</b>	<b>57</b>
4.1	Motivation & Research Goal . . . . .	58
4.2	Problem Statement . . . . .	60
4.3	Key Contributions . . . . .	62
4.4	System Model . . . . .	64
4.4.1	Network Model . . . . .	64
4.4.2	Node Model . . . . .	66
4.4.3	Attack Model . . . . .	67
4.5	Proposed Approach: Deep Reinforcement Learning-based Network Adaptations	69
4.5.1	Vulnerability Ranking of Edges and Nodes (VREN) . . . . .	69
4.5.2	Fractal-based Solution Search (FSS) . . . . .	73
4.5.3	DRL-based Budget Adaptation . . . . .	74
4.5.4	Greedy MTD Using Density Optimization (DO) . . . . .	78
4.5.5	Hybrid MTD with EVADE and DO . . . . .	79

4.5.6	Practical Applicability of EVADE and DO	80
4.6	Experimental Setup	82
4.6.1	Network Datasets	82
4.6.2	Parameterization	83
4.6.3	Metrics	85
4.6.4	Comparing Schemes	86
4.7	Numerical Results & Analysis	89
4.7.1	Validation of the DREVAN	89
4.7.2	Validation of the DeepNETAR	94
4.7.3	Validation of the EVADE	96
4.8	Key Findings	100
<b>5</b>	<b>Energy-Adaptive Monitoring for Resilient Smart Farms</b>	<b>102</b>
5.1	Research Goal & Motivation	102
5.2	Problem Statement	105
5.3	Key Contributions	106
5.4	System Model	108
5.4.1	Network Model	108
5.4.2	Node Model	110
5.4.3	Attack Model	111

5.5	Proposed Approach: DRL-based Animal Monitoring . . . . .	114
5.5.1	Uncertainty-Aware Animal Monitoring . . . . .	114
5.5.2	Data Aggregation at LoRa Gateways . . . . .	118
5.5.3	DRL-based Monitoring Update . . . . .	119
5.5.4	Mathematical Proof of Effectiveness Using Uncertainty Maximization	121
5.6	Experimental Setup . . . . .	123
5.6.1	Datasets . . . . .	124
5.6.2	Parameterization . . . . .	124
5.6.3	Metrics . . . . .	127
5.6.4	Comparing Schemes . . . . .	129
5.7	Numerical Results & Analysis . . . . .	130
5.7.1	Algorithmic Complexity Analysis . . . . .	130
5.7.2	Comparative Analyses . . . . .	130
5.7.3	Sensitivity Analyses . . . . .	132
5.8	Key Findings . . . . .	136
<b>6</b>	<b>Intrusion Response System for Autonomous Driving</b>	<b>138</b>
6.1	Research Goal & Motivation . . . . .	138
6.2	Problem Statement . . . . .	140
6.3	Key Contributions . . . . .	140

6.4	System Model . . . . .	141
6.4.1	Network Model . . . . .	141
6.4.2	Node Model . . . . .	143
6.4.3	Attack Model . . . . .	143
6.5	Proposed Approach: CCV-based IRS . . . . .	145
6.5.1	Intrusion Detection System for In-Vehicle Networks . . . . .	145
6.5.2	Control Command Value (CCV)-based IRS . . . . .	148
6.6	Experimental Setup . . . . .	151
6.7	Numerical Results & Analysis . . . . .	153
6.8	Key Findings . . . . .	156
<b>7</b>	<b>Conclusion</b>	<b>157</b>
7.1	Completed Work . . . . .	157
7.2	Future Work . . . . .	160
	<b>Bibliography</b>	<b>162</b>
	<b>Appendices</b>	<b>185</b>
	<b>Appendix A Appendix for Chapter 3</b>	<b>186</b>
A.1	Algorithms . . . . .	186
A.2	Real Network Topologies Used . . . . .	188

A.3	Experiments Under a Random Network . . . . .	191
A.3.1	Visualization of an ER Network . . . . .	193
A.3.2	Identification of an Optimal Fraction of Edges to be Adapted ( $\rho$ ) . . . . .	195
A.3.3	Comparative Performance Analysis . . . . .	195
A.4	Identification of Optimal $k$ under a Random Network . . . . .	199
A.5	Identification of Optimal $l$ under a Random Network . . . . .	200

# List of Figures

1.1	An overview of the dissertation research. . . . .	7
3.1	Example of the software diversity-based adaptation strategies: (a) The estimation of node $i$ 's software diversity value; and (b) The edge adaptation based on the software diversity difference in Eqs. (3.5) and (3.7). . . . .	37
3.2	Effect of $\rho$ (fraction of edges to be adapted) on performance of SDA in terms of the size of the giant component ( $S_g$ ) and the fraction of compromised nodes ( $P_c$ ). The optimal $\rho$ for the SDA scheme with respect to $S_g$ and $P_c$ in dense, medium dense, and sparse networks are identified as $\rho = -0.6, \rho = -0.4$ and $\rho = 1$ , respectively. . . . .	44
3.3	Effect of varying the fraction of attackers ( $P_a$ ) under a dense network. . . . .	48
3.4	Effect of the number of software packages ( $N_s$ ) under a dense network. . . . .	49
3.5	Effect of varying the fraction of attackers ( $P_a$ ) under a medium network. . . . .	50
3.6	Effect of the number of software packages ( $N_s$ ) under a medium network. . . . .	51
3.7	Effect of varying the fraction of attackers ( $P_a$ ) under a sparse network. . . . .	52
3.8	Effect of the number of software packages ( $N_s$ ) under a sparse network. . . . .	54
4.1	An overview of the network model. . . . .	66
4.2	Generation of self-similar fractals to reduce solution search space in edge addition and removal budgets, $(b_A, b_R)$ . . . . .	74

4.3	The overall architecture of the proposed DREVAN and DeepNETAR: The color of each node refers to a different software package installed in it. . . . .	77
4.4	DRL-based optimal budget identification in EVADE. . . . .	77
4.5	The dynamics of attacks and defenses with respect to time in terms of attack arrivals, their detection by the NIDS, and network shuffling-based MTD, as described in Section 4.5.5. . . . .	78
4.6	The overview of hybrid MTD, EVADE, with the greedy and DRL-based MTD. . . . .	79
4.7	The procedure of generating an expanded triangle based on the proposed greedy MTD algorithm: This algorithm can reduce and refine the solution search space to identify a desirable $(b_A^*, b_R^*)$ and is detailed in Section 4.5.5. . . . .	80
4.8	Converged reward with respect to training episodes. . . . .	90
4.9	Effect of varying the number of software packages available ( $l$ ) under an ER network. . . . .	92
4.10	Effect of varying the upper bound of the total adaptation budget ( $B$ ) under an ER network. . . . .	92
4.11	Effect of varying probability of state manipulation attacks ( $P_s$ ) under an ER network. . . . .	93
4.12	Effect of varying detection rate of state manipulation attacks ( $D_r$ ) under an ER network. . . . .	94
4.13	Effect of varying the number of software packages available ( $l$ ) under an ER network. . . . .	94

4.14	Effect of varying the upper bound of the total adaptation budget ( $B$ ) under an ER network. . . . .	95
4.15	Comparative performance analysis of the six MTD schemes with respect to the simulation time in terms of the size of the giant component ( $\mathcal{S}_G$ ). . . . .	97
4.16	Comparative performance analysis of the six MTD schemes with respect to time in terms of the fraction of compromised nodes ( $\mathcal{F}_C$ ). . . . .	97
4.17	Effect of varying the attack frequency ( $\lambda$ ). . . . .	98
4.18	Effect of varying the upper bound of the total adaptation budget ( $B$ ). . . . .	98
4.19	Effect of varying probability of state manipulation attacks ( $P_s$ ). . . . .	98
5.1	Wireless solar sensor node-based smart farm network. . . . .	105
5.2	The overall data flow of the smart farm network. . . . .	110
5.3	Attack scenarios by both outsider and insider attackers. Recall that $P_{EDA}$ is the probability of an external attacker performing false data injection attacks, $P_{NCA}$ is the probability of an internal attacker performing non-compliance attacks, and $P_{IDA}$ is the probability of an inside attacker performing false data injection attacks or DoS attacks. . . . .	112
5.4	The proposed Multi-Agent Deep Reinforcement Learning (MADRL) framework. . . . .	121
5.5	Performance of comparing schemes with respect to training episodes. . . . .	131
5.6	Comparative performance with respect to varying the internal attack probability ( $P_{IDA}$ ). . . . .	132
5.7	Comparative performance with respect to varying the external attack probability ( $P_{EDA}$ ). . . . .	132

5.8	Comparative performance with respect to varying the attacker’s non-compliance probability ( $P_{NCA}$ ).	133
5.9	Comparative performance with respect to varying the initial low battery level ( $T_L$ ).	133
5.10	Comparative performance with respect to varying the number of solar sensors ( $N$ ) attached to cows.	133
5.11	Comparative performance with respect to the different levels of sun exposure ( $\alpha$ ).	133
6.1	An overview of the network model.	142
6.2	Control commands in a CAN message.	143
6.3	The kinematic bicycle model.	145
6.4	An overview of the proposed control command value (CCV)-based IRS.	147
6.5	Comparative performance of the <i>Rails</i> with respect to varying the attack probability ( $P_a$ ).	154
6.6	Comparative performance of the <i>Roach</i> with respect to varying the attack probability ( $P_a$ ).	154
A.1	The network topologies used for our experimental study and their degree distributions.	194
A.2	The network topology of a random network used for our experimental study and its degree distribution.	194

A.3	Effect of the threshold of the fraction of edges adapted ( $\rho$ ) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ). . . . .	195
A.4	Effect of the connection probability between two nodes ( $p$ ) under random networks. . . . .	198
A.5	Effect of varying the fraction of seeding attacks ( $P_a$ ) under a random network.	198
A.6	Effect of the number of software packages ( $N_s$ ) under a random network. . .	198
A.7	Effect of varying $k$ (a maximum hop distance) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ) under a random network. . . . .	200
A.8	Effect of varying $l$ (a maximum number of attack paths) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ) under a random network. . . . .	200

# List of Tables

3.1	Key design parameters, their meanings, and their default values. . . . .	39
4.1	Key design parameters, meanings, and default values . . . . .	83
4.2	DRL parameters and values . . . . .	88
4.3	Asymptotic analysis of the compared schemes . . . . .	90
4.4	Asymptotic complexity analysis of the compared schemes . . . . .	96
5.1	EVD dataset description . . . . .	115
5.2	Key design parameters, their meanings and default values . . . . .	127
5.3	Asymptotic complexity analysis of the considered schemes . . . . .	130
5.4	Sensitivity analysis for adversarial attacks . . . . .	132
6.1	CCV performance W/O confidence score-based filtering (CSF) . . . . .	156

# List of Abbreviations

ADA Autonomous Driving Algorithm

ADS Autonomous Driving System

AR Accumulated Reward

BLE Bluetooth Low Energy

C Celsius

CAN Controller Area Network

CCC Control Command Controls

CCV Control Command Value

CPS Cyber-Physical Systems

CPU Central Processing Unit

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DeepNETAR Deep reinforcement learning-based NETWORK Adaptations for network Resilience algorithm

DoS Denial-of-Service

DQN Deep Q-Learning

DREVAN Deep Reinforcement Learning-based Vulnerability-Aware Network Adaptations

DRL Deep Reinforcement Learning

ECU Electronic Control Unit

ER Erdős–Rényi (ER) model

EVADE Efficient Moving Target Defense

EVD EmbediVet Implantable Temperature Devices

FAR False Alarm Rate

FDI False Data Injection

FP False Positive

FSS Fractal-based Solution Search

GPS Global Positioning System

ICS Industrial Control Systems

IDS Intrusion Detection System

IoT Internet of Things

IRS Intrusion Response System

LoRa Long Range

MADDPG Multi-Agent Deep Deterministic Policy Gradient

MADQN Multi-Agent Deep Q-Learning

MADRL Multi-Agent Deep Reinforcement Learning

MAPPO Multi-Agent Proximal Policy Optimization

MIMA Man-in-the-middle Attack

MO Monitoring Opinion

MTD Moving Target Defense

NIDS Network-based Intrusion Detection System

PDF Probability Density Function

PPO Proximal Policy Optimization

SD Software Diversity

SDA Software Diversity Adaptation

SDN Software-Defined Networks

SIR Susceptible-Infected-Removed

SL Subjective Logic

SMA State Manipulation Attack

SV Software Version

TP True Positive

UM Uncertainty Maximization

VREN Vulnerability Ranking of Edges and Nodes

WSN Wireless Sensor Networks

# Chapter 1

## Introduction

### 1.1 Motivation

Cyber-Physical Systems (CPSs) have received significant interest over the past decade spurred in part by programs run by various organizations like the National Science Foundation and the National Institute of Standards and Technology [123]. As this research continues to develop and CPSs become more prevalent in society, there is an increasing demand for research that may solve current and future issues involving CPSs, which include security, privacy, dependability, functionality, etc. This is more critical when CPSs are being developed as a service [18, 44], where the CPS features such as security and dependability necessarily become part of that service.

According to Avizienis et al. [10], the primary security attributes consist of confidentiality, integrity, and availability. The secondary security attributes include accountability, authenticity, and non-repudiation. Kharchenko [85] included availability, confidentiality, and integrity as security attributes. Humayed et al. [74] studied vulnerabilities and attacks in smart grids, medical CPSs, and smart cars, where they interpreted security as availability, one of the security goals. Trivedi et al. [151] defined dependability and security as one property where their attributes are defined based on availability, confidentiality, integrity, performance, reliability, survivability, safety, and maintainability. Compared to [10], performance and survivability are additionally considered in [151].

In this dissertation research, we are interested in ensuring security and dependability of a system to build resilient cyber-physical systems. CPSs include various network environments, such as Internet-of-Things (IoT), smart environments (e.g., smart homes, smart cities, smart farms) [17], and Industrial Control Systems (ICSs) [42, 113]. The key design challenges of those CPS environments include [134]:

- Majority of CPS environments often take forms of distributed communications, data filtering/processing, and a large amount of data dissemination in highly different forms (e.g., text, voice, haptic, image, and video) for large-scale networks by heterogeneous entities (e.g., devices or humans). Ensuring scalability for distributed communications in CPS environments is not a trivial problem to solve.
- We often encounter severe resource constraints in battery, computation, communication (e.g., bandwidth), and storage. Ensuring seamless service delivery in resource-constrained CPS environments, such as solar sensor-based smart farms considered in this research, is highly challenging.
- As a network becomes more complex, heterogeneous, and becomes more large-scaled (e.g., a network of networks), more complex and intelligent adversarial attackers have emerged. This can easily introduce compromised, deceptive entities and data and hinder critical decision making processes of operations to ensure system defense, security, and performance.
- CPSs often include highly dynamic systems, such as tactical networks executing military or search and rescue operations to handle emergency situations. In addition, there are high dynamics of interactions between entities, data, network topology and resources available in which each component dynamically changes in time and space. These can generate various types of uncertainties and require careful investigation of

identifying critical tradeoffs to meet multiple conflicting system objectives.

By considering all these challenges above, this research has an overarching research goal as discussed in the next section.

## 1.2 Research Goal

In this dissertation research, we aim to build resilient technologies that can enhance both dependability and security of cyber-physical systems by leveraging the state-of-the-art artificial intelligence, machine learning, and deep learning approaches. In particular, we adopt the concept of system resilience in Cho et al. [31] where system resilience is defined based on three attributes: *adaptability* to sudden environmental changes, defects, or adversarial attacks, *fault-tolerance* in the presence of faults including attacks or defects (or malfunctioning), and *recoverability* from performance degradation or failure. To this end, we propose the following three research tasks as follows.

### **[Adaptability Task]: Proactive Network Topology Adaptations for Resilient Cyber-Physical Systems**

In this task, we will identify an optimal network topology that can minimize security vulnerabilities caused by software vulnerabilities while maintaining an acceptable level of network connectivity. To this end, we will take a network topology-based moving target defense approach by adding edges to increase network connectivity or removing edges to reduce security vulnerabilities from epidemic attacks such as malware or virus propagation. The edge adaptations strategy will aim to deter threats and prevent their escalation in the attack severity and level. We will propose novel resource-aware deep reinforcement learning (DRL) to design autonomous edge adaptations. This task has been accomplished with two journal

papers [171, 174] and three conference papers [170, 172, 176].

### **[Fault-Tolerance Task]: Fault-Tolerant Service Delivery in Resource-Constrained Cyber-Physical Systems**

In this task, we will aim to develop a suite of fault-tolerant CPSs where seamless services are provided in the presence of adversarial attacks. In this task, we will focus on how a CPS can be fault-tolerant against attacks and resource constraints by delivering seamless services. As an example fault-tolerant service delivery application under high resource constraints, we chose a smart farm environment that uses solar sensors to monitor cows attached with them. The problem we aim to solve is how to ensure high monitoring quality of cows in the smart farm in the presence of severe resource constraints which does not guarantee continuous service provision as well as adversarial attacks aiming to inject false information or leak farm information to competitors. We will also leverage DRL to identify optimal monitoring settings that can maximize seamless service delivery even in the presence of the resource constraints and adversarial attacks. In addition, we will consider different types of uncertainties in sensed data received by solar sensors that can be the basis of making data update decisions with the aim of maximizing resource utilization with fluctuating energy as well as the monitoring quality of the smart farm. This work is accomplished with a conference paper [173] and a journal paper [175].

### **[Recoverability Task]: Resource-Aware Intrusion Response System for Resilient Cyber-Physical Systems**

In this last task, we will develop an automated intrusion response system (IRS) that can minimize false alarms while providing agile actions towards correctly detected threats based on their risk and urgency identified. Our focus is on handling false alarms to save defense costs while autonomously reasoning, and making decisions to identify an optimal response

towards detected attacks/failures that require immediate actions. We will focus on DRL-based autonomous driving and the proposed IRS can leverage the DRL agents' outputs and make autonomous decisions to select the right responses with minimum cost and maximum effectiveness. This work is accomplished with a submitted conference paper [177].

## 1.3 Contribution

We have made the following contributions:

- To address the **Adaptability Task**, we develop a software diversity metric for measuring a network topology in terms of minimizing security vulnerabilities against epidemic attacks (e.g., malware/virus spreading) while maintaining a sufficient level of network connectivity to provide seamless service availability.
- To address the **Adaptability Task**, we develop a deep reinforcement learning-based framework to identify a network topology robust against epidemic attacks by adding or removing edges between nodes to minimize security vulnerabilities introduced by compromised nodes while maximizing network connectivity even under epidemic attacks within the limits of budget to adjust edges.
- To address the **Fault-Tolerance Task**, we propose an uncertainty-aware DRL-based energy-adaptive monitoring system for the smart farm. The proposed system could achieve a high monitoring quality in the smart farm under fluctuating energy and cyberattacks disrupting the operations of collecting sensed data from solar sensors.
- To address the **Recoverability Task**, we build an IRS integrated into DRL-based autonomous driving systems. Our proposed IRS provides a lightweight defense solution

for an in-vehicle controller area network (CAN) bus system under false data injection attacks.

- Overall, we have addressed **Adaptability Task**, **Fault-Tolerance Task**, and **Recoverability Task** with two network adaptation frameworks, one smart farm monitoring system, and one intrusion response system for autonomous driving.

## 1.4 Structure of the Dissertation

This document is prepared as the research defense for my PhD dissertation research and has the following structure:

- Chapter 2 provides the related literature review in terms of the concepts of network resilience, shuffling-based network topology adaptations, deep reinforcement learning-based approaches, intrusion detection systems, and intrusion response systems for resilient CPSs.
- To address the **Adaptability Task**, Chapter 3 describes our proposed vulnerability-aware, software diversity-based network adaptation strategies for software-defined networks (SDNs) based on our published journal papers in [171, 174].
- To address the **Adaptability Task**, Chapter 4 discusses DRL-based vulnerability-aware network adaptations for SDNs based on our two published conference papers [170, 172] and one accepted conference paper [176].
- To address the **Fault-Tolerance Task**, Chapter 5 addresses our proposed energy-adaptive monitoring system for resilient smart farms based on our published conference paper [173] and submitted journal paper [175].

- To address the **Recoverability Task**, Chapter 6 presents our proposed intrusion response system integrated into DRL-based autonomous driving systems based on our submitted conference paper [177].
- Chapter 7 gives the summary of the works completed and the future research directions.
- Additional experiments and their results conducted for the research addressed in Chapter 3 are also included in the Appendix.

The following Fig. 1.1 shows an overview of my PhD dissertation research.

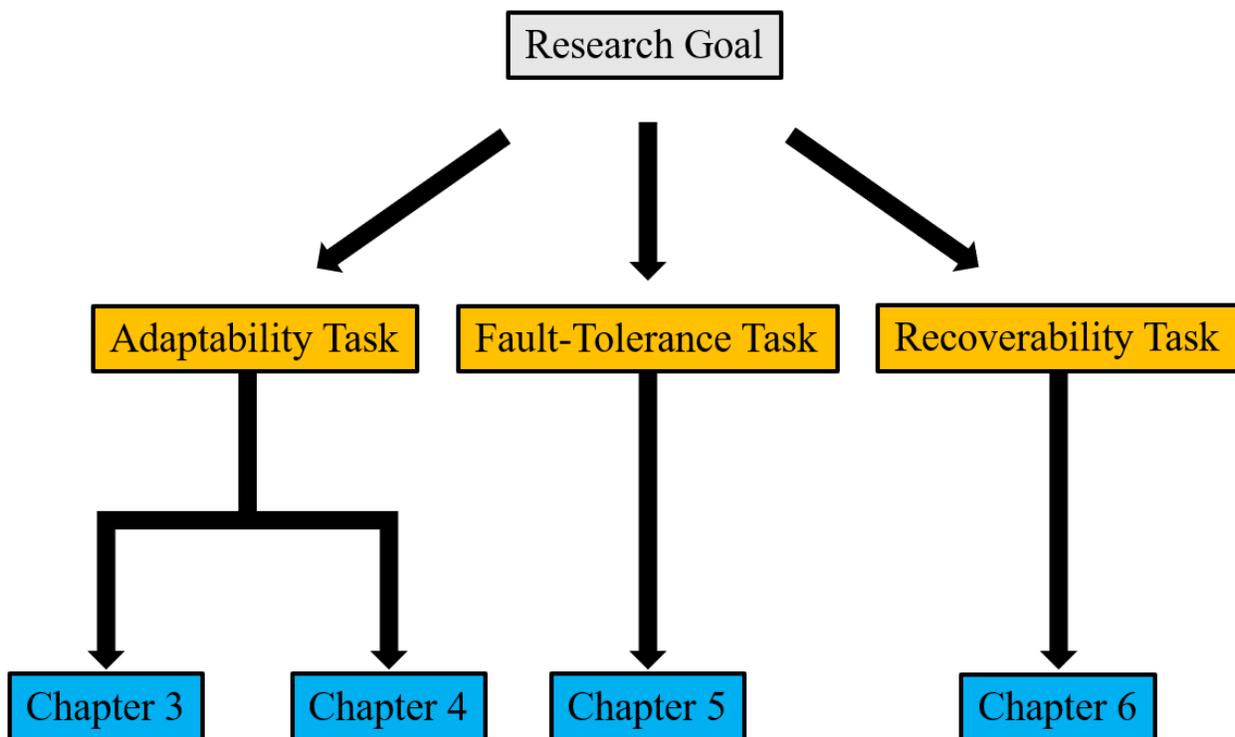


Figure 1.1: An overview of the dissertation research.

# Chapter 2

## Literature Review

### 2.1 Network Resilience

The concept of *network resilience* has been widely discussed in various domains. However, it has not been fully agreed by research communities because each domain has explored its research in different aspects. Although multiple disciplines discussed the concept of ‘resilience’, in this dissertation research, we will consider resilience at a computer-based system level as discussed in [31]. Following the concept of resilience by Cho et al. [31], we define system resilience in terms of three attributes: *adaptability* to sudden environmental changes, defects, or adversarial attacks, *fault-tolerance* in the presence of faults including attacks or defects (or malfunctioning), and *recoverability* from performance degradation or failure.

Mainly network resilience research has been conducted in the network science domain and computer science domain but with slightly different flavors. We found the network scientists studied network resilience based on the degree of fault-tolerance. On the other hand, computer scientists have studied based on all three aspects although fault-tolerance has been more considered than adaptability and recoverability when considering a resilient system or network. In the following subsections, we discuss the concepts of network resilience in both network science and computer science.

### 2.1.1 Network Resilience in Network Science

Percolation theory has been substantially used to investigate network resilience (or robustness) in network science. *Site percolation* and *bond percolation* are commonly used to select a node or an edge to remove or add, to model the choice of nodes to immunize in the context of epidemics on networks, such as disease transmission, computer malware/virus spreading, or behavior propagation (e.g., product adoption) [41, 118].

The origins of percolation theory come from the mathematical formalization of statistical physics research on the flow of liquid through a medium [59]. Percolation theory has been substantially applied to networks to study connectivity, robustness [11, 118], reliability [100], and epidemics [15, 112]. The percolation process was studied in computer science under the notion of “network resilience” [33, 114, 143], independent of its development in the statistical physics literature. More recent developments in the physics literature have profoundly influenced studies in computer science. These contributions have incorporated the recognition that networks are not derived from a random structure, and failures of nodes, whether from attacks or due to dependent cascades, are not uniformly random [6]. Hence, significant interest has developed in removal processes that model targeted attacks on the network using a centrality metric. In the network science domain, the degree of network resilience is commonly measured based on the size of the giant component (i.e., the largest connected component in a given network), which gives a clear sense of how the network is connected with a portion of existing nodes even after a certain number of nodes or edges are removed. Percolation theory has been used to model various processes on networks in the context of network failures or attacks, e.g., connectivity, routing, and epidemic spreading [33, 114].

### 2.1.2 Network Resilience in Computer Science

Recently, percolation theory was leveraged to develop software diversity techniques particularly to solve a software assignment problem [166, 167] because how nodes are connected matters in propagating malware infection while choosing nodes or edges to add or remove is exactly following the concept of site or bond percolation in percolation theory [118].

Furthermore, many approaches have been explored to validate the usefulness of software or network diversity to ensure network resilience. Chen and May [25] investigated the usefulness of software diversity to enhance security. Huang et al. [71, 72] solved a software assignment problem by isolating nodes with the same software to minimize the effect of epidemic worm attacks. This work considered not only the constraints of host functionality and software availability, but also the degree and effect of vulnerability to maximize system security based on the balance between these two key factors. Franz [54] proposed an approach to introduce compiler-generated software diversity for a large scale network, aiming to create hurdles for attackers and eliminate any advantage of knowing the vulnerabilities of a single software. An ‘App Store’ with a diversification engine automatically generates a different version of every program, which is functionally identical whenever a download is requested. Homescu et al. [66] presented a large-scale automated software diversification to mitigate the vulnerabilities exposed by software monoculture. This work gives the description of the implementation of the developed automated software diversity tool and proposed a generic method to measure the effectiveness of software diversity as a potential cure to remove code-reuse attacks. Yang et al. [166, 167] proposed a software diversity technique to combat sensor worms by solving a software assignment problem, given a limited number of software versions available. Taking the key philosophy of software diversity to prolong system survivability, The authors used percolation theory to model the design features of software diversity to defend against sensor worms. Authors also extended this work to investigate the effect of sensors with multiple

software versions, rather than a single version in terms of network robustness under sensor worm attacks [167]. Salako and Strigini [136] developed probability models to consider the commonalities in developing multiple versions of software and proposed alternative ways to consider software diversity in the presence of dependencies between the different versions of software.

Zhang et al. [179] developed a resilient, heterogeneous networking-based system where a single solution was common to increase interoperability. Recently, *network diversity* is proposed as a security metric to measure network resilience against zero-day attacks [169]. This work designed and evaluated a suite of network diversity metrics such as a biodiversity-inspired metric based on the number of different resources, which has positively impacted in enhancing security. Inspired by the network diversity metrics [169], Li et al. [102] further developed the network model and diversity metric based on vulnerability similarity, configuration constraints and multi-label hosts. In order to prevent malware propagation, they tried to solve a software assignment problem and achieve the optimal software distribution. Hosseini and Azgomi [70] mathematically analyzed the malware propagation under a network with six different types of nodes in an epidemic model. They conducted a sensitivity analysis on the number of software configurations by randomly distributing software over all nodes in the network. They proved a positive correlation between network security and the degree of network diversity. Prieto et al. [129] proposed an optimal software assignment algorithm with multiple software packages to enhance network resilience under attacks. Based on their theoretic models and related results in experiments on eight real-world network topologies, their work demonstrates high impact of software diversity on realizing network resilience.

These existing related work shows the applicability of practical use of percolation theory and software diversity in real scenarios to preserve system security and network resilience.

Although the above works discussed the concept of software diversity to ensure system

security, their aim is to solve a software assignment problem by shuffling different types of software packages among nodes without changing the network topology. Unlike the software assignment approach, we aim to generate an optimal network topology that is resilient against epidemic attacks while maximizing network connectivity. The proposed software diversity metric is designed for each node to make a decision on whether to add or remove an edge based on the vulnerabilities on the attack paths reachable to the node [27, 83].

## 2.2 Shuffling-based Network Topology Adaptations for Resilient Networks

Network topology shuffling has been studied under the paradigm of moving target defense (MTD). This technique aims to identify an optimal assignment of software variants to maximize the degree of software variants along attack paths. The main idea is to increase attack cost or complexity for an attacker by increasing hurdles in reaching a target node. Recent MTD network shuffling approaches redirect a certain number of edges to make the network topology more robust against worm attacks [68]. Hong et al. [68] solved a network shuffling problem for software assignment as an online moving target defense (MTD) mechanism. Their proposed algorithm was designed to redirect a certain number of edges such that the reconfigured network topology can be more robust against worm attacks. The key idea is to attain the maximum average number of software variants in attack paths with a fixed number of edges and a fixed assignment protocol in a network. Zhang et al. [178] developed a mechanism for networks with underlying multimedia services in order to redirect routes periodically by a Deep Q-learning method, aiming to thwart Denial-of-Service (DoS) and targeted attacks to routes. For each step, DRL agents can determine the mutated routes for each node and flow from the source to the destination. Similar to [68, 178], Chai et al. [19]

proposed ‘a deep reinforcement learning-based moving target defense against DDoS attacks’ called *DQ-MOTAG* where the MTD is adopted to shuffle the connections between users and servers in a given CPS. The DQ-MOTAG provides the ability to intelligently shuffle the duration of triggering the MTD operation based on reinforcement learning. The authors demonstrated the outperformance of DQ-MOTAG in terms of the system availability and performance. In addition, MTD network shuffling has handled reconnaissance attacks by changing virtual network topologies [2].

However, determining an optimal number of edge adaptations to obtain an optimal topology considering both security vulnerability and network connectivity has not been studied. In our work, we leveraged DRL to identify an optimal number of edge adaptations for both edge additions and removals, in terms of minimizing security vulnerability and maximizing network connectivity.

## 2.3 Deep Reinforcement Learning-based Approaches for Resilient Cyber-Physical Systems

### 2.3.1 DRL-based Network Adaptations

Recently graph embedding-based topology adaptation algorithms using deep reinforcement learning (DRL) have been proposed to develop adversarial attacks in graph neural networks (GNN). Dai et al. [35] used structure2vec (S2V) to obtain a node embedding, which was used by the DRL agent to learn two independent Q-functions in deep Q-learning (DQN). However, their work only considered removing a single edge to perform adversarial attacks on GNN models. Similarly, Darvariu et al. [36] adapted a network topology by adding edges from an empty network based on an S2V variant to obtain a node embedding and

used a customized metric to evaluate network robustness. Chai et al. [19] proposed a DRL-based network topology shuffling MTD framework to deal with Distributed Denial-of-Service (DDoS) attacks by shuffling connections between users and servers in a given CPS. Zhang et al. [178] proposed a mechanism to periodically redirect routes in which DQN agents determine a mutated route to thwart DoS and targeted attacks to routes.

Unlike these works, our approach considers both the additions and removals in the edge rewiring processes and leveraged DRL to identify an optimal number of edge adaptations which will support mitigation of attacks while the attacker expends their resources with little to no success.

Even though the above works leveraged DQN to identify an optimal network topology robust against adversarial attacks, due to the inherent nature of the problem complexity with a large solution space, it is highly challenging for a DRL agent to achieve fast convergence or even find the optimal solution. We propose a novel, break-through idea to reduce this solution space. In addition, our network adaptation algorithm allows the DRL agent to intelligently converge to an optimal (or close-to-optimal) solution without suffering much from any lack of convergence in the reward, which has been a common and challenging issue in DRL research.

### 2.3.2 DRL-based System Optimization for Wireless Sensor Networks

Algorithms to achieve energy-aware wireless sensor networks (WSNs) have been proposed in various WSN applications. A cluster based routing protocol was proposed based on a Q-learning approach called *QL-Cluster* [87]. The QL-Cluster was designed to identify the best routes between individual nodes and remote healthcare stations to continuously and efficiently monitor a patient's health. Qi et al. [132] proposed an adaptive energy

management strategy for a solar-powered WSN with hybrid storage, consisting of both supercapacitors and batteries, based on avoiding high current charging/discharge of the batteries and making full use of the supercapacitors.

Chen et al. [24] proposed a sleep scheduling algorithm for rechargeable sensors based on a DRL algorithm. The authors developed a precedence operator-based group formation algorithm to ensure the desired area coverage and a Q-learning-based active node selection algorithm to maximize the network lifetime while achieving an acceptable coverage. Chen et al. [22, 23] leveraged DRL with Q-learning to control power for communications between the in-body sensor and Wireless Body Area Networks (WBANs) coordinator to build jamming attack-resistant, healthcare applications. Their research aimed to develop a WBAN coordinator that chooses the sensor to transmit the data in the next time slot and decides the transmitting power of these sensors, which is then sent to the sensor. The WBAN coordinator uses Q-learning to achieve an optimal power control strategy.

The proposed Q-learning algorithm uses transfer learning to learn the Q-learning parameters from similar experiences so that it avoids random explorations at the start of the learning process. Furthermore, a hotbooting technique is employed to speed up the learning rate, which uses power control experiences in similar situations to initialise the Q-table at the beginning of learning. To evaluate the power control scheme, a Stackelberg game is designed to model the sequential interaction between the in-body sensors and the jammers in WBAN, and consequently Stackelberg Equilibrium (SE) is derived, which would be the upper bound for the performance of in-body sensors in different WBAN conditions. Results of the Stackelberg game showed that the algorithm proposed converges to the SE after multiple time slots which greatly reduces the energy consumption and hence reduces the attacker's jamming possibility. Thus a reinforcement learning-based power control scheme is designed to resist jamming attacks without knowing the WBAN transmission parameters and the jamming

models. Based on Q-learning and the application of transfer learning for learning the Q-learning parameters (to avoid random exploration at the start of the learning process), Chen et al. [22] achieved an optimal power control strategy that can help the WBAN coordinator choose the sensor for transmitting the data along with the transmitting power for these sensors. Similarly, to address the issues of transmission reliability, energy efficiency and Quality of Service (QoS) in WBANs, Chen et al. [23] proposed a sensor access control scheme based on DRL for the WBAN coordinator to choose the access time and transmit power of the sensor based on the state of the sensor, including signal-to-interference plus noise ratio, the transmission priority, battery level and transmission delay. Furthermore, to accelerate the learning process and address the high dimensionality problem due to the increasing number of sensors, convoluted neural network (CNN) is used to estimate the Q-values according to an approximate Q-function. The deep reinforcement learning model employed outperforms the reinforcement learning based model and the q-learning based strategies in terms of Bit Error Rate (BER), transmission delay, energy consumption, and overall utility and converges to the theoretical value expected. Similar to their previous work [22], the interactions between sensors with the coordinator while competing for transmission resources is modelled as a dynamic cooperative game and transfer learning is applied to learn the parameters from similar experience to avoid random exploration at the start of learning. Reinforcement learning, being an unsupervised learning method, is used since it is not always possible to estimate the channel conditions or the game models. The proposed algorithm was evaluated by simulating on a network with one coordinator and four sensors configured in a star topology and was found to achieve higher performance to benchmark strategies. Since transfer learning often raises privacy concerns in a small feature space application, Zhuo et al. [183] propose a novel federated DRL framework, called *FedRL*, to build models of high quality for agents while also preserving their privacy. The FedRL framework aims to learn a private Q-network policy for each agent by sharing limited information, which is the output of the

Q-network, amongst the agents. The information sent is encoded and the information received is decoded by using Gaussian differentials. The framework is evaluated on two diverse domains - grid world and text2action against four different baselines DQN-alpha, DQN-full, FCN-alpha and FCN-full and it was found to outperform DQN-alpha and FCN-alpha and a similar performance was achieved in comparison to DQN-full.

As discussed above, DRL has been applied in WSN environments to develop energy-efficient WSNs where privacy concerns are considered in applying deep neural networks (DNNs). However, no prior work has proposed an energy-adaptive monitoring system for smart farms with solar sensors where DRL and belief theory are used to maximize uncertainty-aware monitoring quality under limited and fluctuating energy of sensors. Specifically, we develop uncertainty-aware DRL algorithms to maximize uncertainty-aware monitoring quality with sensors having limited and fluctuating energy harvesting.

## 2.4 Intrusion Detection Systems and Intrusion Response Systems

### 2.4.1 Intrusion Detection Systems

An intrusion detection system (IDS) has been classified in terms of how to deploy it or how to detect attacks. The deployment-based IDS classification divides the type of the IDS into *host-based IDSs* (HIDSs) and *network-based IDSs* (NIDSs). The detection method-based IDS classifies IDS approaches into *signature-based IDS* (SIDS) and *anomaly detection-based IDS* (AIDS). This section mainly discusses IDS approaches that belong to NIDS and AIDS, which are well-aligned with the IDS used in our work.

AIDS, a.k.a. a *behavior-based IDS*, is based on the idea of clearly defining a profile for normal activities. Any deviation from this normal profile will be considered as an anomaly or abnormal behavior [105, 115]. The major advantages of AIDS are its ability to detect unknown and new attacks [182] and the customized nature of the normal activity profile for different networks and applications [60]. However, the main drawback is the high false alarm rate (FAR), as it is difficult to find the boundary between the normal and abnormal profiles for intrusion detection [20].

Multiple methods have been investigated to reduce false alarms and address this issue. These approaches include alert clustering, alert classification, and alert correlation. Alert clustering, a.k.a. alarm aggregation and alarm fusion, uses a set of unlabeled alarms and creates a set of clusters of similar type [4, 5, 40, 95, 125]. Later, these clusters are considered false or true alarms. Alert classification assumes a set of labeled alarms available for training a classification algorithm [13, 108, 124, 126, 127, 142]. An expert usually initially labels the alarms as true positive (TP) or false positive (FP), which is used for training a classifier. Once the classifier is trained, it can be used to classify future alarms. Alert correlation analyzes a large volume of IDS alarms possibly generated by different places of the network and different IDS engines and reconstructs the attack scenarios [49, 57, 145, 153, 155]. These works provide an output by merging some related alarms and generating a condensed view of the attack scenario for the administrator.

Hamada et al. [63] proposed CDEC as an anomaly-based IDS for in-vehicle networks. CDEC uses a vehicle data model to estimate the expected sensor-based control data and compare it with the actual control data for intrusion detection. Since they used correlated data to train a regression model for anomaly detection, the performance of CDEC is limited by the model's representational capacity and data correlation in the memory.

Similar to [63], our work leverages vehicle control data to train an anomaly-based IDS. But

unlike [63], the underlying model used in our approach is a bicycle forward model with a high representational capacity for vehicle dynamics and few parameters to learn. As the forward model captures temporal correlation in nature, this allows our IDS to work without additional analyses of the data correlation. Furthermore, our work falls into the *alert classification* as we adopt a confidence score-based online classification model to minimize the false alarms given by an IDS.

### 2.4.2 Intrusion Response Systems

The state-of-the-art IRS research is often classified into three categories based on the degree or level of automation: notification, manual, and automated response systems (ARSs) [76]. Our proposed IRS belongs to the ARSs, and we mainly review them for in-vehicle networks. Kwon et al. [91] developed a mitigation mechanism against in-vehicle network intrusion by reconfiguring electronic control units (ECUs) and disabling attack packets. Hamad et al. [62] investigated the general IRS framework for in-vehicle networks. The responses are dynamically activated based on attack severity, system properties, and operational modes. However, they did not select responses based on the alerts generated by the IDS.

Cheng et al. [28] used a zero-sum stochastic game to model the network intrusions in a Bayesian attack graph. They introduced different orders of *Theory of Mind* to describe the attacker and defender's actions. They assumed that attackers could fully observe the system's vulnerability. Furthermore, defenders could thwart the attackers by removing the compromised nodes or shielding the vulnerable nodes in the network. However, they did not consider the partially observable environment due to false positives generated by the IDS. Ullah et al. [152] developed an IRS for targeted attacks where the system aimed to balance the security and operational efficiency by considering both attack opportunity and

functional dependency in its objective function. Their experiments showed that the proposed IRS significantly deterred attacks by prolonging their attack paths. However, their proposed IRS aimed to defer attacks rather than defend them. Thus, its application is limited to multi-step attack sequences. Nespoli et al. [116] studied a *immuno*-based response system where the system assets could be equipped with different countermeasures. Their main objective was to find the optimal combinations of antibodies to reduce security risks in a timely manner. They used a Genetic Algorithm to optimize the selection of the solution's parameters. However, their proposed approach did not aim to learn a model for future parameter selections, which can introduce huge computational overheads for repetitive tasks.

DRL has been used to develop model-free intrusion response systems. Hughes et al. [73] used deep Q-learning to develop the automatic response system. They handcrafted 21 actions using different system functions. They also defined the reward based on the alert number, nodes compromised, and packet error. The proposed DRL-based IRS showed great performance using fine-tuned hyper-parameters. Iannucci et al. [75] also developed a model-free IRS based on (Deep) Q-Learning. They designed the states as node attributes, such as node activity and node vulnerability. They chose a reward function to minimize the system cost and execution time of a given response. However, the current DRL-based IRS approaches have large action spaces, which may cause slow convergence and require huge amounts of training data. This is not suitable for autonomous driving tasks as interactions with complex traffic environments have expensive costs.

Unlike the existing IRS research above using offline learning, our work leverages the value function in a DRL-based autonomous driving algorithm and builds an online IRS to choose the best response based on the utilities of control commands returned by the value function of the DRL-based autonomous driving algorithm.

# Chapter 3

## Vulnerability-Aware, Software Diversity-based Network Adaptation for Resilient Software-Defined Networks

This chapter addresses the **Adaptability Task**. This chapter is based on the paper “Vulnerability-Aware Resilient Networks: Software Diversity-based Network Adaptation” published in IEEE Transactions on Network Services and Management, 2020 [171].

### 3.1 Motivation & Research Goal

Inspired by the close relationship between the diversity of species and the resilience of ecosystems [156], information and software assurance research has evolved to include the concept of *software diversity* for enhanced security [64, 65, 92, 93, 121]. Due to the dominant trend of software monoculture deployment for efficiency and effectiveness of service provisions, attackers have been granted significant advantages in that acquiring the intelligence needed to compromise a single software vulnerability enables the capability of efficiently compromising other homogeneous system components, such as operating systems, software packages,

and/or hardware packages [167]. To deny this advantage, the concept of diversity has been applied in the cybersecurity literature [88]. Randomization of software features has been used to thwart cyber attacks by increasing uncertainty towards a target system whose critical information was known to an attacker previously. The concept of moving target defense (MTD) [67, 107] has been proposed to change the attack surface in order to increase uncertainty and confusion for attackers and software diversity-based security mechanisms have also been used as part of MTD techniques.

Research has shown that software diversity is closely related to enhancing the immunization of a computer system that halts multiple outbreaks of malware infections simultaneously occurring with heterogeneous and sparse spreading patterns [118]. Hence, the rationale that software diversity reduces malware spreading is quite well known and has been validated for its effectiveness to some extent [64, 65]. This underlying philosophy encompasses a simple principle: *software polyculture enhances security* [64]. Due to the accessibility to the Internet, which enables the distribution of individualized software and cloud computing with the computational power to perform diversification, massive-scale software diversity is becoming a realistic and practical approach to enhance security [92]. In general, software diversity-based approaches have already been applied in various domains, such as operating systems [130], firewalls [103], intrusion detection systems [150], and malware detectors [? ]. In particular, a common example of software diversity is the use of various kinds of operating systems (OSs) as a MTD strategy, such as Linux-based OSs, Microsoft Windows-based OSs, FreeBSD-based OSs, Apple iOSs, Android OSs, or Apple macOSs in a given network [? ]. Although the benefit of software diversity seems obvious, the secure and transparent implementation of automatic software diversity is highly challenging [93]. In addition, no prior work has considered software diversity metrics as the basis to adapt a network topology to balance network connectivity and system security where each node's software vulnerability

is incorporated into estimating each node's software diversity.

In this work, we are interested in developing a software diversity metric to measure a node's software diversity based on software vulnerabilities of intermediate nodes on attack paths reachable to the node.

## 3.2 Problem Statement

In this work, we develop a software diversity metric for measuring a network topology in terms of minimizing security vulnerabilities against epidemic attacks (e.g., malware/virus spreading) while maintaining a sufficient level of network connectivity to provide seamless service availability. The proposed software diversity metric can be used to make decisions related to which two nodes should be disconnected or connected in order to construct an improved network topology meeting these two goals, minimizing security vulnerability and maximizing network connectivity. However, identifying the optimal network topology requires an exponential solution complexity [166]. In this work, we propose a heuristic method called software diversity-based adaptation (SDA) to generate a better network topology that is resilient against epidemic attacks with a sufficiently high network connectivity where the deployment cost is acceptable. We leverage percolation theory [118], which has been used to describe the process or paths of some liquid passing through a medium. We use site and bond percolation from this theory to model and analyze attack processes and defense or recovery processes. *Site percolation* (i.e., removing a node) [118] is used to model an attacker's behavior in compromising another node, wherein the node being percolated corresponds to the node being compromised (infected) by the attacker. This leads to the disconnection of all edges around the node to reflect its failure or its isolation by an intrusion detection system (IDS). *Bond percolation* is used to model the adaptation of edges between nodes such that

connected nodes with high security vulnerability (e.g., two connected nodes have the same software package installed or a neighbor node has high software vulnerability) are disconnected while disconnected nodes with low or no security vulnerability (e.g., two disconnected nodes using a different software with low software vulnerability) can be connected in a given network.

### 3.3 Key Contributions

We made the following **key contributions** in this work:

- This work is the first that takes a multidisciplinary approach by considering both the computer science’s software diversity to enhance cybersecurity and percolation theoretic network resilience techniques to study the effect of interconnectivity on network connectivity under epidemic attacks. To be specific, we develop network adaptation strategies that determine whether to add or remove edges between two nodes in a given network, aiming to minimize network vulnerabilities against epidemic attacks while maintaining maximum network connectivity. Given that each node is installed with a set of software (we call it a ‘software package’), we investigate network resilience and vulnerability depending on how a network topology is connected under epidemic attackers who can exploit the vulnerabilities based on their knowledge on software vulnerabilities.
- We develop a novel software diversity metric that measures a node’s software diversity level, representing both the vulnerabilities of attack paths reachable to the node and the network connectivity. To minimize computational complexity in estimating the node’s software diversity based on attack path vulnerability, we utilize the only the

$k$ -hop local neighborhood of the node. This approach provides a lightweight method to compute each node’s software diversity. To prove the effectiveness of this software diversity metric, we use it as the criterion to determine whether to add or remove an edge between two nodes.

- Although most software diversity-based network topology adaptations are studied by shuffling the types of software packages [166, 167], our work takes one step further by changing network topology, which is proven much more effective than its software shuffling counterpart (e.g., graph coloring) in reducing vulnerability to epidemic attacks while maximizing the network connectivity. Further, we broaden the concept of software diversity by both maintaining the use of different software in adjacent nodes and minimizing security vulnerability in each node’s ego network (i.e., local network within  $k$ -hop), which has not been considered in the state-of-the art. In addition, our proposed software diversity-based network adaptations are lightweight showing acceptable operational cost while achieving minimum security vulnerability and maximum network connectivity, which opens a door for the applicability in resource-constrained, contested network environments.
- We validate the outperformance of the proposed SDA strategy by conducting a comprehensive comparative performance analysis with the following six schemes (see Section 3.6.2): non-adaptation, random adaptation, graph-coloring, and three variants of the proposed SDA strategies. We analyze the effect of key design parameters such as network density, attack density, and the number of software packages available on four performance metrics (see Section 3.6.1), i.e., the size of the giant component, the fraction of undetected compromised nodes, software diversity levels, and defense cost (i.e., shuffling plus network topology adaptation costs). We validate the outperformance of our SDA scheme in three real network topologies covering dense (high), medium

dense, and sparse (low) networks [97]. Further, to profoundly understand the effect of various network characteristics, we conduct sensitivity analysis under a random graph using the Erdős-Rényi (ER) network model and analyze the results. Due to space constraints, we place these results for the ER network in Sections A.3.2–A.3.3 in the appendix.

We will discuss the answers to the research questions in Section 3.7 and summarize our findings in Section 3.8.

## 3.4 System Model

This section discusses our system model in terms of the network model, the node model, the attack model, and the defense model.

### 3.4.1 Network Model

In this work, we assume that our proposed network adaptation strategies are applicable in networks with multiple controllers that can govern a partition of nodes in the network. Typical examples include a software-defined network (SDN) where each node can be instructed by the one or more SDN controllers it is assigned to [89], an edge computing Internet-of-Things (IoT) system with some edge devices or nodes available to perform high computing tasks [101], a wireless sensor network with multiple cluster headers [82], and a hierarchical mobile ad hoc network with decentralized controllers in charge of governing a subset of the nodes [29]. In a reconfigurable network [37, 39, 52, 96, 133, 180? ], regional controllers (e.g., SDN controller(s), or any other network controller(s) in general) can change the flow table in the programmable switches to reconfigure a logical network topology.

Periodic information exchange between nodes and the regional coordinators is required to ensure seamless operations of the system. However, since each node’s software diversity value, which is used to make a decision on edge adaptation (i.e., adding/removing edges), is computed locally by each node, a regional coordinator only needs to rank the software diversity values of neighbor nodes around a target node, and inform the target node of which edges to add or remove based on the estimated ranks. Moreover, the ranking operation of the neighbor nodes around a target node is only periodically performed by a regional coordinator and it does not require high communication overhead for each node to communicate with the regional coordinator.

A temporal network is an undirected network for which the topology evolution (or change) occurs due to node failures or nodes being compromised by attackers. In addition, the network may change its topology when adaptation strategies are performed by connecting between two nodes or disconnecting all the edges associated with compromised nodes to mitigate the spread of infection over the network. We use an index to label a node (e.g., node  $i$ ) and characterize its attributes as described in Section 3.4.2.

An edge between nodes can be on and off depending on the dynamics caused by node failures, node recovery, or edge adaptations (i.e., an edge can be added or removed). We maintain an adjacency matrix  $\mathbf{A}$  in order to keep track of direct or indirect connectivities (i.e., edges) between nodes where  $a_{ij} = 1$  indicates there exists an edge between nodes  $i$  and  $j$  while  $a_{ij} = 0$  indicates that no edge exists.

In order for each node to efficiently estimate its software diversity by considering the vulnerabilities of attack paths reachable to the node, it only considers neighboring nodes within  $k$ -hop distance from itself. This  $k$ -hop local network is used for each node to estimate its software diversity value by considering the vulnerabilities of attack paths within its local network.

Although we utilize a sufficiently small value of  $k$  (e.g., 1 or 2), it does not underestimate the vulnerabilities of possible attack paths because using smaller  $k$  means that an attacker is nearby within the local network. For example, if an attacker wants to compromise a particular target node, it may try multiple attack paths where each attack path has a set of intermediate nodes. When the attack path is long, it means the vulnerability of the target node is low as the attacker needs to compromise all the intermediate nodes in order to finally compromise the target node. However, when the path length is small, it does not necessarily decrease the attack path vulnerability because the attacker is close to the target node.

We assume that software packages installed in each node and the associated vulnerabilities information are given to the regional coordinator in the initial network deployment period. In addition, we assume that each node is also well informed about the software vulnerability information associated with the software packages installed in the neighboring nodes in its  $k$ -hop local network. We assume that the changes of network topology are mainly made by node failures or network adaptations in this work.

Adding or removing an edge between two nodes requires secure communications between them. Even if they are within wireless range of each other but don't share a secret key for secure communications, they are not logically connected. In this work, generating an optimal network topology which is resilient against epidemic attacks with maximum network connectivity is based on a logical network topology.

### 3.4.2 Node Model

Each node  $i$  is characterized by its attributes as follows:

- Node  $i$ 's status on whether it is active or not (i.e., has sufficient energy and responsiveness regardless of being compromised or not), denoted by  $na_i$ , indicating whether

it is alive ( $= 1$ ) or not ( $= 0$ ), respectively;

- Node  $i$ 's status on whether the node is compromised ( $= 1$ ) by an epidemic attack or not ( $= 0$ ), denoted by  $nc_i$ ;
- Node  $i$ 's software package installed, representing the diversified package or the version of the same software providing the same functionality. In this work, we adopt the well-known software diversification approach called *N-version programming* [8, 9]. This concept means that a software has multiple independent implementations. While these different implementations of the software can still provide the same functionalities, since the implementations are different they naturally have different bugs or vulnerabilities. Following this concept, we model the node's software package installed, denoted by  $s_i$ , with a limited number of software packages available,  $N_s$ , so that  $s_i$  is an integer, ranged in  $[1, N_s]$ . The node's software package type stays the same during the adaptation process and our SDA algorithm aims to answer how to adjust edges between two nodes to minimize security vulnerability while maximizing network connectivity. However, a regional coordinator should have knowledge of the software package information of the nodes under its region during the deployment phase;
- Node  $i$ 's degree of software diversity,  $sd_i$ , whose physical meaning is how different node  $i$ 's software package is from its neighbors. The computation of node  $i$ 's software diversity are described in Eq. (3.4); and
- Node  $i$ 's software vulnerability ( $sv_i$ ) is the same as the software vulnerability of the software package (comprising multiple software products) installed in node  $i$ . The software vulnerability of software package  $s$ , denoted by  $vul_s$ , is estimated by:

$$vul_s = 1 - \prod_{k \in S} (1 - v_k), \quad (3.1)$$

where  $S$  refers to a set of software products in software package  $s$ ,  $k$  refers to the  $k$ th software product in  $S$ , and  $v_k$  is vulnerability of the  $k$ th software product estimated based on CVSS scores divided by the maximum CVSS value (i.e., 10).  $vul_s$  is calculated as above because software package  $s$  is vulnerable when any single software product contained within is vulnerable. Suppose node  $i$  is installed with software package  $s_i$ . Then, node  $i$ 's software vulnerability ( $sv_i$ ) is equal to the software vulnerability of software package  $s_i$  ( $vul_{s_i}$ ).

Based on the above five attributes, node  $i$  is characterized by:

$$\mathbf{node}(i) = [na_i, nc_i, s_i, sd_i, sv_i]. \quad (3.2)$$

If attacker  $j$  targets vulnerable node  $i$  (i.e., a node that has not been compromised before), which is one of its direct neighbors, the probability that node  $j$  infects node  $i$ , denoted by  $\beta_{ji}$ , is estimated based on the probability that node  $j$  can exploit the vulnerability of node  $i$ 's software package,  $s_i$ . We estimate this probability based on node  $i$ 's vulnerability to node  $j$ , estimated by [64]:

$$\beta_{ji} = \begin{cases} 1 & \text{if } \sigma_j(s_i) > 0; \\ sv_i & \text{otherwise,} \end{cases} \quad (3.3)$$

where  $\sigma_j$  is a vector of software packages attacker  $j$  has learned about their security vulnerabilities. For example, attacker  $j$  knows the vulnerabilities of software packages 1 and 3 among 5 packages available. It is denoted by  $\sigma_j = [1, 0, 1, 0, 0]$ . In this case, the sum of  $\sigma_j$  indicates the total number of software packages for which attacker  $j$  knows the security vulnerabilities and so can exploit. Note that it is a dynamic value learned after node  $j$  compromises node  $i$  via reconnaissance even if their installed software packages are different, i.e.,  $s_i \neq s_j$ . Here  $sv_i$  refers to the vulnerability of software package  $s_i$ , which can be estimated

based on the degree of a Common Vulnerabilities and Exposures (CVE) with a Common Vulnerability Scoring System (CVSS) severity score [1, 53].

### 3.4.3 Attack Model

This work deals with two stages of attack behaviors: An outside attacker before the node is compromised and an inside attacker after the node is compromised but undetected.

**(1) Node Compromise by Epidemic Attacks:** We consider the so called *epidemic attack* that describes an attacker’s infection behavior based on an epidemic model, called the SIR (Susceptible-Infected-Removed) model [118]. That is, an outside attacker can compromise the nodes directly connected to itself, its direct neighbors, without access rights to their settings or files. Typical example scenarios include the spread of malwares or viruses. Botnets can spread malwares or viruses via mobile devices. A mobile device can misuse a mobile malware, such as a Trojan horse, thus acting as a botclient to receive commands and controls from a remote server [110]. Further, worm-like attacks are popular in wireless sensor networks where the sensor worm attacker sends a message to exploit the software vulnerability in order to cause a crash or take control of sensor nodes [166, 167]. Attacker  $j$  can compromise its direct neighbor  $i$  when node  $i$  uses a software package that attacker  $j$  can exploit because the attacker knows the vulnerability of the software package. This case happens when  $s_i$  is the same as  $s_j$  or attacker  $j$  learned  $s_i$ ’s vulnerability in the past (i.e.,  $\sigma_j(s_i) > 0$ ). When attacker  $j$  is installed with a particular software package,  $s_j$ , we assume that attacker  $j$  knows the vulnerability of its own software package,  $s_j$ . Attacker  $j$  can learn the vulnerabilities of other software packages although it needs to commit more time and resources to obtain the information of their security vulnerabilities. Node  $i$ ’s vulnerability by attacker  $j$  based on these two cases is reflected in Eq. (3.3). When node  $i$  is compromised, node  $i$ ’s status

is changed from ‘susceptible’ to ‘infected’ indicating that node  $i$  is now an attacker. Then, node  $i$  can infect other nodes and learn their software vulnerabilities, which are unknown to it. The attack procedures are described in Algorithm 10 in the appendix.

**(2) Malicious Behavior of Compromised Nodes Undetected by the IDS:** Even if an intrusion detection system (IDS) is assumed to be placed in this work, an attacker may not be detected by the IDS and the inside attacker can perform malicious behaviors such as packet dropping attacks (e.g., gray or black hole attacks), data exfiltration attacks, or denial-of-service (DoS) attacks to compromise the security goals in terms of loss of confidentiality, integrity, and availability [43, 161].

## 3.5 Proposed Approach: Software Diversity-based Adaptation (SDA) Algorithm Design

In this section, we describe our proposed software diversity based adaptation (SDA) algorithm design in detail. SDA uses software diversity as a key determinant to select edges to percolate (i.e., add or remove) for mitigating the spreading of compromised nodes by attackers and also to maximize the network connectivity for network resilience.

### 3.5.1 Software Diversity Metric

A node’s vulnerability is commonly computed based on the software package installed [64, 65]. However, if the node is connected with many other nodes that are directly or indirectly connected, its potential vulnerability is not simply restricted by the vulnerability of its software package. We use a broader concept of node vulnerability by incorporating the vulnerabilities of attack paths reachable to each node. To better capture the relationship

between node vulnerability and network topology, we utilize an attack path  $AP$  an attacker can take to successfully compromise a target node. That is, in order to compromise the target node, the attacker needs to compromise all intermediate nodes on the attack path. Hence, we estimate each node's software diversity value as the probability that a node is robust against vulnerabilities from attack paths  $APs$  reachable to the node.

To this end, we consider the shortest paths (i.e., maximum  $k$ -hop distance paths) between boundary nodes (i.e., nodes in the boundary of a target node's local network) to a target node as attack paths. In addition, to reduce the complexity of measuring each node's software diversity, we use a limited number of attack paths, denoted by  $l$ , where each path has at most  $k$ -hop distance. Target node  $i$ 's software diversity based on  $l$  attack paths within  $k$ -hop distance from node  $i$ , denoted by  $sd_i(k, l)$ , is defined by:

$$sd_i(k, l) := \prod_{j \in \mathbf{ap}_i}^l (1 - apv_{ij}^k), \quad (3.4)$$

where  $\mathbf{ap}_i$  is a set of attack paths available to node  $i$  ranked based on their highest vulnerability and  $apv_{ij}^k$  is the vulnerability of the attack path  $j$  to node  $i$  with maximum hop distance  $k$ . In order to consider the maximum number of nodes associated with the attack paths, we consider disjointed attack paths (i.e., the  $j$ 's in  $\mathbf{ap}_i$ ) from the boundary nodes to node  $i$ .

### 3.5.2 Software Diversity-based Bond Percolation for Network Adaptation

The design objective of SDA is to decide which edges to add or remove in order to maximize the size of the giant component (i.e., the largest network cluster in a network) for maintain-

**Algorithm 1** Software Diversity-based Adaptation (SDA)

---

```

1:  $N \leftarrow$  The total number of nodes in a network
2:  $\mathbf{DN} \leftarrow$  A vector containing the number of removed edges per node
3:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
4:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
5:  $\mathbf{SV} \leftarrow$  A vector of the vulnerabilities associated with software packages
6:  $k \leftarrow$  A hop distance given in a node's local network
7:  $l \leftarrow$  A maximum number of attack paths considered for estimating a node's software diversity
8:  $\rho \leftarrow$  A threshold referring to the fraction of edges to be removed when  $\rho < 0$  and added when  $\rho > 0$ 
9:  $\mathbf{A}' \leftarrow$  An adjacency matrix after edges are adapted in Step 1
10:  $\mathbf{A}'' \leftarrow$  An adjacency matrix after edges are adapted in Step 2
11:
12:  $\mathbf{A}'' = \mathbf{SDA}(\mathbf{DN}, \mathbf{A}, \mathbf{S}, \mathbf{SV}, k, l, \rho)$ 
13:
14: Step 1:  $\mathbf{A}' = \mathbf{SDBA}(\mathbf{DN}, \mathbf{A}, \mathbf{S})$   $\triangleright$  Remove edges between two nodes with the same software package based on Algorithm 3 in the appendix).
15:
16: Step 2: Add or remove edges locally based on the ranks of the software diversity differences estimated in Eqs. (3.5) and (3.7) (Algorithms 6 and 7 in the appendix)
17:  $\mathbf{SD} \leftarrow$  A vector of software diversity where each element,  $sd_i(k, l)$ , refers to node  $i$ 's software diversity value when at most  $l$  number of attack paths are considered where each attack path has at most  $k$ -hop length.
18:  $\mathbf{PV} \leftarrow$  A vector of estimated maximal attack path vulnerabilities associated with each node.  $\triangleright$  Algorithm 4 in the appendix.
19: candidate  $\leftarrow$  A set of edge candidates  $\triangleright$  Algorithms 6 and 7 in the appendix.
20:  $\mathbf{T}^{local}, \mathbf{T}^{global} = \mathbf{setEAB}(\mathbf{DN}, \mathbf{A}', \rho)$   $\triangleright$  Set edge adaptation budget based on Algorithm 5 in the appendix.
21: if  $\rho > 0$  then
22:   candidate =  $\mathbf{GEAC}(\mathbf{A}', \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local})$ 
23:    $\triangleright$  Algorithm 6 in the appendix file.
24: else
25:   candidate =  $\mathbf{GERC}(\mathbf{A}', \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local})$ 
26:    $\triangleright$  Algorithm 7 in the appendix file.
27: end if
28:  $\mathbf{A}'' = \mathbf{AdaptNT}(\mathbf{A}', \mathbf{candidate}, \mathbf{T}^{local}, \mathbf{T}^{global}, \rho)$ 
29:    $\triangleright$  Algorithm 8 in the appendix file.
30: return  $\mathbf{A}''$ 

```

---

ing network connectivity and to minimize the fraction of nodes being compromised due to epidemic attacks with minimum defense cost defined in Section 3.6.1.

We have two tasks to determine which edges to remove or add as follows:

1. Estimate the gain or loss as a result of removing or adding an edge. This is determined from the difference between a node's current software diversity value and its expected software diversity value if the edge adaptation is made between nodes  $i$  and  $j$ . To determine if adding an edge between nodes  $i$  and  $j$  is beneficial, we compute the software diversity (SD) difference by comparing the SD before and after edge adaptations between nodes  $i$  and  $j$ :

$$SD_{\text{diff}}^A(i, j) = (sd_i - sd'_i) + (sd_j - sd'_j), \quad (3.5)$$

where, for conciseness,  $sd_i = sd_i(k, l)$  and  $sd_j = sd_j(k, l)$ , which are defined in Eq. (3.4).  $sd'_i$  and  $sd'_j$  are the expected software diversity values of nodes  $i$  and  $j$  after an edge is added. The most promising candidate edge to be added should be an edge with the lowest  $SD_{\text{diff}}^A(i, j)$ . The expected software diversity value of node  $i$  after addition of an edge with node  $j$  is simply obtained by

$$sd'_i = sd_i(1 - sv_i \cdot pv_j), \quad (3.6)$$

where  $sv_i$  is the software vulnerability of the software package installed in node  $i$  (i.e.,  $s_i$ ) and  $pv_j$  is the estimated attack path vulnerability  $apv$  of an attack path from node  $j$  to a boundary node in node  $j$ 's local network. That is,  $sv_i pv_j$  is the estimation of attack path vulnerability  $apv$  of an attack path from node  $i$  to a boundary node in node  $i$ 's local network through node  $j$ .  $sd'_j$  is similarly obtained. To determine if removing the edge between nodes  $i$  and  $j$  is beneficial, we compute the software diversity difference by:

$$SD_{\text{diff}}^R(i, j) = (sd'_i - sd_i) + (sd'_j - sd_j), \quad (3.7)$$

where now  $sd'_i$  is computed by:

$$sd'_i = sd_i / (1 - sv_i \cdot pv_j). \quad (3.8)$$

Here the division by  $(1 - sv_i pv_j)$  represents the extent of reducing the vulnerability by removing an edge between nodes  $i$  and  $j$  based on Eq. (3.4).  $sd'_j$  is similarly obtained. The most promising candidate edge to be removed should be an edge with the highest  $SD_{\text{diff}}^R(i, j)$ . See Algorithm 6 (Generates Edge Addition Candidates or GEAC) and Algorithm 7 (Generates Edge Removal Candidates or GERC) in the appendix for details on how we generate edge candidates for edge addition and removal, respectively.

2. Estimate how many edges each node can adapt, i.e., remove or add. Based on the rationale that high centrality nodes (e.g., high degree) may expose high vulnerability in terms of security and network connectivity, we minimize the difference between the maximum degree and minimum degree by adding more edges to nodes with lower degree while deleting edges to the nodes with higher degree. Based on this principle, we develop a heuristic method to estimate how many edges should be adapted per node. See Algorithm 5 (Set Edge Adaptations Budget or SetEAB) in the appendix for details.

Algorithm 1 details our proposed software diversity-based adaptation (SDA) algorithm. In Step 1, it executes SDBA (see Algorithm 3 of Appendix A.1) to remove edges between two nodes with the same software package. Then, it makes the decision to add or remove edges locally based on the ranking of software diversity differences estimated in Step 2 using Eqs. (3.5) and (3.7), with the objective to best satisfy both security vulnerability and network connectivity requirements. It first sets up the edge adaptation budget based on Algorithm 5 in Appendix A.1. Then, the SDA generates edge adaptation candidates based on the ranking

of software diversity differences (see Algorithms 6 and 7 of Appendix A.1). As the last step, the SDA adapts the network topology based on edge adaptation candidates and network topology constraints (see Algorithm 8 of Appendix A.1).

Removing an attack path may increase a chance for attackers to use other attack paths. However, this can make the average vulnerability of existing attack paths significantly plummet because the attack paths are not easily exploitable by the attacker. In addition, we choose adjusting edges (removing or adding) to achieve both security and performance goals instead of using firewalls because networks could face significant performance degradation with high security level firewalls [104].

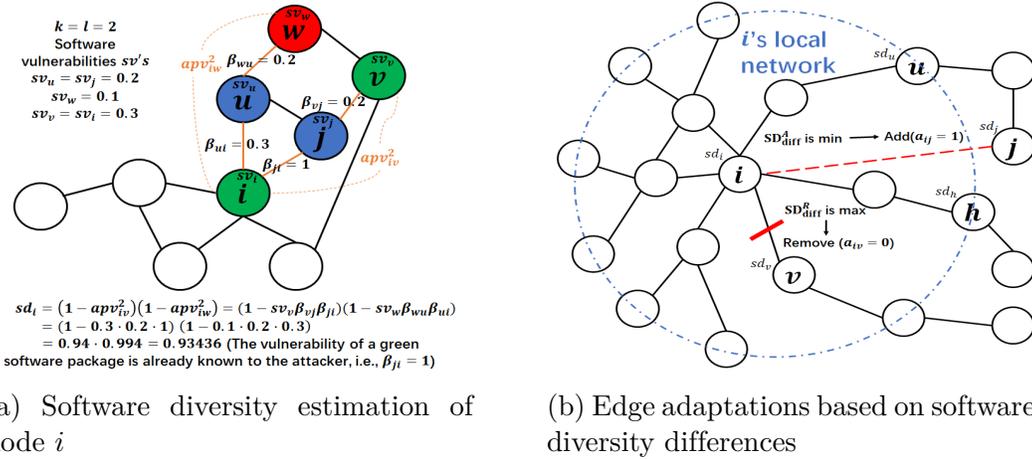


Figure 3.1: Example of the software diversity-based adaptation strategies: (a) The estimation of node  $i$ 's software diversity value; and (b) The edge adaptation based on the software diversity difference in Eqs. (3.5) and (3.7).

Fig. 3.1 illustrates the SDA algorithm execution with an example network where distinct software packages are marked with distinct colors. Fig. 3.1 (a) illustrates how node  $i$  estimates its software diversity value when  $k = l = 2$ . Fig. 3.1 (b) illustrates how node  $i$  determines whether to add or remove edges based on the software diversity differences,  $SD_{diff}^A$  and  $SD_{diff}^R$ , based on Eqs. (3.5) and (3.7), respectively.

### 3.5.3 Practical Operations of the SDA Algorithm

Several practical real-world examples of network topology adaptation (by reconfiguration) are given below. In an SDN, since its key merit is flexible manageability that can separate data plane from control plane, an SDN controller has been commonly used to reconfigure a logical network topology in its flow table so that packets can be forwarded based on routing instructions given from the SDN controller at node levels [? ]. Generating virtual network topologies, called “virtual topology design”, in optical networks is well-known to optimize service provision [52, 180]. In wireless sensor networks, network topology reconfiguration has been frequently considered for accurate estimates of sensed data by sensors where a gateway provides each node its next node to which a packet is forwarded [39, 96]. Moreover, by opening sectionalizing and closing tie switches of the network, power distribution systems perform efficient and effective network reconfigurations to minimize their power loss [37, 133].

An example of day-to-day operations can include network configurations, parameter updates, and maintenance operations upon attacks or outages when the proposed SDA algorithm is applied in a given network. Since there is a potential for service degradation while SDA is actively being applied, there is a tradeoff between the frequency of implementation of SDA thereby enhancing network survivability and the service performance of the system. We currently envision an infrequent active implementation to limit any effect on network service during the execution of the adaptations.

These operations can be performed based on the following procedures: (1) *Network Configurations*: A network needs to be configured with the key design parameters that the SDA algorithm requires. For instance, we need to configure the values of key parameters (see Table 3.1) affected by the network density and system constraints of the current state of the network. (2) *Parameter Updates*: As network and environmental conditions may vary due

Table 3.1: Key design parameters, their meanings, and their default values.

Param.	Meaning	Value
$N$	Total number of nodes in a network	1000
$p$	Connection probability between pairs of nodes in a ER network	0.025
$\gamma$	Intrusion detection probability	0.95
$k$	The upper bound of hops considered in calculating software diversity $SD_{k,l}^i$	[1,2]
$l$	The upper bound of # of paths considered in calculating software diversity $SD_{k,l}^i$	1
$n_r$	Number of simulation runs	100
$N_s$	Number of software packages available	[3,7]
$P_a$	Percentage of attackers in a network	[10,30]
$\rho$	Threshold of fraction of edges adapted	[-1, 1]
<b>SV</b>	A vector of vulnerabilities associated with software packages which are selected based on the uniform distribution with the range in (0, 0.5] (i.e., $U(0, 0.5]$ ). For the maximum 7 different software packages, the <b>SV</b> of the corresponding vulnerabilities are used.	$\begin{bmatrix} 0.41 \\ 0.35 \\ 0.48 \\ 0.22 \\ 0.16 \\ 0.19 \\ 0.12 \end{bmatrix}^T$

to network topology changes (e.g., node mobility or failure) or attacks, each node needs to calculate its software diversity value based on the change and dynamics periodically. We assume that each node's software diversity value will be updated upon an event or time interval; (3) *Maintenance*: Each node will inform its software diversity value to the regional coordinator periodically. We assume that all network configuration information is backed up and can provide redundancy to maintain reliability and resilience. Under some situations caused by power outage, operational failures, or successful internal and external attacks, the backup information of the network configuration is used. Since each node performs periodic calculation of its software diversity value based on the changed network conditions and relays this value to the regional coordinator, no additional overhead is generated.

## 3.6 Experimental Setup

In this section, we describe the performance metrics, the counterpart baseline schemes against which our proposed SDA algorithm (i.e., Algorithm 1) is compared for performance comparison, and the simulation environment setup for performance evaluation.

### 3.6.1 Performance Metrics

We use the following performance metrics:

- **Software diversity ( $SD$ ):** This metric measures the mean software diversity for all nodes in a network. Since node  $i$ 's software diversity, i.e.,  $sd_i$ , is computed based on Eq. (3.4), the mean software diversity for all nodes in the network is obtained by:

$$SD = \frac{\sum_{i=1}^N sd_i}{N}. \quad (3.9)$$

Recall that  $k$  is used to determine node  $i$ 's local network and thus is the maximum possible hop distance from node  $i$  to all other neighboring nodes in its local network. Higher software diversity is more desirable to ensure high system security.

- **Size of the giant component ( $S_g$ ):** This metric captures the degree of network connectivity composed of non-compromised (uninfected), active nodes in a network.  $S_g$  is computed by:

$$S_g = \frac{N_g}{N}, \quad (3.10)$$

where  $N$  is the total number of nodes in the network and  $N_g$  is the number of nodes in the giant component. Higher  $S_g$  is more desirable, implying higher network resilience in the presence of epidemic attacks.

- **Fraction of compromised nodes ( $P_c$ ):** This metric measures the fraction of the number of compromised nodes due to epidemic attacks over the total number of nodes in a network. This includes both currently infected (not detected by the IDS) and removed (previously infected and detected by the IDS) nodes.  $P_c$  is computed by:

$$P_c = \frac{N_c}{N}, \quad (3.11)$$

where  $N_c$  represents the total number of compromised nodes after epidemic attacks on a network (i.e., the original network under No-Adaptation and an adapted network under all adaptation schemes). See Section 3.6.2 for a listing of counterpart baseline schemes against which our proposed SDA algorithm is compared for a comparative performance analysis.

- **Defense cost ( $D_c$ ):** This metric measures the defense cost associated with the following defense strategies employed by an adaptation scheme: (1) edge adaptations (i.e., adding or removing edges) to isolate detected attackers (or compromised nodes) by the IDS; (2) edge adaptations to maximize software diversity by each node based on the value of the software diversity metric in Eq. (3.4); and (3) shuffling operations based on the fraction of nodes whose software package is randomly shuffled over the total number of nodes.  $D_c$  is computed by:

$$D_c = \frac{\text{sum}(|\mathbf{A} - \mathbf{B}|)}{\text{sum}(\mathbf{A} + \mathbf{B})} + \frac{N_{SF}}{N} \quad (3.12)$$

In the first term, the numerator refers to the differences of edges between the adjacency matrix of an original network  $\mathbf{B}$  and that of an adjusted network  $\mathbf{A}$  after edges adaptations are made. The denominator is the sum of the addition of the two matrices. In the second term,  $N_{SF}$  is the number of nodes whose software packages are shuffled and

$N$  is the total number of nodes. Note that when a node’s software package is shuffled but stays with its original software package, it is excluded from counting toward  $N_{SF}$ . This shuffling cost is estimated only when shuffling a software package is used such as random graph coloring, which is compared against our proposed SDA scheme in our work. Lower defense cost is more desirable.

### 3.6.2 Counterpart Baseline Schemes for Performance Comparison

In this work, we compare the performance of our proposed SDA scheme against No-adaptation (No-A), Random adaptation (Random-A), and Random graph coloring (Random-Graph-C) counterpart baseline schemes for a comparative performance analysis.

Our SDA scheme uses the software diversity-based metric in Eq. (3.4) to select an edge to remove or add based on the concept of bond percolation, as discussed in Section 2.1.1. To be specific, SDA first removes all edges between two connected nodes with the same software package as shown in Step 1 of Algorithm 1 (i.e., executing Algorithm 3 in the appendix file). Then SDA decides a set of edges to be added or removed given  $\rho$  (the percentage of edges to be added if  $\rho > 0$  or to be removed if  $\rho < 0$ ) as shown in Step 2 of Algorithm 1. The effect of  $\rho$  on performance will be analyzed in Section 3.6.3 to identify the optimal  $\rho$  value that can best balance security and network connectivity. We experiment with various  $\rho$  values in the range of  $[-1, 1]$  where  $-1$  means removing all edges (such that no edges exist in the network) and  $1$  means fully restoring edges removed from Step 1. For example, SDA with  $\rho = 1$  means fully restoring edges lost from Step 1 while SDA with  $\rho = 0$  refers to only executing Step 1 (removing edges between two nodes with the same software package). SDA with  $\rho = 0.6$  means only restoring 60% of the edges lost in Step 1 while SDA with  $\rho = -0.6$  means removing 60% of edges in the network after Step 1. What edges to remove or add

(see Step 2 of Algorithm 1) significantly affects network security and resilience.

Below we briefly discuss the three counterpart baseline schemes to be compared against our proposed SDA schemes:

- **No-adaptation (No-A):** This represents the case in which no adaptation is applied, thus showing the effect of attacks on the performance of the original network. However, we allow an IDS to detect attackers. When the IDS detects compromised nodes with probability  $\gamma$ , all edges connected to the detected attacker will be disconnected in order to isolate the attackers, ultimately resulting in mitigating the spread of compromised nodes in the network. Therefore, when No-A is used, the adaptation cost can be high because the number of edges disconnected is affected by the network topology, which is one of the key factors impacting the degree of network vulnerability.
- **Random adaptation (Random-A):** This scheme first removes an edge between two nodes with the same software package (i.e., executing Algorithm 3 in the appendix file) and then randomly adds edges between nodes with a different software package (see Algorithm 9 in the appendix file). In this scheme, we add the same number of edges lost due to the execution of Step 1.
- **Random graph coloring (Random-Graph-C):** This scheme uses a simple rule for each node to shuffle its software package with the least common software package without changing any network topology. As a special case, when a node has many neighbors, it may choose the least common software package of those used among its neighbors. It may occur that a node shuffles to its original software package. In such a case, when the shuffled software package is the same as the original software package, we do not count it toward the shuffling cost in Eq. (3.12). We treat this scheme as an adaptation scheme because it also involves changing a configuration of its software by

using a different implementation although it does not make any change to the network topology.

The pseudocode for SDA is presented in Algorithm 1 and that for Random-A is described in Algorithm 9 in the appendix. In our experiment, we compare the performance of No-A, Random-A, Random-Graph-C, and three variants of SDA with three different thresholds  $\rho$  in terms of the 4 performance metrics discussed in Section 3.6.1. We treat Random-A, Random-Graph-C, and the SDA schemes as adaptation schemes while No-A is treated as a baseline scheme without adaptation.

### 3.6.3 Environment Setup

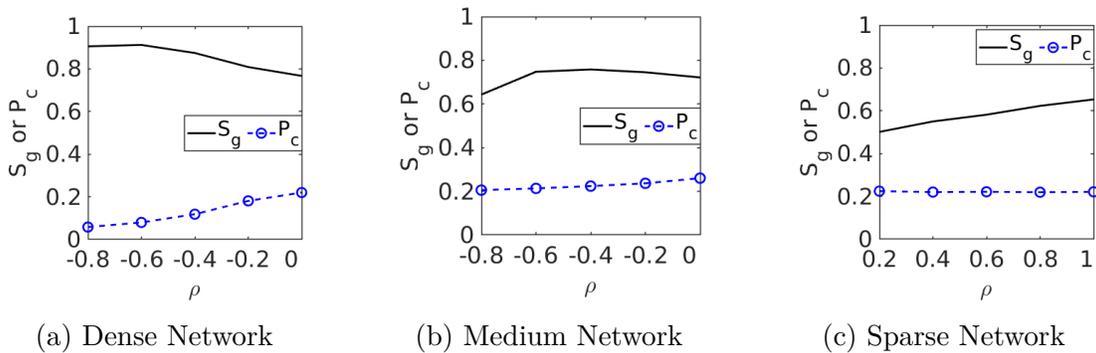


Figure 3.2: Effect of  $\rho$  (fraction of edges to be adapted) on performance of SDA in terms of the size of the giant component ( $S_g$ ) and the fraction of compromised nodes ( $P_c$ ). The optimal  $\rho$  for the SDA scheme with respect to  $S_g$  and  $P_c$  in dense, medium dense, and sparse networks are identified as  $\rho = -0.6$ ,  $\rho = -0.4$  and  $\rho = 1$ , respectively.

### Parameters and Data Collection

Table 3.1 summarizes the key parameters, their meanings, and their default values used in this work. We use the average of the performance measures collected based on 100 simulation runs. In the experiment, we examine the effect of the following key design parameters on

performance: (1) attack density (i.e., percentage of attackers); and (2) the number of software packages available. For the ER network, we also study the effect of the network connection probability on performance in Appendix A.3.3.

### Network Topology Datasets

We setup 4 different undirected networks to evaluate the proposed work: (1) a sparse network from an observation of the Internet at the autonomous systems level [97]; (2) a medium dense network derived from an Enron email network [97]; (3) a dense Facebook ego network [97]; and (4) an Erdős-Rényi (ER) random network [118]. The network topologies and their degree distributions are shown in Figs. A.1 and A.2. Except for the medium dense network, we use the original network topologies. For the medium dense network, in order to derive a network of comparable size with the other networks (the Enron email network has 36,692 nodes and 183,831 edges) we generate the medium dense network with 985 nodes and 7,994 edges using the following procedure: (i) Rank all nodes in the Enron email network by degree in descending order; (ii) identify the medium dense network as the largest connected component of the induced subgraph consisting of nodes with ranks from 501 to 1500.

### Optimal Parameter Settings Used for SDA

**Fraction of edges to be adapted ( $\rho$ ):** We have conducted a sensitivity analysis of  $\rho$  for the SDA scheme in terms of maximizing the size of the giant component ( $S_g$ ) for network resilience without overly increasing the fraction of compromised nodes ( $P_c$ ) for network security. As shown in Fig. 3.2, the optimal  $\rho$  for the SDA scheme with respect to  $S_g$  and  $P_c$  in dense, medium dense, and sparse networks have been identified as  $\rho = -0.6$ ,  $\rho = -0.4$  and  $\rho = 1$ , respectively. Due to space constraints, we have conducted the sensitivity analysis of

$\rho$  for the ER random network in Appendix A.3.2, from which we have observed the optimal  $\rho$  with respect to  $S_g$  and  $P_c$  for the ER random network is  $-0.6$ . In summary, the optimal values of  $\rho$  are observed at  $-0.6$ ,  $-0.4$ ,  $1$ , and  $-0.6$  for dense, medium dense, sparse, and ER random networks, respectively.

**The number of maximum attack paths ( $l$ ) and the maximum hop distance in each attack path ( $k$ ):** The network type (i.e., dense, medium dense, sparse, or ER random) affects node density which in turn can affect the optimal setting of  $l$  and  $k$  under which SDA can best achieve both security (i.e., a low fraction of compromised nodes) and network resilience (i.e., a large size of the giant component). We have conducted a sensitivity analysis of  $l$  or  $k$  on the performance of the SDA scheme in all four types of networks. Due to space constraints, we put the sensitivity analysis of  $l$  and  $k$  on performance of SDA in Appendix A.4 and A.5. In summary, for dense, medium dense, and ER random networks, we have selected  $k = 1$  and  $l = 1$  to calculate software diversity  $sd_i(k, l)$  for each node in the network because we have observed no significant performance improvement with  $k > 1$  and  $l > 1$ . For the sparse network, we have not observed high sensitivity when  $l > 1$ . However, for  $k$ , we have observed that SDA performs the best when  $k = 2$  with  $\rho = 1$ . Thus, we have selected  $k = 2$  and  $l = 1$  for the sparse network.

Although network connectivity during the edge adaptation process is highly sensitive to the network density (i.e., the number of edges), a network can be either more or less vulnerable to the epidemic attacks than another network even when both have the same density due to differences in the topology. Since our proposed SDA algorithm has both the removing-only phase and the recovery phase that can adapt edges based on software diversity, it can meet the goals of maximizing network connectivity and minimizing security vulnerability in the network.

## 3.7 Numerical Results & Analysis

In this section, we present the experimental results for a comparative performance analysis of the proposed SDA scheme against the counterpart baseline schemes and provide physical interpretations of the results. In our experiment, we compare 6 schemes: (1) Non-adaptation (No-A); (2) Random adaptation (Random-A); (3) Random graph coloring (Random-Graph-C); (4) SDA with  $\rho = 0$ ; (5) SDA with  $\rho = 1$ ; and (6) SDA with optimal  $\rho$ . See Section 3.6.2 for more detail on how each scheme is implemented. The 6th scheme “SDA with optimal  $\rho$ ” is network-type dependent. As discussed earlier in Section 3.6.3, the optimal values of  $\rho$  are observed at  $-0.6$ ,  $-0.4$ ,  $1$ , and  $-0.6$  for dense, medium dense, sparse, and ER random networks, respectively.

Initially a set of attackers is randomly and uniformly distributed to the network based on the percentage of attackers parameter  $P_a$  and all such attackers perform epidemic attacks as described in Section 3.4.3. See Algorithm 10 in the appendix for detail on how the attackers perform epidemic attacks. Below we only report the experimental results under dense, medium dense, and sparse networks. The experimental results under the ER random network are reported in Appendix A.3.3.

### 3.7.1 Comparative Performance Analysis under a Dense Network

#### Effect of Varying the Fraction of Initial Attacks ( $P_a$ )

Fig. 3.3 shows the effect of varying the attack density ( $P_a$ ) on the performance of the six schemes in terms of the four metrics in Section 3.6.1 under the dense network, whose network topology and degree distribution are shown in Fig. A.1 (a) in the appendix. We observe that increasing the percentage of attackers ( $P_a$ ) decreases software diversity ( $SD$ ) and the size of

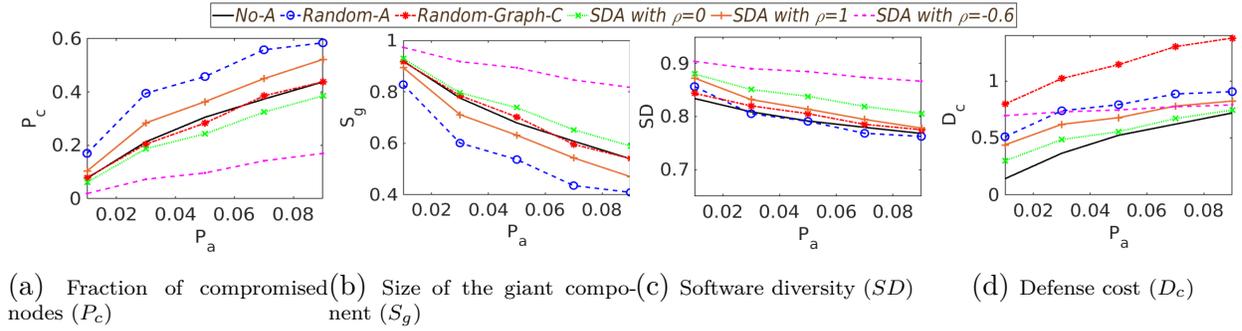


Figure 3.3: Effect of varying the fraction of attackers ( $P_a$ ) under a dense network.

the giant component ( $S_g$ ) while increasing the percentage of compromised nodes ( $P_c$ ) and the defense cost ( $D_c$ ). We note that when more nodes are compromised, the defense cost would also increase since it requires more site percolation based adaptations to be performed when compromised nodes are detected by the IDS (i.e., for disconnecting all edges of a detected, compromised node).

The overall performance order with respect to  $P_c$  (representing network security) and  $S_g$  (representing network connectivity and resilience) is observed as: SDA with optimal  $\rho$  (set at  $-0.6$ )  $\geq$  SDA with  $\rho = 0 \geq$  Random-Graph-C  $\approx$  No-A  $\geq$  SDA with  $\rho = 1 \geq$  Random-A. It is apparent that the network density of a given network significantly affects both security and performance since SDA with  $\rho = -0.6$  and SDA with  $\rho = 0$  have relatively fewer edges after adaptation and perform better than the other schemes in terms of  $P_c$ ,  $S_g$  and  $SD$  (i.e., the average software diversity level), as shown in Figs. 3.3 (a)-(c).

In Fig. 3.3 (d), SDA with optimal  $\rho$  (set at  $-0.6$ ) also shows significant resilience with relatively low defense cost ( $D_c$ ) as  $P_a$  increases. The overall performance order for the other five schemes in  $D_c$  is: Random-Graph-C  $\geq$  Random-A  $\geq$  SDA with  $\rho = 1 \geq$  SDA with  $\rho = 0 \geq$  No-A. Not only do the SDA schemes outperform the counterpart baseline schemes in  $P_c$ ,  $S_g$ , and  $SD$ , but also the defense cost of SDA schemes are significantly lower than that of Random-Graph-C and are comparable with Random-A (e.g., compared to SDA with

optimal  $\rho = -0.6$ ) and No-A (e.g., compared to SDA with  $\rho = 0$ ). This is a significant merit as SDA-based schemes outperform the counterpart baseline schemes with relatively low defense cost.

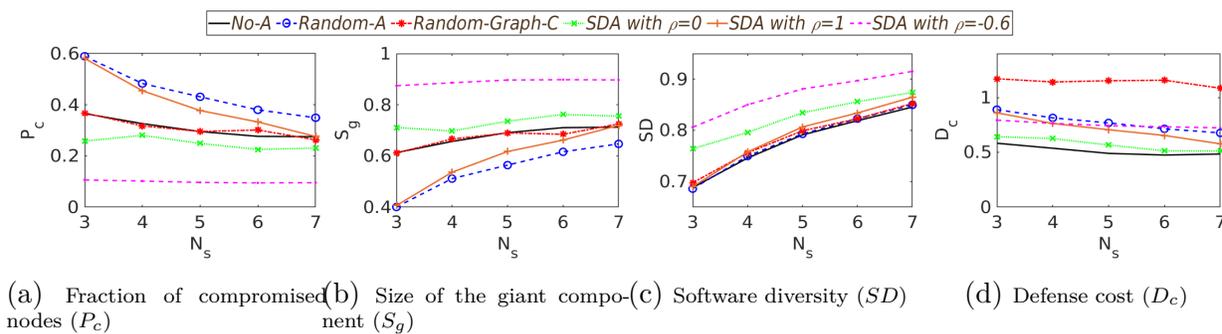


Figure 3.4: Effect of the number of software packages ( $N_s$ ) under a dense network.

### Effect of Varying the Number of Software Packages ( $N_s$ )

Fig. 3.4 shows the effect of varying the number of software packages available ( $N_s$ ) on the performance of the six schemes with respect to the metrics defined in Section 3.6.1 under the dense network. We observe that increasing the number of software packages available ( $N_s$ ) increases software diversity ( $SD$ ) and the size of the giant component ( $S_g$ ) while decreasing the percentage of compromised nodes ( $P_c$ ) and the defense cost ( $D_c$ ). Note that based on the concept of  $N$ -version programming, the number of software packages ( $N_s$ ) here refers to the number of versions being implemented for the same piece of software. Hence, as  $N_s$  increases, the software diversity strength increases, resulting in a decrease of the percentage of nodes being compromised due to attacks, an increase of the network connectivity, and a decrease of the defense cost because less nodes are being compromised.

The overall performance order in  $P_c$ ,  $S_g$ , and  $SD$  is very similar to what we observed in Fig. 3.3, with SDA with optimal  $\rho = -0.6$  outperforming all other schemes. For  $D_c$ , SDA with optimal  $\rho = -0.6$  generates a defense cost comparable to that generated by Random-A

and in-between those generated by No-A (lowest cost) and Random-Graph-C (highest cost).

### 3.7.2 Comparative Performance Analysis Under a Medium Network

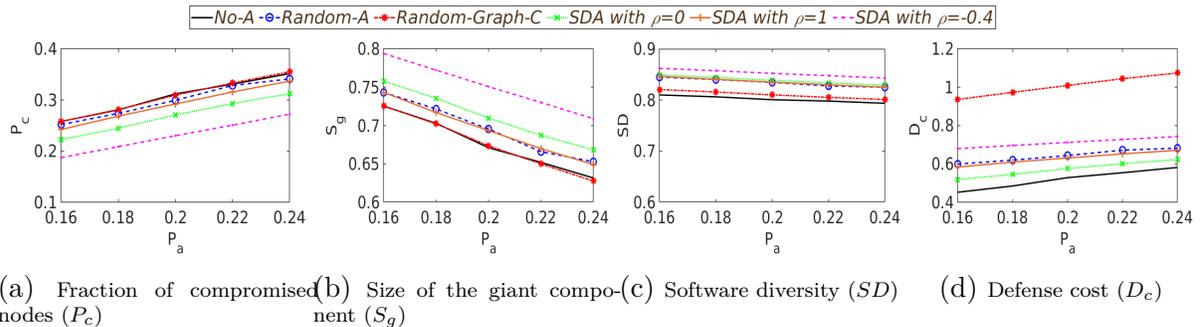


Figure 3.5: Effect of varying the fraction of attackers ( $P_a$ ) under a medium network.

#### Effect of Varying the Fraction of Initial Attacks ( $P_a$ )

Fig. 3.5 demonstrates the effect of varying the percentage of initial attacks on metrics defined in Section 3.6.1 under the medium network, whose network topology and degree distribution are shown in Fig. A.1 (b) in the appendix. Similar to Fig. 3.3, Fig. 3.5 also shows that increasing the percentage of attackers ( $P_a$ ) decreases software diversity ( $SD$ ) and the size of the giant component ( $S_g$ ) while increasing the percentage of compromised nodes ( $P_c$ ) and the defense cost ( $D_c$ ).

The overall performance order in terms of  $P_c$  (representing network security) and  $S_g$  (representing network connectivity and resilience) is: SDA with optimal  $\rho = -0.4 \geq$  SDA with  $\rho = 0 \geq$  SDA with  $\rho = 1 \approx$  Random-A  $\geq$  Random-Graph-C  $\approx$  No-A. In terms of  $SD$  (software diversity), a similar performance order is observed except that Random-Graph-C has a higher  $SD$  than No-A. These results demonstrate that SDA schemes clearly are more

effective than traditional software shuffling schemes that do not change the network topology (e.g., Random-Graph-C). For the defense cost ( $D_c$ ), the overall performance order (the lower cost the better) is: Random-Graph-C  $\geq$  SDA with optimal  $\rho = -0.4 \geq$  Random-A  $\approx$  SDA with  $\rho = 1 \geq$  SDA with  $\rho = 0 \geq$  No-A. Again these results support the claim that SDA-based schemes incur relatively low cost, while outperforming all counterpart baseline schemes in  $P_c$ ,  $S_g$ , and  $SD$ .

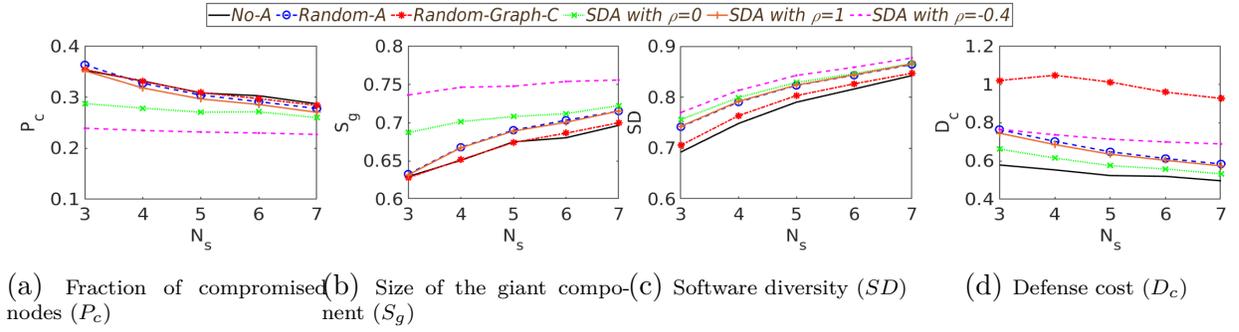


Figure 3.6: Effect of the number of software packages ( $N_s$ ) under a medium network.

### Effect of Varying the Number of Software Packages ( $N_s$ )

Fig. 3.6 shows the effect of  $N_s$  on performance under the medium network. We again observe that increasing the number of software packages available ( $N_s$ ) increases software diversity ( $SD$ ) and the size of the giant component ( $S_g$ ) while decreasing the percentage of compromised nodes ( $P_c$ ) and the defense cost ( $D_c$ ). The overall performance order is the same as that in Figs. 3.5, with SDA with optimal  $\rho = -0.4$  outperforming all other schemes in terms of  $SD$ ,  $S_g$ , and  $P_c$  and performing comparably to Random-A in terms of  $D_c$ . By comparing Fig. 3.6 (for the medium dense network) with Fig. 3.4 (for the dense network), we also observe that SDA with optimal  $\rho$  is more effective in the dense network. We attribute this to node density. That is, SDA is more effective when there are many connections between nodes in the network allowing SDA to effectively decide which edges

to add or remove to effectively maximize software diversity ( $SD$ ) and the size of the giant component ( $S_g$ ) thereby minimizing the percentage of compromised nodes ( $P_c$ ).

### 3.7.3 Comparative Performance Analysis Under a Sparse Network

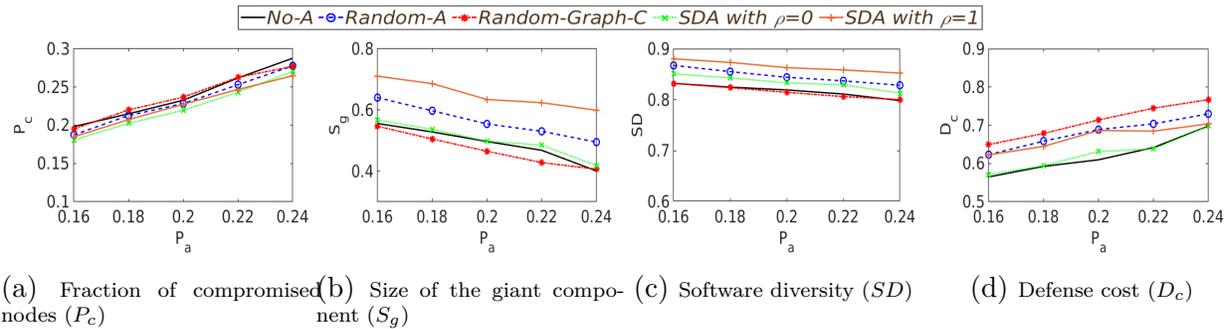


Figure 3.7: Effect of varying the fraction of attackers ( $P_a$ ) under a sparse network.

#### Effect of Varying the Fraction of Initial Attacks ( $P_a$ )

Fig. 3.7 shows the effect of varying the initial attack density ( $P_a$ ) on the performance of the five schemes with respect to the 4 performance metrics discussed in Section 3.6.1 under the sparse network, whose network topology and degree distribution are shown in Fig. A.1 (c) in the appendix. Unlike in the cases of the medium and dense networks, the SDA with optimal  $\rho$  scheme in the sparse network is the same as the SDA with  $\rho = 1$  scheme which restores all edges from the lost edges in Step 1 (i.e.,  $\rho = 1$ ). Therefore, we only show comparative experimental results of the five schemes.

In the sparse network, the degrees of most nodes are very small, implying that nodes are minimally connected where most nodes only have 1-3 neighbors at most. This means that the network itself is relatively much less vulnerable to epidemic attacks because the attackers inherently cannot reach many nodes to compromise due to network sparsity. On the other

hand, this means that when there is a higher percentage of attackers, the damage upon an attack success (i.e., failing or compromising a node) is more detrimental by resulting in a much smaller size of the giant component representing a significantly lower network resilience (or availability), which introduces a great hindrance to providing continuous services due to a lack of paths available from a source to a destination. This trend can be clearly observed with the sharp decrease in the size of the giant component ( $S_g$ ) under high attack density (i.e.,  $P_a = 0.24$ ), when compared to the corresponding results under the medium network (i.e., Fig. 3.5 (b)). A more interesting result is that the overall performance trend does not follow the previous results shown under the dense network (i.e., Fig. 3.3) and medium network (i.e., Fig. 3.5) which have a sufficiently larger number of edges than the sparse network. The performance order in  $S_g$  is: SDA with optimal  $\rho = 1 \geq \text{Random-A} \geq \text{Random-Graph-C} \approx \text{No-A} \geq \text{SDA with } \rho = 0$ . Since the original network itself is sparsely connected, SDA with  $\rho = 0$  is not as effective as shown in our previous results for  $S_g$  under the dense network (see Fig. 3.3) and medium network (see Fig. 3.5). SDA with optimal  $\rho = 1$  with all edges restored from the lost edges in Step 1 performs the best in  $S_g$ . This result is reasonable because the sparse network does not need to disconnect more edges because it is already sparse enough and significantly less vulnerable to epidemic attacks.

The overall performance with respect to  $P_c$  is very similar among all five schemes, with slightly better results in the two SDA schemes. Similarly to the result shown for the dense network, Random-Graph-C exhibits the same level of performance as No-A in  $P_c$  and  $S_g$ , but with a higher software diversity ( $SD$ ). This indicates the advantage of topology-aware adaptation in a sparse network. For the defense cost ( $D_c$ ) the performance order is: Random-Graph-C  $\geq$  Random-A  $\geq$  SDA with optimal  $\rho = 1 \geq$  SDA with  $\rho = 0 \geq$  No-A. It is interesting to observe that all SDA-based schemes incur a lower defense cost than Random-A and Random-Graph-C possibly due to fewer compromised nodes in the system and thus

less frequent IDS interventions.

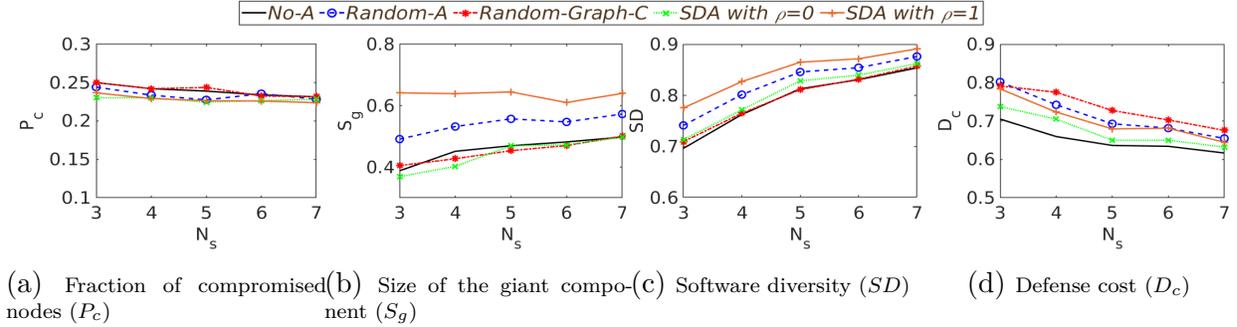


Figure 3.8: Effect of the number of software packages ( $N_s$ ) under a sparse network.

### Effect of Varying the Number of Software Packages ( $N_s$ )

Fig. 3.8 shows the effect of varying the number of software packages available ( $N_s$ ) on the performance of the five schemes under the sparse network. As expected, as  $N_s$  increases,  $SD$  (software diversity) increases and  $D_c$  (defense cost) decreases. As  $N_s$  increases,  $S_g$  (size of the giant component) also increases for all schemes except for the SDA with optimal  $\rho = 1$  scheme. The reason is that when  $\rho = 1$ , SDA will restore all edges removed in Step 1 (see Step 1 in Algorithm 1). When  $N_s$  is higher, fewer edges will be removed in Step 1 because of a smaller probability that two neighbor nodes will have the same software package. Consequently, when  $N_s$  is higher, the very same smaller number of edges will be added back in Step 2 (see Step 2 in Algorithm 1), thus resulting in the size of the giant component in the shuffled topology not necessarily larger than the one when  $N_s$  is lower.

By comparing Fig. 3.8 (for the sparse network) with Fig. 3.6 (for the medium dense network) and Fig. 3.4 (for the dense network), we notice that SDA with optimal  $\rho$  is most effective in the dense network. We conclude that our proposed SDA algorithm is most effective in a dense network under which SDA can effectively decide which edges among many to add or remove to effectively maximize software diversity ( $SD$ ) and the size of the giant component

( $S_g$ ) as well as minimizing the percentage of compromised nodes ( $P_c$ ).

## 3.8 Key Findings

From our extensive simulation experiments, we obtain the following key findings:

- Overall under epidemic attacks, more interconnectivity between nodes in a network introduces higher security vulnerability while bringing a larger size of the giant component, implying higher network connectivity. In addition, when two nodes use the same software package where the vulnerability of the software package is known to an attacker, it provides a high advantage to the attacker. How nodes are connected to each other is highly critical in determining the network's vulnerability to epidemic attacks.
- Even if two network topologies have the same network density (i.e., the same number of edges), how nodes are connected to each other can vastly change the extent of the security vulnerability to epidemic attacks. It is even possible that a sparser network may introduce more security vulnerability than a denser network depending on how the nodes are connected to each other.
- It is not necessary to consider the entire network topology for each node to make effective edge adaptation decisions to minimize security vulnerability while maximizing network connectivity. Our SDA algorithm allows each node to make effective decisions on edge adaptation in a lightweight manner. This is because edge adaptation decisions are determined based on ranking of node software vulnerability values, which is more flexible than using a threshold, to achieve the dual goals of security and performance.
- Under medium dense and dense networks, our SDA scheme significantly outperforms

existing counterpart baseline schemes. However, under the sparse network, although our SDA scheme still outperforms other schemes, the difference was less significant. We conclude that our SDA scheme is most effective in a dense network under which SDA can effectively decide which edges among many existing connections to add or remove to effectively maximize software diversity and the size of the giant component as well as minimizing the percentage of compromised nodes.

- Our proposed SDA scheme is extremely resilient to harsh environments. The performance gain relative to counterpart baseline schemes increases as the environment is harsher, i.e., as the percentage of attackers increases or as the number of the software packages decreases. This proves the high resilience of the proposed SDA scheme under a highly disadvantageous environment.

## Chapter 4

# Deep Reinforcement Learning-based Vulnerability-Aware Network Adaptations for Resilient Software-Defined Networks

This chapter addresses the **Adaptability Task**. This chapter is based on the paper “DREVAN: Deep Reinforcement Learning-based Vulnerability-Aware Network Adaptations for Resilient Networks” published in The ninth annual IEEE Conference on Communications and Network Security (IEEE CNS), Oct. 2021 [172], the paper “Network Resilience Under Epidemic Attacks: Deep Reinforcement Learning Network Topology Adaptations” published in The 2021 IEEE Global Communications Conference (GLOBECOM), Dec. 2021 [170], and the paper “EVADE: Efficient Moving Target Defense for Autonomous Network Topology Shuffling Using Deep Reinforcement Learning” accepted to The 21st International Conference on Applied Cryptography and Network Security (ACNS), Jun. 2023 [176].

## 4.1 Motivation & Research Goal

Network resilience research has been studied in order to maximize network connectivity even under the presence of node failures or compromise in the network science domain [157]. However, network scientists mainly focused on maximizing the size of the giant component to maintain network resilience, whereas security vulnerability exposed in the network due to epidemic attacks has not been sufficiently addressed. Further, in the network security domain, researchers have proposed the concept of diversity as a security design to ensure software polyculture with the aim of preventing attackers from taking huge advantages of using the same software [174]. When nodes use the same software, this allows attackers to exploit the known vulnerability of a single software in a network with the so-called software monoculture [167]. Although software shuffling based on graph coloring algorithms has been used to solve a software assignment problem [167], their focus is mainly on minimizing security vulnerability. However, software shuffling has been proven less effective in reducing security vulnerability in [171], which showed the clear outperformance of edge shuffling in maximizing system security and network connectivity over software shuffling, which does not change network topology. Further, although a vulnerability-aware diversity metric has been proposed to determine which edges to shuffle [171], identifying an optimal budget of edge adaptations to generate a resilient network topology has not been studied by minimizing security vulnerability and maximizing network connectivity in the presence of epidemic attacks. We seek to address this issue under the concept of moving target defense (MTD) that dynamically changes the attack surface (e.g., network topology) to increase uncertainty for the attacker [32]. Prior research [68, 174] has studied a network shuffling-based MTD; however, frequent changes to a network topology can be a detriment to the seamless service provisions in a system [32]. It is an NP-hard problem to identify an optimal network topology that is robust against attacks while maintaining sufficient connectivity under high

dynamics.

In this chapter, we mainly aim to achieve the following three tasks:

- We develop a vulnerability-aware network adaptation framework that can generate a robust network topology against epidemic attacks by leveraging deep reinforcement learning (DRL). We call our proposed framework DREVAN, representing Deep REinforcement Learning-based Vulnerability-Aware Network Adaptations. In this work, we mainly use the two metrics to ensure system performance and security, which are the size of the giant component for maintaining acceptable network connectivity and the fraction of compromised nodes in a given network.
- Taking one step further to ensure service availability of the network, we also used the actual delivery of correct messages in the network with insider attackers or under network congestion, in addition to the size of the giant component. The proposed scheme is called DeepNETAR representing *a Deep reinforcement learning-based NETwork Adaptations for network Resilience algorithm*. DeepNETAR will identify an optimal pair of network adaptations (i.e., numbers of edge additions and removals) that can meet conflicting system objectives of maximizing service availability and minimizing network vulnerability.
- To extend the above schemes under the Moving Target Defense (MTD) framework, we proposed a scheme called EVADE, representing Efficient moVing tArget DEfense. EVADE could adapt the network topology periodically to ensure long-term system security and performance under epidemic attacks.

## 4.2 Problem Statement

In this work, we aim to identify a network topology robust against epidemic attacks by adding or removing edges between nodes to minimize security vulnerabilities introduced by compromised nodes (see our attack model in Section 4.4.3) while maximizing network connectivity even under epidemic attacks within the limits of budget  $B$  to adjust edges. We leverage the state-of-the-art algorithms of DRL, particularly a series of Deep Q-learning network (DQN) (i.e., regular DQN, Double DQN, and Dueling DQN), and let a DRL agent identify a pair of budgets for adding edges,  $b_A$ , and removing edges,  $b_R$ , given the constrained budget  $0 \leq b_A + b_R \leq B$ .

Given an original network topology  $G$ , the DRL agent aims to generate the pair of edge adaptations (i.e.,  $(b_A, b_R)$ ) and generate an adapted network  $G'$  by identifying  $(b_A, b_R)$ , respectively. Since it is an NP-hard problem to identify an optimal robust network topology against epidemic attacks by considering all possible edges to remove or add, we use two heuristic algorithms to substantially reduce the solution space while maintaining high solution optimality: (1) *Vulnerability Ranking of Edges and Nodes* (VREN) is designed to generate vulnerability scores of existing edges and nodes based on how often an edge can be used by attackers or how often a node appears on attack paths. When the DRL agent identifies an optimal pair of budgets  $(b_A, b_R)$ , the system simply removes the  $b_R$  most vulnerable edges from  $G$  and adds the  $b_A$  least vulnerable edges to the original network  $G$ , resulting in an adapted network,  $G'$ . We provided the detail of the VREN in Section 4.5.1; and (2) *Fractal-based Solution Search* (FSS) is presented to significantly reduce the complexity of identifying an optimal pair of the two edge adaptation budgets,  $(b_A, b_R)$ , as the complexity increases depending on how many pairs of  $(b_A, b_R)$  the DRL agent tries to identify the optimal solution. FSS is designed to significantly reduce this complexity and introduce a fast

convergence of the solution identified by the DRL agent, as detailed in Section 4.5.2.

To state this formally, the DRL agent aims to achieve the following objective via the proposed VREN and FSS:

$$\arg \max_{b_A, b_R} f(G') - f(G), \quad s.t. \quad 0 \leq b_A + b_R \leq B, \quad (4.1)$$

where  $B$  is an upper bound of the total budgets that can be used to make edge adaptations,  $f(G)$  returns the values depending on the evaluation function,  $f(\cdot)$ , that can be defined according to different system objectives as below:

- **O-SG:**  $f : G \mapsto \mathcal{S}_G(G) - \mathcal{F}_C(G)$  for the schemes that aim to maximize the size of the giant component and minimize the fraction of compromised nodes. We will append ‘SG’ at the end of a scheme that considers this evaluation function.
- **O-MD:**  $f : G \mapsto \mathcal{P}_{MD}(G) - \mathcal{F}_C(G)$  for the schemes that aim to maximize service availability in terms of the delivery ratio of correct messages and minimize the fraction of compromised nodes. We will append ‘MD’ at the end of a scheme with this evaluation function.
- **O-SG-MD:**  $f : G \mapsto \mathcal{S}_G(G) + \mathcal{P}_{MD}(G) - \mathcal{F}_C(G)$  for the schemes to maximize both the size of the giant component and the service availability and minimize the fraction of compromised nodes. We will append ‘SG-MD’ at the end of a scheme using this evaluation function.

We describe the metrics associated with these three objectives in Section 4.6.3. Note that  $f(\cdot)$  will be generated based on the two algorithms (i.e., VREN and FSS) that can contribute to efficiently identifying an optimal  $(b_A, b_R)$  by the DRL agent, as described in Section 4.5. Note that it is a non-trivial task for the DRL agent to identify an optimal  $(b_A, b_R)$  that can

meet multiple objectives, which is well known as the nature of multi-objective optimization (MOO) problems [30]. This implies that the optimal solutions identified by the DRL do not necessarily achieve the best performance in all objectives. We elaborate this discussion with more details in Section 4.7.1 and 4.7.2.

In the CNS paper, we only consider  $f(\cdot)$  to be **O-SG**. In the Globecom work, we consider all scenarios listed above: **O-SG**, **O-MD**, and **O-SG-MD**.

### 4.3 Key Contributions

We make the following **key contributions** in this work:

- Although DRL has been used to generate an optimal robust network topology against adversarial attacks, this is the first that considers both edge additions and removals to maximize network resilience by achieving the conflicting goals of maximizing system security and network connectivity.
- We present two algorithms to support the DRL agent to efficiently identify an optimal adaptation budget strategy to meet the two system goals. They include: (1) the Vulnerability Ranking algorithm of Edges and Nodes, namely VREN, and using the vulnerability levels to select candidates for edge adaptations; and (2) the Fractal-based Solution Search algorithm, namely FSS, to substantially reduce the steps to identify an optimal adaptation budget of edge additions and removals.
- Via simulation experiments, we conduct extensive comparative performance analysis based on six network topology adaptation schemes, including DRL algorithms, three Deep Q-learning algorithms (i.e., DQN, Double DQN, and Dueling DQN) and three existing counterparts (i.e. Greedy, Random, and Optimal). We find that DRL-based

network topology adaptations particularly outperform with regard to minimizing system security vulnerability.

- We also conduct additional experiments to ensure the effectiveness and efficiency of the DeepNETAR which can meet different types of security and performance objectives using diverse metrics. We found that a larger size of the giant component is not necessarily aligned with higher service availability (i.e., correct message delivery ratio). In addition, allowing a higher fraction of compromised nodes can increase actual service availability due to the existence of more paths available between two nodes.
- We introduce a hybrid approach to speed up the MTD process. We develop a greedy algorithm, called Density Optimization (DO), to narrow the search space for the DRL agent. The agent leverages the initial estimate from DO and quickly converges to the optimal solution(s).
- We consider *epidemic attacks* and *state manipulation attacks*. The epidemic attacks are modeled as dynamically arriving attackers that can compromise adjacent nodes. The state manipulation attacks are modeled as inside attackers that can perturb system states informing DRL. The performance of EVADE is evaluated as a network topology shuffling-based MTD triggered multiple rounds over time based on real datasets. We prove our approach shows higher resilience against those attacks than existing counterparts.
- Via extensive experiments, we analyze the performance of EVADE incorporating one of two DRL methods, Deep Q-Network (DQN) [111] and Proximal Policy Optimization (PPO) [138]. We prove that DRL-based MTD schemes (i.e., DQN/PPO with EVADE or hybrid EVADE combining DRL and DO) outperform over existing counterparts in the two objectives.

## 4.4 System Model

Our system model includes the network model, node model, and attack model along with the assumptions made in this work. Note that all three works [170, 172, 176] share the same system model unless we state otherwise.

### 4.4.1 Network Model

We consider an IP network with a centralized coordinator to run our proposed network topology shuffling framework. This type of network is very common, such as a software-defined network (SDN) with a single SDN controller [89], an Internet-of-Things (IoT) network that has an edge server or central cloud server, or a special type of an ad-hoc mobile network with a centralized controller [82]. In the present work, we consider a single DRL agent that is capable of making autonomous decisions. The DRL agent will identify a network topology that is robust against epidemic attacks and ensures network connectivity for providing seamless network services. For a large-scale network, we can divide it into multiple network partitions and deploy one DRL agent for each partition. Each agent could update and share its partition with other agents. Therefore, our work can be extensible to a large-scale network with multiple DRL agents for cooperative, autonomous decision-making.

To realize this, we adopt a hybrid SDN structure as the control plane of our IP network. As in Fig. 4.1, the single SDN controller can gather information from OpenFlow switches and form a global view on the current network state, i.e., the network topology and software packages used by nodes in the network. This information will allow the DRL agent placed on the server to make autonomous decisions in adding or removing edges. The adapted network topology based on such decisions can mitigate security vulnerability and maintain network connectivity. Once the topology adaptation decision is made, the centralized controller

notifies the hosts to execute the action through their IP addresses.

We represent a network as an undirected graph,  $G = (V, E)$ , where  $V$  refers to a set of nodes and  $E$  is a set of edges between two nodes. The network is dynamic and changes when nodes fail due to defects or attacks or when a network topology is changed by the network shuffling process to meet the two system objectives. We denote a network topology by an adjacency matrix  $\mathbf{A}$ , which has an entry 1 (i.e.,  $a_{ij} = 1$ ) for an edge existing between two nodes  $i$  and  $j$  and 0 (i.e.,  $a_{ij} = 0$ ) otherwise. In EVADE, We adopt a time-based MTD performing a network shuffling operation per time cycle in minutes. The DRL agent keeps making network topology shuffling decisions to prevent potential epidemic attacks while maintaining acceptable network connectivity (e.g., maximum possible connectivity in terms of the size of the giant component described in Section 4.6.3). We describe how the proposed schemes works in Section 4.5.

As a background defense mechanism, we assume that a network-based intrusion detection system (NIDS) exists on the SDN controller to monitor all inter-host traffic and detect outside attackers and inside attackers, nodes compromised by the attackers. The NIDS will capture and analyze network traffic from switches in a given SDN. We consider the detection accuracy of the NIDS with probability  $\zeta$ , which has the similar effect of removing nodes as in the Susceptible-Infected-Removed (SIR) epidemic model [118]. As the response to the detected compromised nodes by the NIDS, a secret key used in the network will be changed and this allows their isolation from the network for secure group communications. We characterize the NIDS by false positives and negatives with the probabilities  $P_{fp}$  and  $P_{fn}$  that set to  $1 - \zeta$ . Since the DRL agent relies on the information from the NIDS for the status of nodes to measure system security (see the metrics defined in Section 4.6.3), it may not be able to choose the best solution under the ground truth knowledge, which reflects a realistic scenario in practice. Note that our work does not develop a NIDS, which is beyond the scope

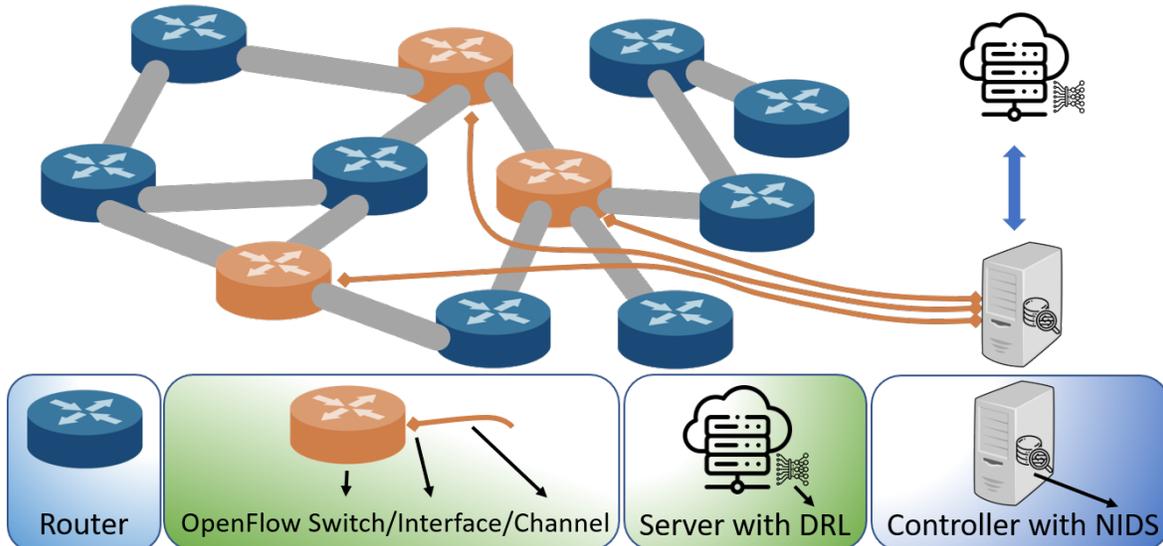


Figure 4.1: An overview of the network model.

of our work. We focus on developing an efficient network topology shuffling framework with lightweight features for faster learning convergence to find an optimized pair of network adaptation budgets. We display our network model in Fig. 4.1.

#### 4.4.2 Node Model

Each node  $i$  has the following attributes: (1) Whether node  $i$  is active or not (i.e.,  $na_i = 0$  or  $na_i = 1$ ); (2) Whether node  $i$  is compromised or not (i.e.,  $nc_i = 0$  or  $nc_i = 1$ ); (3) What software node  $i$  has, denoted by  $s_i \in \{1, 2, \dots, N_s\}$ , where  $N_s$  indicates how many software packages are available for nodes in the network; and (4) What level of vulnerability node  $i$  has, denoted by  $sv_i$ . To properly model  $sv_i$ , we employ the Common Vulnerability Scoring System (CVSS) [1]. In the CVSS, a node's average vulnerabilities for different types of attacks are given in the range of  $[0, 10]$ . We normalize it to  $[0, 1]$  and consider  $sv_i$  as  $\frac{sv_i}{N_s+1}$ . Our case study will select these vulnerability values randomly based on uniform distribution as our work does not focus on particular software vulnerabilities, which is beyond the scope

of this work.

### 4.4.3 Attack Model

This work considers the following attack behaviors to investigate the performance of the proposed DREVAN [172], DeepNETAR [170] and EVADE [176]:

- *Epidemic Attacks* [170, 172, 176]: In [170, 172], we randomly select initial epidemic attackers following the attack probability  $P_a$ . In [176], we randomly select initial epidemic attackers following a Poisson distribution with  $\lambda$ , which is the expected number of new attackers arriving in one attack round. Each initial attacker is assumed to compromise one host from its public IP address. The infection rate,  $\beta_{ji}$ , is used to model the compromising behavior of epidemic attacks from attacker  $j$  to node  $i$  [64]:

$$\beta_{ji} = \begin{cases} 1 & \text{if } \sigma_j(s_i) > 0; \\ sv_i & \text{otherwise,} \end{cases} \quad (4.2)$$

where  $\sigma_j \in \mathbb{B}^l$  defines a boolean vector on  $l$  software packages and indicates whether each software package's vulnerability is learned by the attacker  $j$ . The  $\beta_{ji}$  returns  $sv_i$  when attacker  $j$  has not compromised nodes with the same  $s_i$  (i.e.,  $\sigma_j(s_i) = 0$ ) and returns 1 otherwise. We consider the epidemic attack behavior based on the Susceptible-Infected-Removed (SIR) model [118] where the removed status refers to the compromised nodes being detected and evicted from the system. The attacker can directly spread viruses or malwares and compromise its adjacent nodes without access rights to their settings/files, such as botnets [110].

- *State Manipulation Attacks* [172, 176]: Compromised but undetected inside attackers can disrupt the process of the DRL agent’s learning and decision process. The DRL agent can learn and make decisions based on the feedback received for a previous action taken, which is considered as a reward to the environment, and aims to reach a desirable system state (i.e., minimum security vulnerability and maximum network connectivity). Hence, it is critical for the DRL agent to keep track of correct system states. In order to validate the resilience of the proposed schemes, we considered a state manipulation attack (SMA) with  $P_s$ , which is the probability for a system state to be manipulated by the attacker. To counter this type of attack, the proposed schemes also uses a detector to detect the SMA with the detection probability,  $D_r$ . We evaluate the impact of the SMA by varying  $P_s$ , given  $D_r$ . Note that developing a detector for the SMA is beyond the scope of our work. It will be considered in our future work. If the SMA is correctly detected, the DRL agent will simply use a previous state instead of using a compromised state by the SMA. Otherwise, the DRL agent will use the compromised state to identify an optimal strategy of the edge adaptation budget pair.
- *Packet drop attack* [170]: If an attacker (i.e., compromised and undetected by the NIDS) receives a packet, it will drop a packet randomly with probability  $P_d = 0.5$ .
- *Packet modification attack* [170]: If the attacker decides to forward the packet, it will modify the packet randomly with probability  $P_m = 0.5$ .

## 4.5 Proposed Approach: Deep Reinforcement Learning-based Network Adaptations

In this section, we describe the detail of our proposed deep reinforcement learning-based network adaptation schemes, i.e., DREVAN, DeepNETAR, and EVADE.

### 4.5.1 Vulnerability Ranking of Edges and Nodes (VREN)

As shown in Algorithm 2, the proposed *VREN* can perform network topology adaptations with any given budget pair  $(b_A, b_R)$ , where  $b_A$  and  $b_R$  refer to edge addition and edge removal budgets, respectively. We define the node (or vertex) vulnerability levels  $V_V$  and edge vulnerability levels  $V_E$  in an underlying graph  $G = (V, E)$  of a network. We define the vulnerability level of each edge by the number of times used by attackers to compromise other nodes. Similarly, we define the vulnerability level of each node by the number of times it can be compromised by attackers as it appears on attack paths. Then, we calculate  $V_V$  and  $V_E$  by running  $n_a$  attack simulation runs on  $G$ . To this end, `EpidemicAttacks( $G$ )` is implemented and returns the vulnerability values of nodes and edges in graph  $G$  based on the frequency of them appearing on attack paths. Lastly, we rank edges and nodes respectively based on their vulnerability levels. We denote the rank of edges and nodes as  $R_E$  and  $R_V$ , respectively, where  $R_E$  is sorted in descending order and  $R_V$  is sorted in ascending order. The  $R_E$  and  $R_V$  converge as  $n_a$  increases. Hence, choosing an appropriate  $n_a$  is critical. The

---

**Algorithm 2 Vulnerability Ranking of Edges and Nodes (VREN)**

---

```

1:  $G \leftarrow$  An original network
2:  $N \leftarrow$  Total number of nodes in a network
3:  $b_A \leftarrow$  Addition budget
4:  $b_R \leftarrow$  Removal budget
5:  $n_a \leftarrow$  Total number of attack simulations to identify edge and node vulnerabilities based on their appearance on attack paths
6:  $k \leftarrow$  Upper hop bound for edge addition
7:  $G' = \mathbf{VREN}(G, b_A, b_R)$ 
8:  $V_V \leftarrow$  Node vulnerability levels, initialized as a zero vector
9:  $V_E \leftarrow$  Edge vulnerability levels, initialized as a zero vector
10: 1.A: Edges Removals
11: for  $t \leftarrow 1$  to  $n_a$  do
12:    $V_V^*, V_E^* = \text{EpidemicAttacks}(G)$  ▷ Single attack simulation
13:    $V_E = V_E + V_E^*$ 
14: end for
15:  $R_E \leftarrow$  rank  $E$  in descending order with key  $V_E$ 
16:  $G^* \leftarrow$  remove top  $b_R$  edges from  $G$  according to  $R_E$ 
17: for  $t \leftarrow 1$  to  $n_a$  do
18:    $V_V', V_E' = \text{EpidemicAttacks}(G^*)$  ▷ Single attack simulation
19:    $V_V = V_V + V_V'$ 
20: end for
21: 1.B: Edges Additions
22:  $R_V \leftarrow$  rank  $V$  in ascending order with key  $V_V$ 
23:  $gc \leftarrow$  the giant component of  $G^*$ 
24:  $R_V^{gc}, R_V^{ngc} \leftarrow$  order  $gc$  and the complement  $ngc$  based on  $R_V$ 
25:  $G' = G^*$  ▷ All the following adaptations are on  $G'$ 
26: for  $i$  in  $R_V^{ngc}$  do
27:   for  $j$  in  $R_V^{gc}$  do
28:      $G' \leftarrow$  Connect nodes  $i, j$  and break if  $dist(i, j) \in [2, k]$  ▷  $dist$  considers the length of a shortest path between nodes  $i$  and  $j$  and  $k$  is the maximum number of hops allowed
29:   end for
30: end for
31:  $b_A^{temp} \leftarrow \|*\| R_V^{ngc}$ 
32: for  $h \leftarrow 2$  to  $N$  do
33:    $R_V^h \leftarrow$  top  $h$  nodes in  $R_V$ 
34:   for  $i$  in  $R_V^h$  do
35:      $R_V^{i-1} \leftarrow$  top  $i - 1$  nodes in  $R_V$ 
36:     for  $j$  in  $R_V^{i-1}$  do
37:       break if  $b_A^{temp} \geq b_A$ 
38:       if  $dist(i, j) \leq k$  then
39:          $G' \leftarrow$  Connect nodes  $i, j$  if  $dist(i, j) \in [2, k]$ 
40:          $b_A^{temp} = b_A^{temp} + 1$ 
41:       end if
42:     end for
43:   end for
44: end for
45: return  $G'$ 

```

---

expected fraction of compromised nodes ( $\mathcal{F}_C$ ) in the network can be estimated by:

$$\begin{aligned}
\mathbb{E}(\mathcal{F}_C) &= \sum_{i=1}^N P_i = \sum_{I \subseteq V} P_I \sum_{i=1}^N P_i^I = \sum_{i=1}^N \sum_{I \subseteq V} P_I P_i^I \\
&= \sum_{i=1}^N \lim_{n_a \rightarrow \infty} (V_V)_i + \|\ast\| I = \|\ast\| I + \lim_{n_a \rightarrow \infty} \sum_{i=1}^N V_V^i \\
&= \|\ast\| I + \lim_{n_a \rightarrow \infty} \sum_{1 \leq i < j \leq N} V_E^{ij}, \tag{4.3}
\end{aligned}$$

where  $N$  is the total number of nodes in a network,  $I$  is the set of initial attackers,  $\|\ast\| \cdot$  returns the set cardinality,  $P_i$  is the probability of node  $i$  being compromised,  $P_I$  is the probability of  $I$  as the set of initial attackers,  $P_i^I$  is the conditional probability of  $i$  being compromised if  $I$  is the set of initial attackers,  $V_V^i$  is the vulnerability of node  $i$  and  $V_E^{ij}$  is the vulnerability of an edge between nodes  $i$  and  $j$ .

Given  $G = (V, E)$  and  $(b_A, b_R)$ , our proposed network topology adaptation produces a new topology  $G' = \text{VREN}(G, b_A, b_R)$  following the procedures below:

1. The edge vulnerabilities,  $\mathbf{V}_E$ , are estimated based on  $G$ , and the top  $b_R$  edges are removed based on the edge vulnerability ranks in  $\mathbf{R}_E$ . The resulting graph,  $G^*$ , represents an intermediate adapted graph.
2. A minimum number of edges are added to restore the connectivity between a pair of nodes with the maximum distance  $d_a$  in  $G^*$  where the resulting adapted graph is  $G'$ .
3.  $\mathbf{R}_V$  of  $G'$  is estimated and maximum  $b_A$  edges are added to  $G'$  based on the rank in  $\mathbf{R}_V$ . Since a pair of nodes can be connected also based on the distance between them, they can be connected only when the distance between the pair is no greater than distance  $d_a$ .

To introduce efficiency while maintaining accuracy of vulnerability estimation, we can use ego networks to estimate  $\mathbf{R}_E$  and  $\mathbf{R}_V$ . We can use hop distance  $h$  to describe the ego network. Specifically, an  $h$ -hop ego network for node  $k$ , denoted by  $EN_k^h$ , is defined to be the network including all nodes within distance  $h$  from  $k$ . Due to the nature of epidemic attacks, nodes are more likely to be compromised when they are closer to the attackers. We consider each node's ego network to simulate attacks and calculate  $\mathbf{R}_E$  and  $\mathbf{R}_V$ . We choose a hop distance  $h^*$  and simulate attacks in  $N$  ego networks,  $EN_k^{h^*}$ , for node  $1 \leq k \leq N$ . Given a certain number of attack simulation runs, using the ego network allows us to compute the number of times each node  $i$  ( $\neq k$ ) becomes compromised and generate the following list:

$$[NC_i^0, NC_i^1, \dots, NC_i^j, \dots, NC_i^{h^*}], \quad (4.4)$$

where  $NC_i^j$  is the number of times node  $i$  becomes compromised in  $EN_k^{h^*}$  for all node  $k$  such that node  $i$  is within  $j$  hop(s) distance away from node  $k$ .  $h^*$  determines the size of node  $k$ 's ego network and  $j$  decides the distance between nodes  $i$  and  $k$ . Hence,  $NC_i^0$  refers to the case of  $i = k$  and indicates the number of times node  $i$  being compromised in its ego network,  $EN_i^{h^*}$ .

After then, we can rank nodes and edges based on  $NC_i^j$  where  $h^*$  refers to the hop value used to determine every node's ego network. When  $h^*$  is no smaller than the maximum hop-distance between two nodes in the original network, all  $h^* + 1$  rankings will become almost the same as all nodes' ego networks would be the same. Specifically, given  $h^*$  and a certain number of attack simulations,  $n_a$ , larger  $NC_i^j$  is obtained with larger  $j$ , and vice-versa. In general, given different nodes  $i_1$  and  $i_2$ , larger  $j$  can result in more distinct differences between  $NC_{i_1}^j$  and  $NC_{i_2}^j$ . Suppose we rank nodes  $i_1$  and  $i_2$  when  $|NC_{i_1}^j - NC_{i_2}^j| > C > 0$ , where constant  $C$  is a given ranking resolution. Larger  $j$  can allow ranking with larger  $C$ , and vice-versa. That is, using larger  $j$  allows more easily identifying  $C$  as  $NC_i^j$  becomes

larger and more differences with other  $NC_i^j$ 's for other  $i$ 's. On the other hand,  $NC_i^j$  with smaller  $j$  can represent higher vulnerability than  $NC_i^j$  with larger  $j$ . In this work, we use  $h^* = 2$  and rank each node  $i$  based on  $NC_i^2$  for our experiments in Section 4.7.3.

We consider Random Adaptation (RA) as a baseline alternative adaptation if an adaptation algorithm does not use VREN. Given fixed budget pair  $(b_A, b_R)$ , RA would first remove  $b_R$  edges at random and then add  $b_A$  edges at random.

## 4.5.2 Fractal-based Solution Search (FSS)

In this section, we discuss the *Fractal-based Solution Search* algorithm, namely *FSS*. Recall that in Eq. (4.1), the objective function is defined based on the differences of the rewards from the original network  $G$  and the adapted network  $G'$ . We rewrite the objective function Eq. (4.1) by:

$$\begin{aligned} \arg \max_{b_A, b_R} f(G') - f(G) \quad \text{where } G' = \text{VREN}(G, b_A, b_R) \\ \text{s.t. } 0 \leq b_A + b_R \leq B, \end{aligned} \quad (4.5)$$

where  $f : G \mapsto \mathcal{S}_G(G) - \mathcal{F}_C(G)$ ,  $b_A$  is the addition budget,  $b_R$  is the removal budget, VREN follows Algorithm 2, and  $B$  is the upper bound of the total adaptation budget.

Since Eq. (4.5) above is an optimization problem defined within the triangle area  $\{(b_A, b_R) \mid 0 \leq b_A + b_R \leq B\}$ , we consider the fractal structure of this area in order to develop an efficient search space for the DRL agent to quickly identify an optimal solution. To be specific, we divide the triangle as described in Fig. 4.2. Since it is a self-similar fractal, we can obtain four self-similar parts for each subdivision, namely  $A, B, C$ , and  $D$ . This allows us to evaluate these divisions by using the centroid to represent each division and the nearest

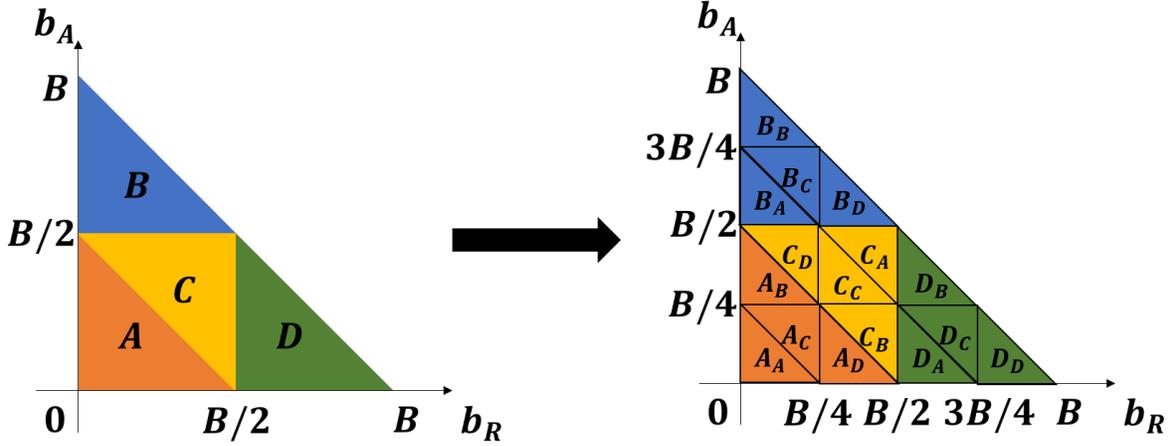


Figure 4.2: Generation of self-similar fractals to reduce solution search space in edge addition and removal budgets,  $(b_A, b_R)$ .

integer pair  $(b_A, b_R)$  from the centroid for evaluating Eq. (4.5). For each subdivision, division  $C$  always has the invariant centroid. Considering  $b_A$  and  $b_R$  as integers, it is sufficient to iterate the subdivision until the standard triangle (with leg length less than 1) is obtained.

To clearly show the enhanced efficiency in DRL by this FSS, we conducted comparative performance analysis of DRL schemes with FSS and conventional linear search (LS) as a baseline model in Section 4.7.1. LS starts searching an optimal budget from the origin in the triangular area and keeps increasing one budget unit per step until finding an optimal budget  $(b_A, b_R)$ .

### 4.5.3 DRL-based Budget Adaptation

Here we describe how the DRL agent makes decisions to add or remove edges when it is given the total budget of edge adaptations following the learning stage of the DRL agent. For this work to be self-contained, we describe the basic steps of DRL agents to learn and make decisions as follows.

For the DRL agent to learn a policy following the principle of RL, we use the mathematical

framework of RL following a Markov Decision Process (MDP), consisting of (1) a set of states  $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_t, \dots, \mathbf{s}_T\}$  with a distribution of starting states  $p(\mathbf{s}_1)$  where  $\mathbf{s}_t$  refers to a set of states  $s_t$  at time  $t$ ; (2) a set of actions,  $\mathcal{A} = \{\mathbf{a}_1, \dots, \mathbf{a}_t, \dots, \mathbf{a}_T\}$  where  $\mathbf{a}_t$  is a set of actions  $a_t$  available at time  $t$ ; (3) a transition probability,  $\mathcal{T}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ , from state  $\mathbf{s}_t$  to state  $\mathbf{s}_{t+1}$  via  $\mathbf{a}_t$ ; (4) an immediate reward function,  $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ ; (5) a discount factor  $\gamma \in [0, 1]$  where lower  $\gamma$  counts immediate rewards more; and (6) a policy  $\pi$  mapping from states to a probability distribution of actions,  $\pi : \mathcal{S} \rightarrow (\mathcal{A} = a \mid \mathcal{S})$ . Given the MDP has an episode with length  $T$ , the sequence of states, actions, and rewards in an episode are the components of a trajectory of the policy. The accumulated reward based on the trajectory of the policy results in  $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$  [146]. Therefore, RL aims to identify an optimal policy  $\pi^*$  that can generate the maximum expected reward from all states [7]:  $\pi^* = \arg \max_{\pi} \mathbb{E}[R \mid \pi]$ .

Now we describe how the key components of the MDP, including states, actions, and rewards, are formulated to identify an optimal setting of budget constraints. We describe these under our proposed FSS and the baseline LS as follows:

- **States:** There will be in total  $\lceil \log_2 B \rceil$  steps if we use FSS and do the subdivision for each step, where  $B$  is the total budget. In each step  $t$  of RL, state  $s_t$  is represented by:

$$s_t = (b_A^t, b_R^t, G_t'), \quad (4.6)$$

where  $b_A^t$  and  $b_R^t$  refer to the addition budget and removal budget identified through the division evaluation at time  $t$ , respectively, and  $G_t'$  is an adapted network at time  $t$ . Under LS,  $B$  steps will be taken to guarantee the availability of optimal trajectories in each episode.

- **Actions:** Under FSS, the action  $\mathbf{a}_t$  at step  $t$  is determined by the division ID for a

next subdivision, and given by:

$$\mathbf{a}_t = \{A, B, C, D\}, \quad 1 \leq t \leq \lceil \log_2 B \rceil.$$

On the other hand, under LS,  $\mathbf{a}_t$  is given by:

$$\mathbf{a}_t = \{\text{stop}, \text{add}, \text{remove}\}, \quad 1 \leq t \leq B,$$

where, given the total budget for edge adaptations  $B$ , **stop** refers to terminating the episode at a current step  $t$ , and **add** and **remove** increment  $b_A$  and  $b_R$  by 1, respectively.

- **Rewards:** The same reward function is used for both FSS and LS. We define the reward function at step  $t$ , denoted by  $\mathcal{R}(s_t, a_t, s_{t+1})$ , based on the difference between the evaluation functions of two consecutive states by:

$$\mathcal{R}(s_t, a_t, s_{t+1}) = f(G'_{t+1}) - f(G'_t), \quad (4.7)$$

where  $f(\cdot)$  can be defined based on different objective functions in each scheme, as discussed in Section 4.2.

In Fig. 4.3, we summarized the overall process of the proposed DREVAN and DeepNETAR described in this section.

Fig. 4.4 summarizes the overall process of DRL-based MTD considered in our EVADE. In each iteration of RL, the DRL agent needs to go through the whole EVADE process. Further, the DRL agent recalculates the vulnerability rankings by VREN and uses FSS to determine a next action.

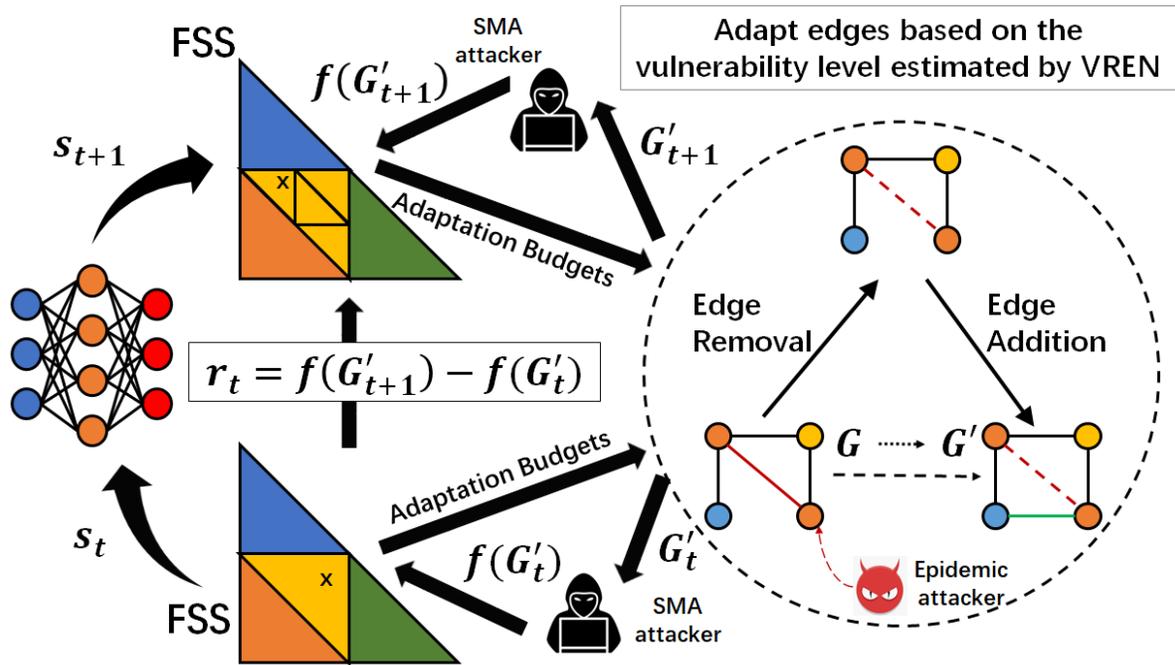


Figure 4.3: The overall architecture of the proposed DREVAN and DeepNETAR: The color of each node refers to a different software package installed in it.

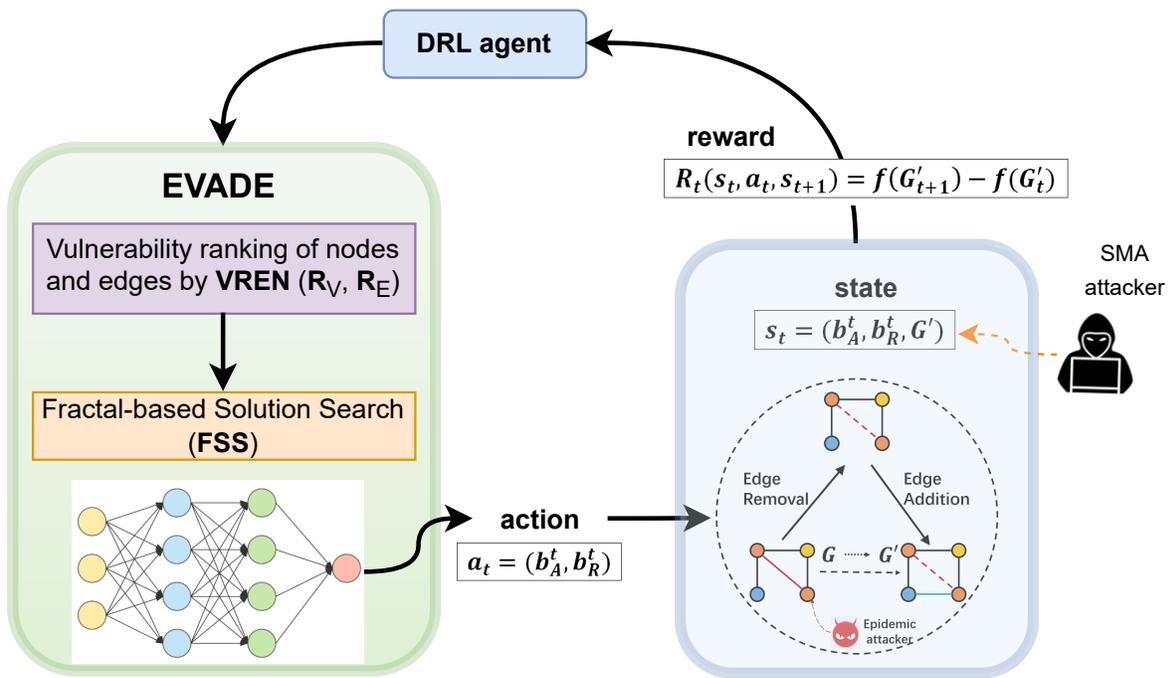


Figure 4.4: DRL-based optimal budget identification in EVADE.

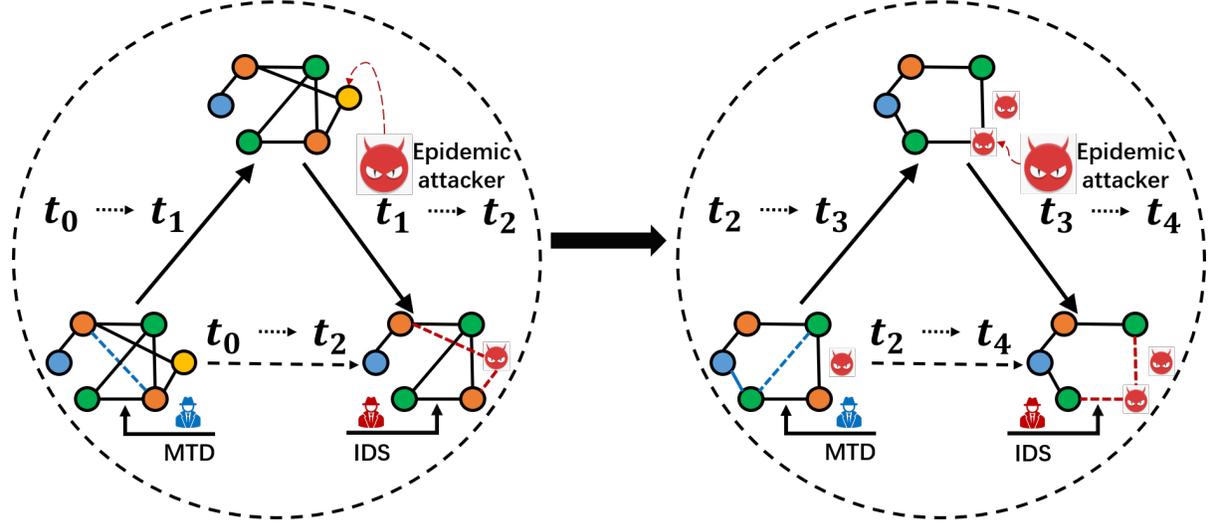


Figure 4.5: The dynamics of attacks and defenses with respect to time in terms of attack arrivals, their detection by the NIDS, and network shuffling-based MTD, as described in Section 4.5.5.

#### 4.5.4 Greedy MTD Using Density Optimization (DO)

Although we introduce a lightweight solution search algorithm, FSS, we found that DRL still requires substantially long training times. Since DRL is mainly useful when the network environment is significantly varied, we introduce a hybrid MTD approach that can minimally trigger DRL-based MTD while a more lightweight greedy algorithm is developed to trigger MTD more frequently. We discuss how this time-based MTD is triggered in Section 4.5.5. DO aims to optimize the network density with respect to the given function  $f(\cdot)$  by only adding or removing edges from the current network. First, we enumerate all possible adapted network densities, given a budget constraint  $B$  of edge additions and removals. Then we calculate the minimum budget needed to achieve any possible network density under the constraint  $B$ . Lastly, we sample the enumerated budget pairs and check them. After then, we choose the best budget  $b_{max}$  with respect to a given evaluation function  $f(\cdot)$  defined in Eq. (4.1).

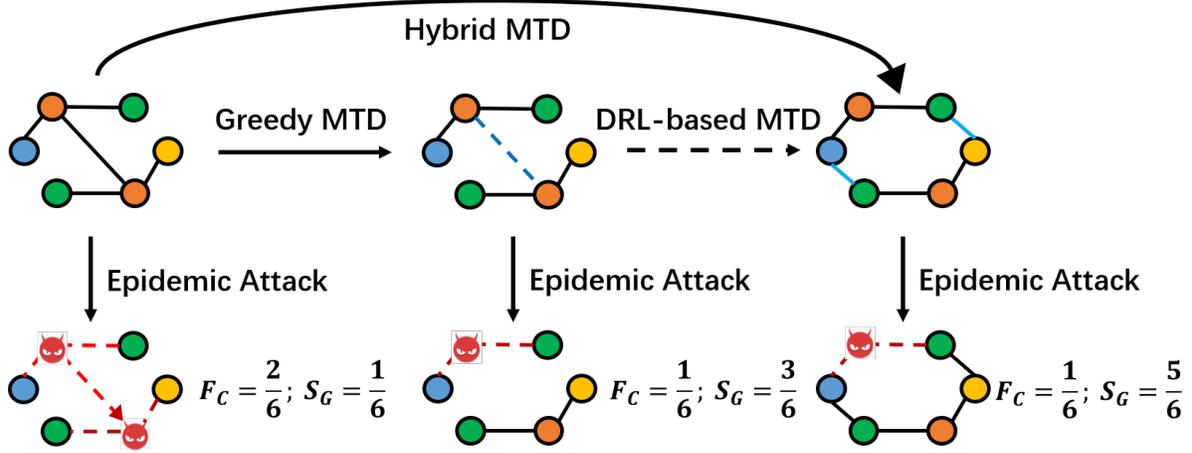


Figure 4.6: The overview of hybrid MTD, EVADE, with the greedy and DRL-based MTD.

#### 4.5.5 Hybrid MTD with EVADE and DO

We consider a time-based MTD [32] which triggers an MTD operation at fixed intervals. As shown in Fig. 4.5, we first use MTD to take proactive defense by adapting a network topology. Then epidemic attacks are applied on the adapted network. In the attack phase, new attackers will arrive in and the NIDS will detect and isolate them with probability  $\zeta$  (see Section 5.4). These events, including MTD, attack, and detection by the NIDS, occur within each round in the simulation. We will repeat this procedure multiple times until a given session ends. We consider a hybrid MTD (see Fig. 4.6) that uses both our proposed greedy MTD algorithm in Section 4.5.4 and our proposed DRL-based MTD in Section 4.5.3. We initiate the greedy MTD and DRL-based MTD periodically with time intervals  $ADC_g$  and  $ADC_{DRL}$ , respectively. In the hybrid MTD, we can refine the solution search area by FSS with DO in Section 4.5.4. To realize this, we start with the point  $b_{max}$  identified by DO and obtain the corresponding expanded triangle as described in Fig. 4.7. Here  $\rho_1$  and  $\rho_2$  are the customized lengths related to the area of the expanded triangle. Notice that the angle is 45 degrees so that the DRL agents can search for budgets corresponding to similar network densities in the refined triangle area. To develop an efficient and effective online MTD, we propose the following ways to reduce the offline evaluation times for the DRL-based MTD:

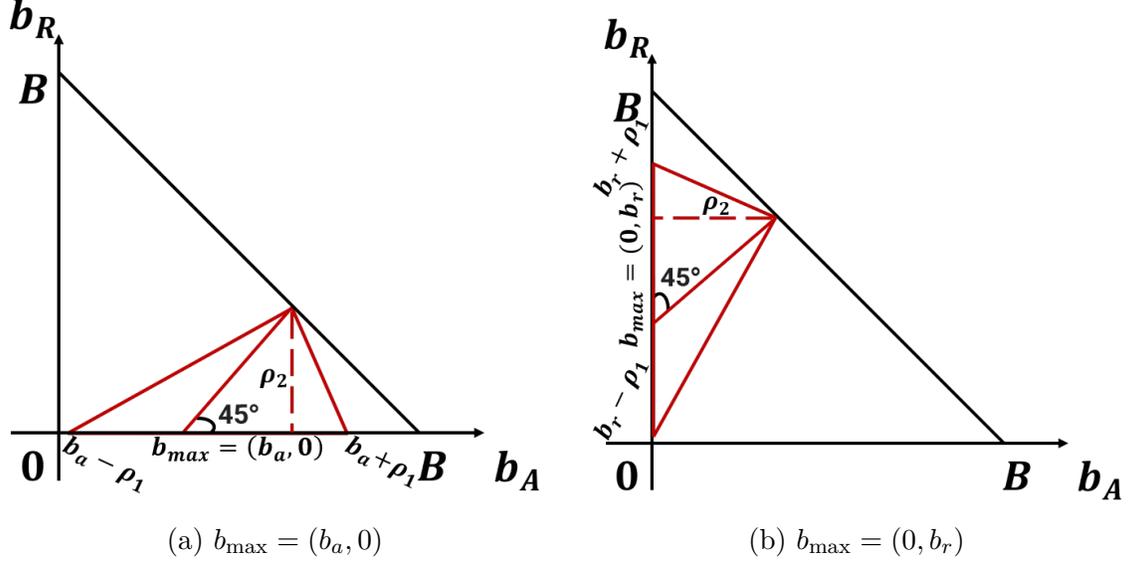


Figure 4.7: The procedure of generating an expanded triangle based on the proposed greedy MTD algorithm: This algorithm can reduce and refine the solution search space to identify a desirable  $(b_A^*, b_R^*)$  and is detailed in Section 4.5.5.

(1) We build a reward tree to store the previous evaluation results; (2) We adjust the DRL agent’s reward function to  $\mathcal{R}(s_t, a_t, s_{t+1}) = f(G'_{t+1})$  if  $t = \lceil \log_2 B \rceil$ ; 0 otherwise; and (3) We initialize multiple FSS environment instances proposed in Section 4.5.2 and evaluate them simultaneously with the same DRL policy function until the policy updates. The MTD agent makes decisions and takes actions based on the information provided by the NIDS about which nodes are compromised in the network. Since the NIDS is characterized by false positives and false negatives, the MTD agent may make decisions based on the imperfect information from the NIDS as the ground truth attack states are not available in real environments.

### 4.5.6 Practical Applicability of EVADE and DO

We discuss several practical, real-world examples of our proposed network topology adaptation strategies. In an SDN, since its vital merit is flexible manageability that can separate the data plane from the control plane, an SDN controller has been commonly used to recon-

figure a logical network topology and its flow table. Thus, packets can be forwarded based on routing instructions given by the SDN controller at node levels [? ]. Generating virtual network topologies in optical networks, called “virtual topology design,” is well-known to optimize service provision [52, 180]. In wireless sensor networks, network topology reconfiguration has been frequently considered for accurate estimates of sensed data by sensors where a gateway provides each node its next node to which a packet is forwarded [39, 96]. Moreover, by opening, sectionalizing, and closing tie switches of the network, power distribution systems perform efficient and effective network reconfiguration to minimize their power loss [37, 133]. There is a potential for service degradation when the moving target defense (MTD) is actively being applied. Thus, there is a critical tradeoff because more frequent MTD operations to enhance system security can introduce service and performance degradation due to the interruptions introduced by the system reconfiguration. We currently envision a least active implementation to limit any adverse effect on network service during the execution of the network shuffling adaptations as an MTD mechanism.

An example of day-to-day operations can include network configurations, status updates, and maintenance operations upon attacks or outages when the proposed EVADE and DO are applied in a given network. These operations can be performed based on the following procedures: (1) *Network Configurations*: A network needs to be configured online with the key design parameters that the EVADE and DO require. Furthermore, to proactively defend against existing attacks, the attack model should be known in advance. For instance, we need to configure the values of key parameters (see Table 4.1) affected by the system constraints of the current state of the network and the attack model. (2) *Network Status Updates*: As network and environmental conditions may vary due to network topology changes (e.g., node mobility or failure) or attacks, the network vulnerability needs to be reevaluated by VREN periodically. Note that EVADE and DO can be executed offline once the network status

is updated. The optimally identified network configuration can be deployed online afterward. (3) *Maintenance*: The network topology adaptation performance can be recorded online every time the MTD operation is triggered. We assume that all the topology and corresponding performance information is backed up and can provide redundancy to maintain reliability and resilience. Under some situations caused by a power outage, operational failures, or successful internal and external attacks, the offline backup information of the network configuration can be used.

## 4.6 Experimental Setup

### 4.6.1 Network Datasets

We use an Erdős-Rényi (ER) random network with 200 nodes and 1,021 edges [118]. In [176], we use three additional real networks: (1) a sparse network from an observation of the Internet at the autonomous systems level with 1,476 nodes and 2,907 edges [99]; (2) a medium dense network with 1,000 nodes and 6,123 edges derived from a backbone topology of the Internet service provider Level 3 [122]; (3) a dense network with 963 nodes and 11,310 edges derived from a traceroute-based Internet mapping provided by CAIDA Macroscopic Internet Topology Data Kit [98]. We use the original network topology for the sparse network. We derive networks of comparable size for the medium-dense and dense networks with the sparse network. To this end, we generate them using the following procedure: (1) Rank all nodes in the original network by the degree in descending order, and (2) Identify a new network as the largest connected component of the induced subgraph consisting of nodes with rankings from 1 to 1000.

Table 4.1: Key design parameters, meanings, and default values

Param.	Meaning	Value
$n_a$	Attack simulation times	500
$n_r$	Number of simulation runs	200
$n_e$	Training episodes of DRL-based schemes	1000
$N$	Total number of nodes in a network	200
$k$ [170, 172]/ $d_a$ [176]	Upper hop bound for edge addition	3
$\gamma^*$	Intrusion detection probability	0.9
$P_{fn}, P_{fp}$	False negative or positive probability	0.1, 0.05
$\mathbf{x}$	Degree of software vulnerability	0.5
$p$	Connection probability between pairs of nodes in an ER network	0.05
$l$	Number of software packages available	5
$P_a$	Fraction of initial attackers in a network	0.3
$B$	Upper bound of the total adaptation budget	500 [170, 172]/800 [176]
$P_s$	Probability of state manipulation attacks	0.3
$D_r$	Detection rate of state manipulation attacks	0.99
$P_d$	Package drop probability	0.5
$P_m$	Package modification probability	0.5
$\lambda$ [170]	Constant used in package forward failure rate	0.1
$\lambda$ [176]	Expected number of new attackers arriving in each attack round	2
$\rho_1, \rho_2$	Length parameters of the expanded triangle	64
$P_s$	Probability of state manipulation attacks	0.3
$D_r$	Detection rate of state manipulation attacks	0.8
$ADC_g$	Adaptation cycle for the greedy MTD (min.)	5
$ADC_{DRL}$	Adaptation cycle for the DRL-based MTD (min.)	20
$T$	Number of attack rounds	60
$t_{atk}$	The time interval a set of attackers arrive in a given network	1

### 4.6.2 Parameterization

We first deploy a given network and adapt the network once based on our proposed network adaptation algorithm. DRL agents would adapt the network by addressing the optimization problem in Eq. (4.1). After then, epidemic attacks are applied to the network, which is

evaluated in terms of system security and network connectivity.

In [176], for a given network,  $G(V, E)$ , the proposed hybrid MTD is triggered periodically to shuffle the network topology aiming to minimize network vulnerability while maintaining network connectivity. Specifically, greedy MTD will be triggered per  $ADC_g$  min. while the DRL-based MTD will be triggered per  $ADC_{DRL}$  min. We simulate epidemic attacks for 60 rounds, where the attack rounds happen per  $t_{atk}$  min. In each attack round, new attackers will arrive based on the Poisson distribution with  $\lambda$ , which is the expected number of new attackers arriving in one attack round. Then all existing attackers in the network will perform attacks once and all nodes' attributes (i.e.,  $na_i$  and  $nc_i$ ) will be updated accordingly. After attackers perform epidemic attacks, the network  $G$  is evaluated in terms of the size of the giant component and the fraction of the compromised nodes.

We assume attackers can perform *state manipulation attacks* (SMAs) in Section 4.4.3, with probability  $P_s$ . We assume DRL agents can detect such an attack with detection rate  $D_r$ . If the attack is detected, DRL agents would use the system state of the previous step; otherwise, it uses the falsified state. For the attack simulations to identify the expected vulnerabilities of nodes and edges (see Algorithm 2), we used  $n_a = 500$  for all our experiments. The original network is initialized by an Erdős–Rényi (ER) model [50], representing a random graph, with the total number of nodes  $N = 200$  and its connection probability,  $p = 0.05$ . All demonstrated results are based on the averages collected from  $n_r = 200$  simulation runs. We evaluate all three DRL-based schemes (i.e., DQN, Double DQN, and Dueling DQN), after  $n_e = 1,000$  training episodes.

If an attacker (i.e., compromised and undetected by the NIDS) receives a packet, it will drop a packet randomly with probability  $P_d = 0.5$ . If the attacker decides to forward the packet, it will modify the packet randomly with probability  $P_m = 0.5$ . An attacker can also send packets to a particular legitimate node. We set the probability of a node successfully

forwarding a packet to  $e^{-\lambda \cdot N_p}$  where  $\lambda = 0.1$  is a constant to model a reasonable failure rate of packet delivery in a given network and  $N_p$  is the number of packets received in a given node. An attacker also performs epidemic attacks with the infection probability based on Eq. (4.2).

We summarized the key design parameters, their meanings, and default values used in Table 4.1.

### 4.6.3 Metrics

We used the following **metrics** for our experiments:

- **Converged reward ( $\mathcal{CR}$ ):** This is the converged reward identified by a DRL agent aiming to solve the optimization problem defined in Eq. (4.1).
- **Size of the giant component ( $\mathcal{S}_G$ ):** This captures the extent of network connectivity composed of non-compromised, active nodes in a network, and measured by  $\mathcal{S}_G = \frac{N_G}{N}$ , where  $N$  is the total number of nodes in the network and  $N_G$  is the number of nodes in the giant component. Higher  $\mathcal{S}_G$  is more desirable.
- **Fraction of compromised nodes ( $\mathcal{F}_C$ ):** This measures the fraction of the number of compromised nodes (both detected and undetected) over the total number of nodes  $N$  in the network due to epidemic attacks. Lower  $\mathcal{F}_C$  is more desirable.
- **Delivery of correct messages ( $\mathcal{P}_{MD}$ )** measures the ratio of the number of correct messages delivered over the total number of the messages transmitted by a given set of source-destination pairs. This is a key indicator of measuring actual service availability of a system.

#### 4.6.4 Comparing Schemes

##### Comparing Schemes for the Validation of the DREVAN

To emphasize the necessity of each component (VREN/FSS) in our model, we consider different variants of DQN algorithms and baselines for **comparative performance analysis**:

- **DQN with DREVAN** [111]: This utilizes neural networks parameterized by  $\theta$  to represent an action-value function (i.e., Q function) and assumes that the agent can fully observe the environment. This uses VREN and FSS.
- **Double DQN with DREVAN** [154]: This has two networks, namely an online network and a target network. By updating parameters only based on the periodically updated target network, the agent could perform more stably than DQN. This uses VREN and FSS.
- **Dueling DQN with DREVAN** [160]: This also has both an online network and a target network and utilizes the dueling network architecture decoupling value and advantage. This is known for its outperformance over DQN and DDQN in Atari games. This uses VREN and FSS.
- **DQN with VREN**: This is the same as DQN, but it does not use FSS and instead uses linear search (LS).
- **DQN with FSS**: This is the same as DQN, but it does not use VREN and instead uses random adaptation (RA).
- **DQN**: This is a baseline model that does not use DREVAN (i.e., VREN and FSS). That is, this uses LS and RA.

- **Greedy:** This uses FSS to make greedy choices by looking one-step ahead at each step by choosing the subdivision returning a maximum reward.
- **Random:** This is considered as a baseline model where an agent first selects a removal budget  $b_R$  at random and then selects an addition budget  $b_A$  at random.
- **Optimal:** This identifies adaptation budgets corresponding to a maximum converged reward provided by VREN.

### Comparing Schemes for the Validation of the DeepNETAR

Given different evaluation functions as discussed in Section 4.2 (i.e., O-SG, O-MD, O-SG-MD), we use the following **six schemes for comparative performance analysis**:

- **DQN-DeepNETAR-SG** utilizes DQN with neural networks parameterized by  $\theta$  to represent an action-value function (i.e., a Q function) assuming that a DRL agent can make full observations about the environment [111]. This scheme uses DeepNETAR in Section 4.5 with the evaluation function O-SG.
- **DQN-DeepNETAR-MD** follows DeepNETAR with DQN and O-MD.
- **DQN-DeepNETAR-SG-MD** follows DeepNETAR with DQN and O-SG-MD.
- **Greedy-SG** uses the FSS algorithm to make decisions at each step by looking ahead one step and greedily choosing the subdivision that returns the maximum reward with O-SG.
- **Optimal-SG** identifies adaptation budgets maximizing O-SG.
- **Random** is the baseline model where an edge removal budget  $b_R$  and an edge addition budget  $b_A$  are selected at random.

Table 4.2: DRL parameters and values

Model	Parameter	Value
PPO	Discounting factor	0.9
	Actor learning rate	0.008
	Critic learning rate	0.08
	Clipping range	0.5
	Number of epoch when optimizing the surrogate loss	10
DQN	Discount factor	0.9
	Learning rate	0.08
	Exploration decay rate	0.85
	Minimum exploration rate	0.001

DQN is evaluated with or without using FSS or VREN. Given fixed adaptation budgets  $(b_A, b_R)$ , random adaptation (RA) is used as a baseline adaptation algorithm. We used linear search (LS) as a baseline alternative for FSS and RA as an baseline alternative for VREN. We call DQN with FSS and VREN as ‘DQN with DREVAN,’ DQN with LS and VREN as ‘DQN with VREN,’ DQN with FSS and RA as ‘DQN with FSS,’ and DQN with LS and RA as ‘DQN.’

### Comparing Schemes for the Validation of the EVADE

We consider the following algorithms to identify the budget pairs for network topology adaptations. For DRL algorithms used in EVADE, we use Proximal Policy Optimization (PPO) and Deep-Q Network (DQN) because DQN represents the most common DRL algorithm and PPO is known for its fast convergence and efficient hyperparameter optimization. Table 4.2 lists the parameters and their values used in DQN and PPO. We compare the performance of the following schemes in our experiments:

- **PPO with EVADE** uses PPO that adopts the actor-critic architecture to find the optimal policy for the DRL agent along with VREN and FSS.

- **S-G-PPO with EVADE** uses a greedy MTD to obtain the initial budget and then uses PPO.
- **DQN with EVADE** employs deep neural networks parameterized by  $\theta$  to represent an action-value function (i.e., Q function) along with VREN and FSS. In this scheme, the DRL agent is assumed to fully observe the environment.
- **S-G-DQN with EVADE** uses greedy MTD to obtain the initial budget and then uses DQN to further refine the results.
- **GA** is based on the genetic algorithm proposed in [56]. The agent selects an optimal network topology among  $n_s$  random generated candidate topologies based on Eq. (4.1). This scheme triggers MTD once every 5 min.
- **Random** is a baseline scheme where the DRL agent first selects  $b_R$  at random and then selects  $b_A$  at random. This scheme triggers MTD once every 5 min.

## 4.7 Numerical Results & Analysis

### 4.7.1 Validation of the DREVAN

#### Learning Performance and Complexity of DREVAN

In this section, we discuss the learning performance and complexities of our proposed schemes. Since we observed that Double DQN and Dueling DQN perform similar to DQN, we focus on analyzing the algorithmic complexity of DQN with DREVAN, DQN with FSS, DQN with VREN, and DQN and the speed of reaching an optimal solution in this section. As shown in Table 4.3, DQN with DREVAN and DQN with FSS have less complexity than DQN

Table 4.3: Asymptotic analysis of the compared schemes

Scheme	Complexity
DQN with DREVAN	$O(n_e \times \lceil \log_2 B \rceil \times T_{\text{train}} \times n_a)$
DQN with FSS	$O(n_e \times \lceil \log_2 B \rceil \times T_{\text{train}} \times n_a)$
DQN with VREN	$O(n_e \times B \times T_{\text{train}} \times n_a)$
DQN	$O(n_e \times B \times T_{\text{train}} \times n_a)$
Greedy	$O(\lceil \log_2 B \rceil \times n_a)$
Random	$O(n_a)$
Optimal	$O(B^2 \times n_a)$

with VREN and DQN since the proposed fractal environment could significantly reduce the number of trajectories per training episode. Here,  $n_e$  is the training episode,  $T_{\text{train}}$  is the training time per episode,  $B$  is the upper bound of total adaptation budget, and  $n_a$  is the attack simulation times. Since Greedy, Random and Optimal are rule-based algorithms that do not have the training phase, their complexities are only based on  $B$  and  $n_a$ . As shown in Table 4.3, Optimal has higher complexity than Greedy while Random is the most efficient algorithm among all. Note that the assumption in Optimal is that an agent knows all the rewards in advance, thus it will not be realistic in practice to have all the rewards available as prior knowledge.

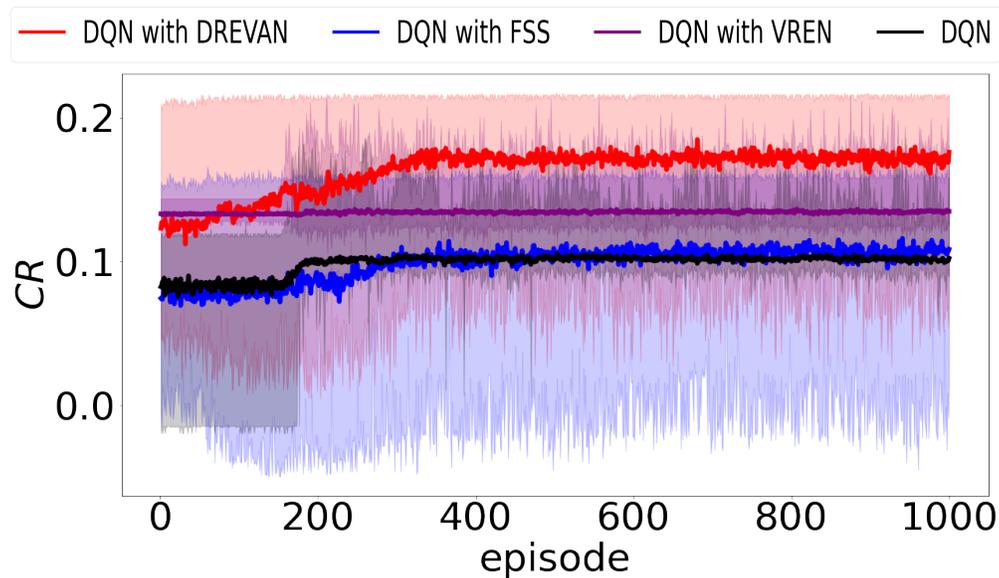


Figure 4.8: Converged reward with respect to training episodes.

Now we discuss how quickly each DRL algorithm (i.e., DQN with DREVAN, DQN with FSS, DQN with VREN, or DQN) identifies a best solution achievable. Fig. 4.8 shows the learning curve of the four DQN-based schemes with respect to training episodes under the default settings in Table 4.1. We observed that DQN with DREVAN performs the best among all. DQN with FSS can only learn a sub-optimal policy since the underlying RA performs much worse than VREN. DQN with VREN and DQN cannot learn well with LS. This clearly demonstrates that FSS provides a critical environment for a DRL agent to efficiently identify the best solution achievable.

### Sensitivity Analysis

In this section, we conducted in-depth comparative performance analysis of three types of DQN (i.e., DQN, Double DQN, and Dueling with DREVAN) and three baseline and existing counterparts (i.e., Greedy, Random, and Optimal) under varying a wide range of the number of available software packages  $l$ , the upper bound of the total adaptation budget  $B$ , probability of state manipulation attacks  $P_s$ , and detection rate of state manipulation attacks  $D_r$ . Since we already proved that DQN with DREVAN outperforms in Section 4.7.1, we did not include non-DREVAN-based DQN algorithms for comparative performance analysis in this section. Rather, to investigate further performance analysis of DREVAN-based DQN algorithms, we include two additional, advanced DQN algorithms, Double DQN and Dueling DQN.

**Effect of Varying the Number of Software Packages** Fig. 4.9 shows the effect of varying the number of software packages available ( $l$ ) on the performance of the six schemes with respect to the three metrics under the ER network model. We observed that increasing  $l$  increases converged reward ( $\mathcal{CR}$ ) and size of the giant component ( $\mathcal{S}_G$ ) while decreasing

the fraction of compromised nodes ( $\mathcal{F}_C$ ). Based on the concept of  $N$ -version programming, the number of software packages ( $l$ ) here refers to the number of versions being implemented for the same piece of software. Hence, as  $l$  increases, the level of software diversity increases, resulting in a decrease of the percentage of nodes being compromised due to attacks, an increase of the network connectivity because less nodes are being compromised. The overall performance order in terms of all three metrics is summarized as: Optimal  $\geq$  DQN  $\approx$  Double DQN  $\approx$  Dueling DQN  $\geq$  Greedy  $\geq$  Random.

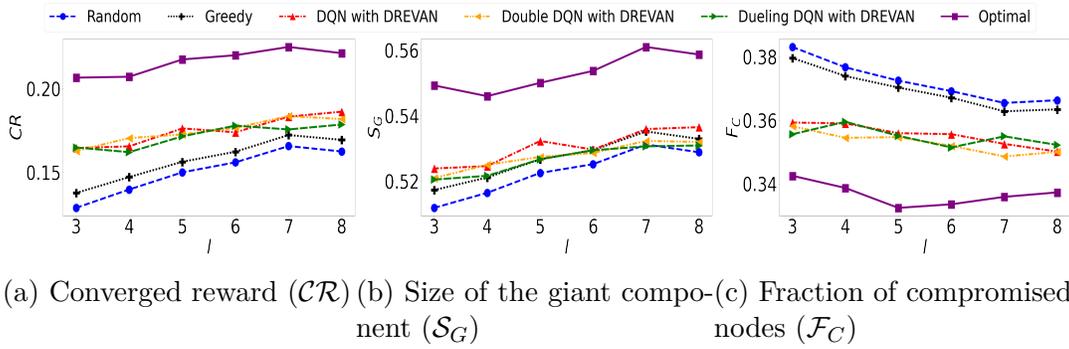


Figure 4.9: Effect of varying the number of software packages available ( $l$ ) under an ER network.

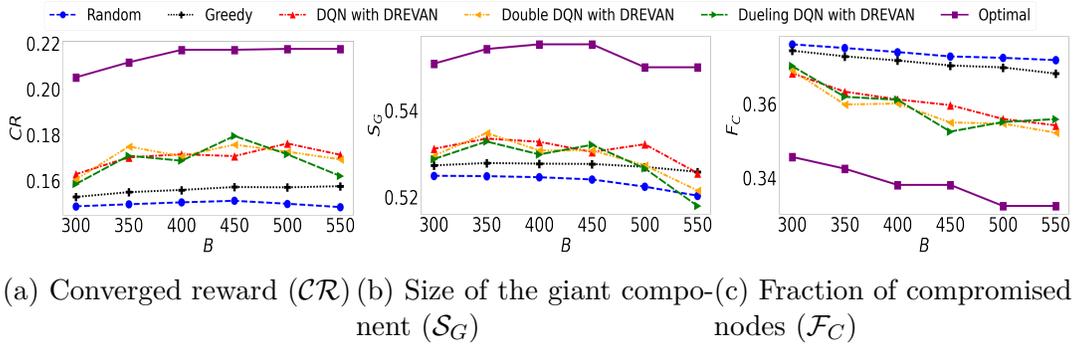


Figure 4.10: Effect of varying the upper bound of the total adaptation budget ( $B$ ) under an ER network.

**Effect of Varying the Amount of Adaptation Budget** Fig. 4.10 shows the effect of varying the upper bound of the total adaptation budget ( $B$ ) on the performance of the six schemes in terms of the three metrics under the ER network model. We observed that higher

$B$  increases  $\mathcal{CR}$  while decreasing  $\mathcal{F}_C$ . Notice that higher  $B$  does not necessarily improve  $\mathcal{S}_G$  since all schemes adapt the network based on  $\mathcal{CR}$  where edge adaptations are based on the ranks of expected vulnerabilities. The overall performance order with respect to  $\mathcal{CR}$ ,  $\mathcal{F}_C$  and  $\mathcal{S}_G$  is summarized as: Optimal  $\geq$  DQN with DREVAN  $\approx$  Double DQN with DREVAN  $\approx$  Dueling DQN with DREVAN  $\geq$  Greedy  $\geq$  Random.

Dueling DQN with DREVAN is more sensitive to  $B$  than DQN with DREVAN and Double DQN with DREVAN as it performs significantly worse under a higher budget  $B$ . This is due to the dueling structure used in Dueling DQN, which is in general much harder to be trained than the other two DQN algorithms with a larger episode length for higher  $B$ .

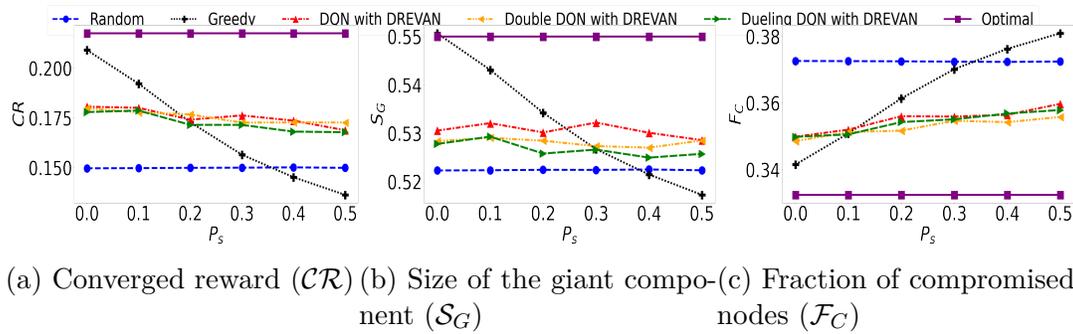


Figure 4.11: Effect of varying probability of state manipulation attacks ( $P_s$ ) under an ER network.

**Effect of Varying the Strength of State Manipulation Attacks** Fig. 4.11 shows the effect of varying the probability of SMAs ( $P_s$ ) on the performance of the six schemes in terms of the three metrics under the ER network model. We observe that higher  $P_s$  brings lower  $\mathcal{CR}$  and  $\mathcal{S}_G$  while introducing more  $\mathcal{F}_C$ . Since SMAs can be only performed when rewards are used, their severity only affects the performance of the Greedy and DRL-based schemes while no effect is introduced to Random and Optimal. Greedy is more sensitive to  $P_s$  than DRL-based schemes as it performs better than three DRL algorithms with smaller  $P_s$  but significantly worse than them with higher  $P_s$ . Again, this is due to the nature of Greedy, which looks ahead at each step for the reward of a next action.

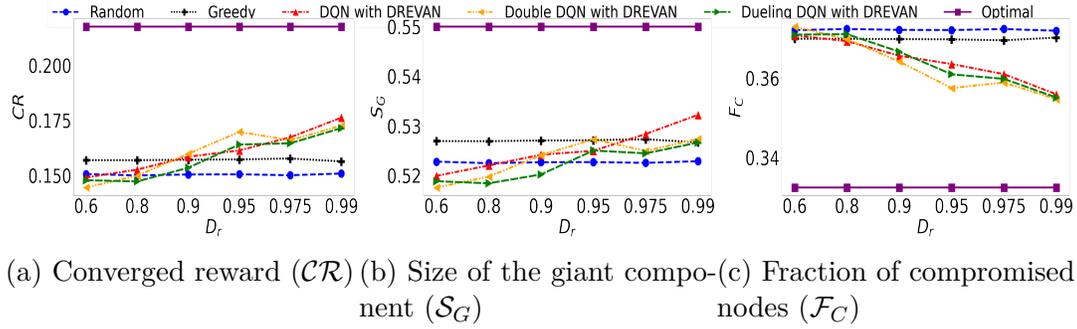


Figure 4.12: Effect of varying detection rate of state manipulation attacks ( $D_r$ ) under an ER network.

### Effect of Varying the Detection Rate of State Manipulation Attacks (SMAs)

Fig. 4.12 shows the effect of varying detection rate of SMAs ( $D_r$ ) on the performance of the six schemes in terms of the three metrics under the ER network model. We observed that higher  $D_r$  increases  $\mathcal{C}\mathcal{R}$  and  $\mathcal{S}_G$  while lowering  $\mathcal{F}_C$ . Since the detection mechanism is only implemented under DRL-based schemes, the other three schemes have no sensitivity against  $D_r$ . The results show the outperformance of DRL-based schemes only with higher  $D_r$ . We can observe that overall DQN with DREVAN performs slightly better than Double DQN with DREVAN and Dueling DQN with DREVAN.

### 4.7.2 Validation of the DeepNETAR

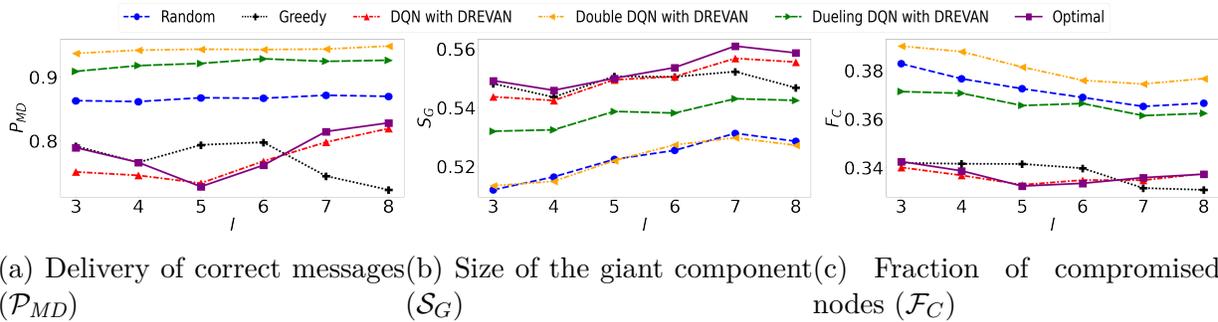


Figure 4.13: Effect of varying the number of software packages available ( $l$ ) under an ER network.

This section demonstrates the experimental results obtained from the extensive comparative

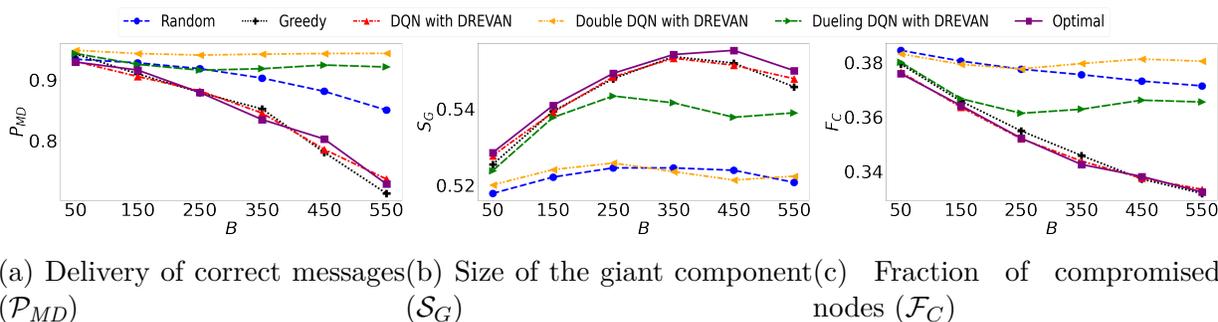


Figure 4.14: Effect of varying the upper bound of the total adaptation budget ( $B$ ) under an ER network.

performance analysis of the following schemes: DQN-DeepNETAR-SG, DQN-DeepNETAR-MD, and DQN-DeepNETAR-SG-MD, and three counterparts (i.e., Random, Greedy-SG, and Optimal-SG). We varied the number of available software packages  $l$  and the upper bound of the total adaptation budget  $B$  to investigate their impact on the three metrics (i.e.,  $\mathcal{P}_{MD}$ ,  $\mathcal{S}_G$ ,  $\mathcal{F}_C$ ).

Fig. 4.13 shows how the different number of software packages available ( $l$ ) affect the performance of each scheme in the three metrics in an ER network. We observed that larger  $l$  introduces the increase of the delivery of correct messages ( $\mathcal{P}_{MD}$ ) and size of the giant component ( $\mathcal{S}_G$ ) while decreasing the fraction of compromised nodes ( $\mathcal{F}_C$ ). Notice that high  $l$  corresponds to a high level of software diversity, which leads to the low percentage of compromised nodes and, consequently, better network connectivity and service availability. DQN-DeepNETAR-MD, with the highest  $\mathcal{F}_C$  and the highest  $\mathcal{P}_{MD}$ , sacrifices security to achieve the best service availability. Conversely, DQN-DeepNETAR-SG has the lowest  $\mathcal{F}_C$  and  $\mathcal{P}_{MD}$  by focusing more on security over service availability. DQN-DeepNETAR-SG-MD balances these two schemes by achieving a relatively high security level while achieving fairly good service availability. As shown in Fig. 4.13, in conclusion,  $\mathcal{S}_G$  is more closely related to  $\mathcal{F}_C$  than  $\mathcal{P}_{MD}$ . Specifically, high  $\mathcal{S}_G$  corresponds to low  $\mathcal{F}_C$ , but does not necessarily correspond to high  $\mathcal{P}_{MD}$ , representing service availability.

Table 4.4: Asymptotic complexity analysis of the compared schemes

Scheme	Complexity
S-G-DQN/S-G-PPO with EVADE	$O(n_e \times \lceil \log_2 B \rceil \times n_a)$
DQN/PPO with EVADE	$O(n_e \times \lceil \log_2 B \rceil \times n_a)$
GA	$O(n_s \times n_a)$
Random	$O(n_a)$

Fig. 4.14 shows how different upper bound levels of the total adaptation budget ( $B$ ) influence the performance of each scheme in terms of the three metrics. As for DQN-DeepNETAR-MD and DQN-DeepNETAR-SG-MD, we observed overall that higher  $B$  decreases  $\mathcal{P}_{MD}$  and  $\mathcal{F}_C$  since  $\mathcal{F}_C$  is much more sensitive than  $\mathcal{P}_{MD}$  with a relatively small  $B$ , which motivates the agent in each scheme to focus more on the security gain. However, once the optimal budget is identified, higher  $B$  would slightly degrade the performance since higher  $B$  corresponds to a larger search space, which makes it harder to identify the optimal adaptation budgets. In DQN-DeepNETAR-SG, its performance in  $\mathcal{F}_C$  continues increasing under higher  $B$ . Note that larger  $B$  does not necessarily increase  $\mathcal{S}_G$  since all schemes perform network adaptations with their corresponding evaluation functions, and  $\mathcal{S}_G$  is closely related to  $\mathcal{F}_C$ . Also notice that different schemes have different optimal  $B$  with respect to  $\mathcal{P}_{MD}$ ,  $\mathcal{S}_G$  and  $\mathcal{F}_C$ . For example, the optimal  $B$  of DQN-DeepNETAR-MD and DQN-DeepNETAR-SG-MD is near 250 while the optimal  $B$  of DQN-DeepNETAR-SG, Greedy-SG, Optimal-SG, and Random is identified at about 350.

### 4.7.3 Validation of the EVADE

#### Algorithmic Complexity Analysis of EVADE

We discuss the algorithmic complexity of S-G-DQN/S-G-PPO with EVADE, DQN/PPO with EVADE, GA, and Random. Table 4.4 shows the asymptotic complexities in Big- $O$  notation of the six schemes considered in this work. We notice that S-G-DQN/S-G-PPO with EVADE

incurs the similar low cost as DQN/PPO with EVADE because the proposed FSS can significantly decrease the number of trajectories per training episode. Note that  $n_e$  refers to the training episode,  $B$  is the maximum total adaptation budget allowed (i.e., upper bound budget),  $n_a$  is the number of attack simulations, and  $n_s$  is the number of random sample topologies generated. Since the GA and Random algorithms are rule-based and do not have the training phase, their complexities are represented by  $n_a$  and  $n_s$ . Furthermore, our proposed DRL schemes could be faster than GA given a high convergence speed, i.e.,  $n_e \ll n_s$ . Table 4.4 shows that the GA and Random are the most efficient algorithms among all while showing poor performance in  $\mathcal{S}_G$  and  $\mathcal{F}_C$ .

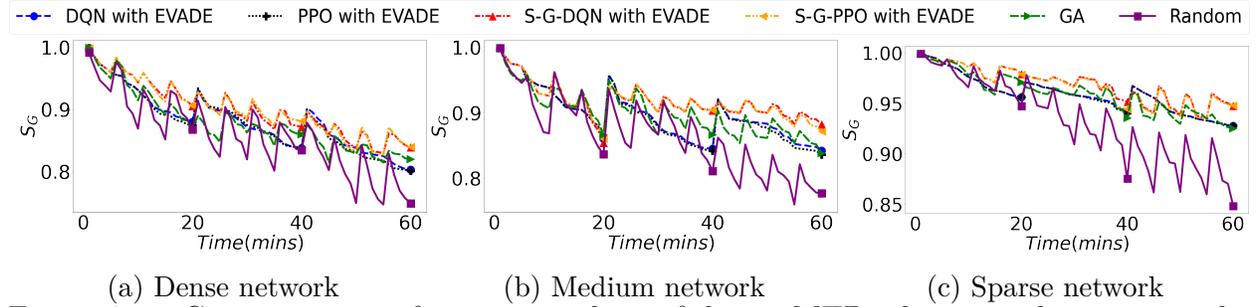


Figure 4.15: Comparative performance analysis of the six MTD schemes with respect to the simulation time in terms of the size of the giant component ( $\mathcal{S}_G$ ).

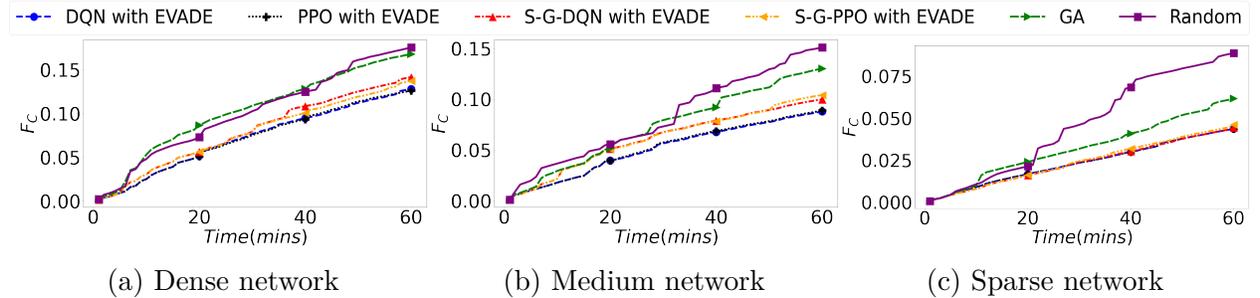
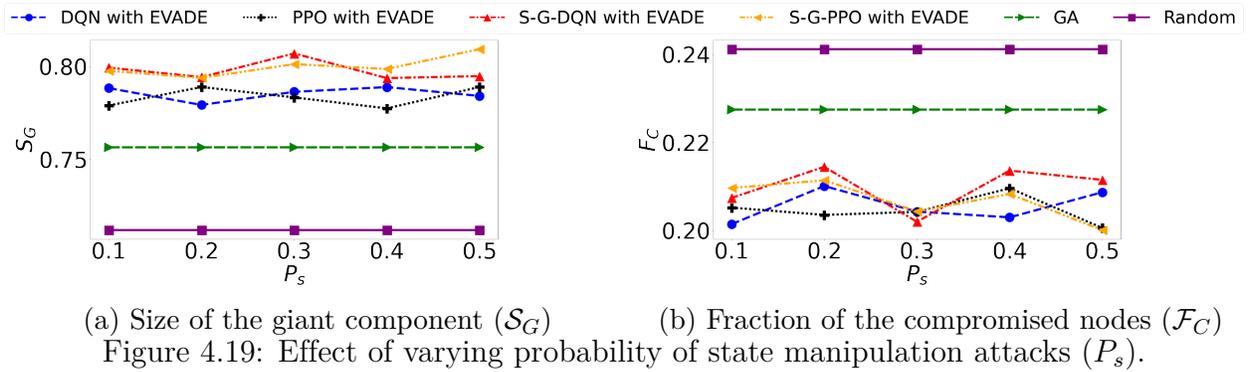
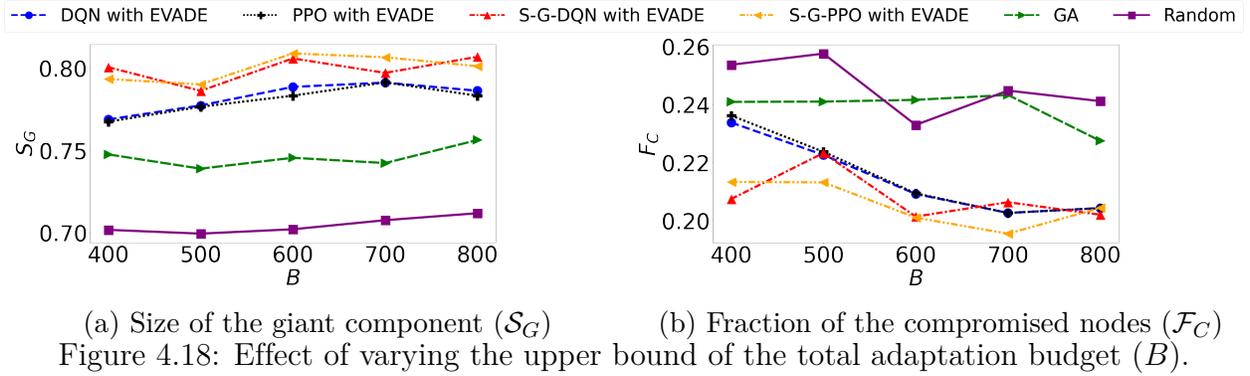
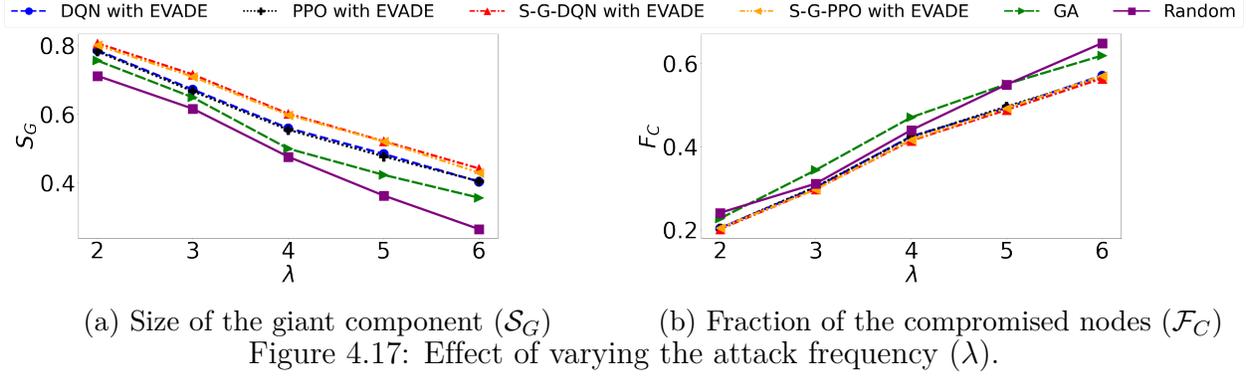


Figure 4.16: Comparative performance analysis of the six MTD schemes with respect to time in terms of the fraction of compromised nodes ( $\mathcal{F}_C$ ).

## Comparative Analyses

For the three real network topologies, we conducted comprehensive comparative analyses of two types of hybrid schemes (i.e., S-G-DQN, S-G-PPO), two DRL schemes (i.e., DQN, PPO),



and two baselines (i.e., GA and Random). In Figs. 4.15 and 4.16, we observe that when MTD operates for an hour,  $\mathcal{S}_G$  decreases while  $\mathcal{F}_C$  increases due to the incoming attack frequency  $\lambda$ . There are some bumps in  $\mathcal{S}_G$  curves where each bump is aligned with the time the MTD is triggered for the network topology adaptations. With the same adaptation frequency, GA performs worse than hybrid schemes, S-G-DQN/S-G-PPO, and DQN/PPO (without greedy). This shows the effectiveness of our proposed hybrid MTD schemes and DRL-based MTD schemes. In general, EVADE-based schemes outperform under the sparse network. Based

on Eq. (4.3), network vulnerability is related to the sum of edge vulnerabilities. Thus, a fewer number of edges means less edge vulnerability for each edge and thus less network vulnerability overall. Furthermore, the edge number also affects the effectiveness of VREN. A lower edge number leads to better convergence of rankings given by VREN, thus showing better adaptation performance. This is shown in the results where all schemes perform worst and our proposed EVADE-based schemes outperform least in a dense network. The overall performance order in the two metrics (i.e.,  $\mathcal{S}_G$  and  $\mathcal{F}_C$ ) is: S-G-PPO with EVADE  $\approx$  S-G-DQN with EVADE  $\geq$  PPO with EVADE  $\approx$  DQN with EVADE  $\geq$  GA  $\geq$  Random.

### Sensitivity Analyses

For the Erdős-Rényi (ER) random network [117], we conducted in-depth sensitivity analyses of two types of hybrid schemes (i.e., S-G-DQN, S-G-PPO), two DRL schemes (i.e., DQN, PPO), and two baselines (i.e., GA and Random) under varying a wide range of the expected number of new attackers arriving in the network  $\lambda$ , the upper bound of the total adaptation budget  $B$ , and the probability of state manipulation attacks  $P_s$ .

Fig. 4.17 shows the effect of varying the expected number of new attackers arriving in the network,  $\lambda$ , on the performance of the six schemes in terms of the two metrics in the network. We observe that increasing attack frequency ( $\lambda$ ) decreases  $\mathcal{S}_G$  while increasing  $\mathcal{F}_C$ . The hybrid approaches outperform other schemes as they use DRL and greedy MTD with a higher adaptation frequency. Fig. 4.18 explains how the total adaptation budget ( $B$ ) can affect the performance of the six schemes in terms of the two metrics in the given network. Higher  $B$  enables the increased  $\mathcal{S}_G$  while decreasing  $\mathcal{F}_C$  for all MTD schemes. This result implies that higher budgets enable the agent to find more desirable network topologies and lead to higher performance. Fig. 4.19 shows how the different levels of SMAs ( $P_s$ ) impact the performance of the six MTD schemes using the two metrics. We found that our EVADE

outperforms other baselines and counterparts and clearly shows high resilience against SMAs. Recall that SMAs can perform attacks only with DRL-based approaches.

## 4.8 Key Findings

From this study, we obtain the following **key findings**:

- Our proposed DREVAN uses a fractal-based environment (FSS) that can significantly reduce the training complexity of our DRL algorithms. This consequently improves the speed of convergence to an optimal solution with a higher converged reward.
- Our proposed DREVAN uses a vulnerability-aware ranking algorithm (i.e., VREN) to strategically adapt edges for the network to be reconfigured effectively and efficiently.
- Our DREVAN using three different types of Deep Q-learning algorithms (i.e., DQN with DREVAN, Double DQN with DREVAN, and Dueling DQN with DREVAN) showed higher performance than the counterpart and baseline schemes particularly in minimizing system vulnerability while maintaining comparable or better network connectivity under a wide range of key design parameter values.
- By setting an appropriate evaluation function, DQN-DeepNETAR-SG-MD can better ensure security, connectivity, and service availability simultaneously.
- As a network connectivity metric, the size of the giant component is more related to security rather than actual service availability under epidemic attacks.
- By setting different evaluation functions, our proposed DeepNETAR is a generic framework that can handle multiple, competing objectives.

- Our **EVADE** uses a density optimization (DO)-based greedy algorithm to further reduce the search space for DRL algorithms, along with **VREN**. Using DO enabled our hybrid MTD approach to converge even faster with a smaller solution space and result in the outperformance of **EVADE** over the existing counterparts.
- Our **EVADE** was evaluated based on two different DRL algorithms, including Deep Q-learning Networks (DQN) and Proximal Policy Optimization (PPO), along with their corresponding hybrid approaches (i.e., S-G-DQN with **EVADE** and S-G-PPO with **EVADE**), and two existing baselines (i.e., GA and Random). The hybrid **EVADE** showed acceptable asymptotic complexity compared to their effectiveness. In addition, they showed higher performance than the counterpart and baseline schemes in minimizing system vulnerability while maintaining comparable or better network connectivity.

# Chapter 5

## Energy-Adaptive Monitoring for Resilient Smart Farms

This chapter addresses the **Fault-Tolerance Task**. This chapter is based on the paper “An Attack-Resilient and Energy-Adaptive Monitoring System for Smart Farms” published in The 2022 IEEE Global Communications Conference (GLOBECOM), Dec. 2022 [173] and the paper “Attack-Resistant, Energy-Adaptive Monitoring for Smart Farms: Uncertainty-Aware Deep Reinforcement Learning Approach” submitted to IEEE Internet of Things Journal [175].

### 5.1 Research Goal & Motivation

According to the Food and Agriculture Organization (FAO) of the United Nations [51], the food production rate must increase by a factor of up to 70 percent to absorb the increase in population, estimated as more than 9 billion people by 2050 [135]. To support farms’ productivity, flexibility, or availability, smart farm technologies have developed by leveraging sensors, Internet-of-Things (IoT), edge and cloud computing technologies. Smart farm research is applied to develop agricultural business practices [34], improve monitoring of animal welfare [144], and provide data sensing and environmental controls [144]. Smart farm research also investigated efficient data transmission considering CPU usage, signal strength,

and battery operation time [77] for wireless sensor networks [120].

However, existing research lacks efforts to develop secure solutions for wireless sensor networks with energy constraints such as ones powered by solar energy harvesting. According to the World Health Organization (WHO), over half a million people died due to food contamination caused by bacteria, viruses, toxins, or chemicals. Cyber attacks on farms, transportation systems, and food processing industrial control systems to distort and disrupt the handling of correct data can worsen the problem. Any distortion in the data received from the livestock monitoring systems can lead to serious situations such as the spread of disease, possible pandemics, and the provision of wrong information to potential customers of the livestock [61].

In this work, we are interested in improving the accuracy of the livestock monitoring system in farms under the presence of cyber attacks that can forge, modify, or drop sensed data from sensors to gateways or edge devices, or inject false data. Most wireless sensor networks (WSNs) are unable to accurately record biometrics for cattle because battery-powered sensors attached to the collar of the cattle can last only a few days or weeks. Frequent replacing or recharging of batteries for sensor nodes is laborious for a farm with a large number of animals. To address this problem, we consider wireless solar sensor nodes attached to the livestock's ears, which are powered by solar energy harvesting. Due to the small size of the solar panel attached to a sensor node, the amount of solar energy harvested is low. Moreover, the harvested energy level fluctuates as the livestock or its ear moves, which can make sensing and transmission of the data unstable.

To address the problem, we consider sensor nodes adopting two communication protocols, LoRa (Long Range) [148] and BLE (Bluetooth Low Energy) [162]. The LoRa protocol aims for long-distance communication with a range of several *kms*, but the data rate is only 27 *kbps*. Contrarily, the BLE protocol aims for a short distance with a line-of-sight distance

of 100 *meters* and a higher data rate of 2 *Mbps*. Furthermore, the BLE protocol drains considerably less power than the LoRa one. Fig. 5.1 shows the WSN system considered in this work. A sensor node monitors the energy level of its battery. A sensor node with a high energy level transmits its sensed data with LoRa to LoRa gateways, and the gateways upload the data to the cloud server accessible to the user. A sensor node with a low energy level may not be able to send its data directly to a LoRa gateway due to insufficient energy. Instead, the sensor node seeks a nearby sensor node with a high energy level. If it finds one, the sensor node sends its data by BLE to the nearby sensor node with a high energy level, and the nearby sensor node transmits the received data to LoRa gateways.

Under this scenario, since there may be multiple sensor nodes with low energy requesting to transmit their sensed data to the sensor node with high energy, a decision on which sensed data to transmit to the gateways can significantly impact the accuracy of monitored data. Instead of continuously transmitting sensed data that are already received sufficiently and hence high quality (i.e., low uncertainty), a sensor node with high energy can select a sensor node with low energy whose data have high uncertainty, and transmit the data, resulting increase in the data certainty. The process will significantly increase the certainty of the overall data monitored from all animals on the farm. To this end, we introduce an uncertainty-aware transmission policy based on the assessment by LoRa gateways. Specifically, a LoRa gateway can request sensor nodes to send sensed data of particular animals whose monitored data have trended high uncertainty (i.e., low certainty). In this work, we leverage deep reinforcement learning (DRL) to identify sensor nodes whose data need to be transmitted to improve the overall monitoring accuracy.

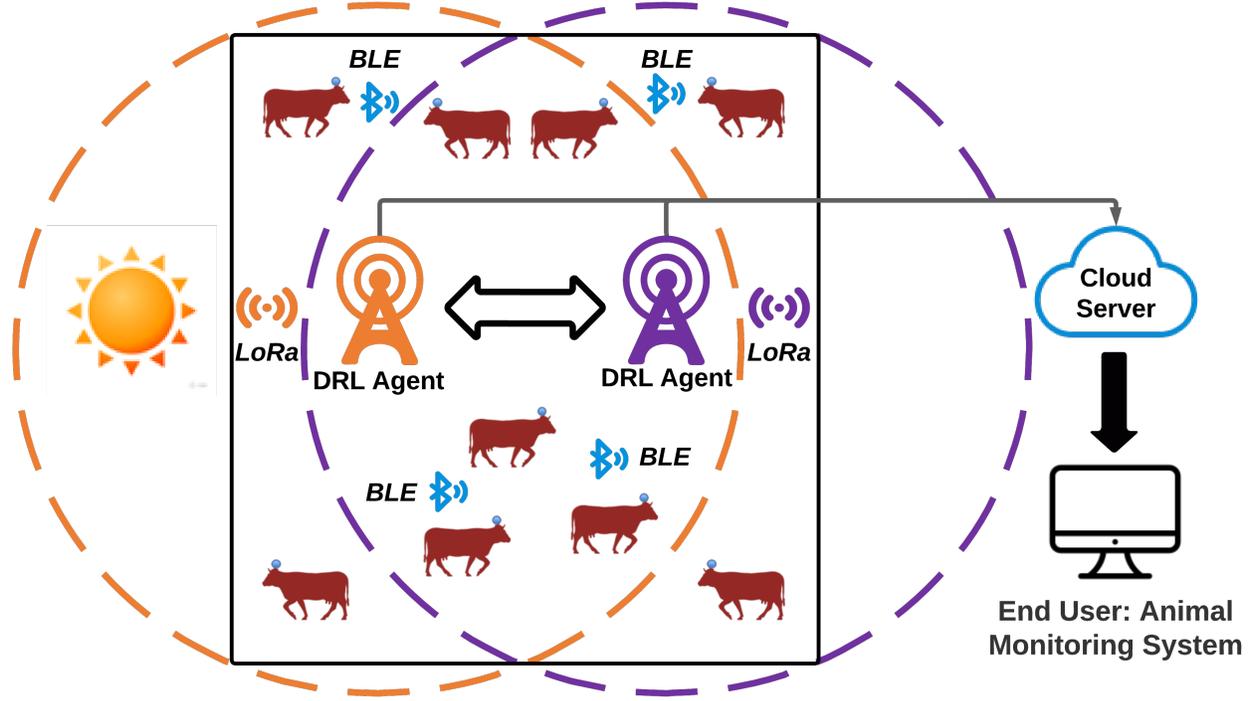


Figure 5.1: Wireless solar sensor node-based smart farm network.

## 5.2 Problem Statement

In this work, we aim to minimize the monitoring error rate (i.e., a gap between the sensed data aggregated from sensors and the ground truth; see Eq. (5.2)) and system overload (i.e., a mean fraction of the failed requests of all requests sent from low-energy sensors; see Eq. (5.3)) in a sensor network by identifying an *optimal policy*. An updated policy  $\mathcal{P}_T = \{p_1, p_2, \dots, p_T\}$  contains monitoring actions  $p_i$ , where  $i \in [1, T]$ ,  $p_i \in \mathcal{P}_T$  and  $\mathcal{P}_T$  is a set of monitoring actions available to the sensor in every monitoring step. Given a dynamic sensor network  $\mathcal{G}_T = \{g_1, \dots, g_i, \dots, g_T\}$ , the objective function is defined by:

$$\begin{aligned} \arg \max_{\mathcal{P}_T} \sum_{i=1}^T f(g_i(p_1, p_2, \dots, p_i)), \\ \text{s.t. } \forall i \in [1, T], p_i \in \mathcal{P}_T, \end{aligned} \quad (5.1)$$

where  $f(g)$  is based on the evaluation function  $f : g \mapsto -\mathcal{ME}(g) - \mathcal{OL}(g)$ , aiming to minimize the monitoring error rate  $\mathcal{ME}$  and system overload  $\mathcal{OL}$ , that is detailed:

$$\mathcal{ME} = \frac{\sum_{t \in T} \sum_{x \in X} \text{me}_t^x}{NT|X|}, \quad \text{me}_t^x = \sum_{j=1}^n D(\text{eo}_t^x(j), \text{gt}_t^x(j)), \quad (5.2)$$

$$s.t. \quad D(a, b) = \begin{cases} 1 & \text{if } a \neq b; \\ 0 & \text{if } a = b. \end{cases}$$

Here  $T$  is the total monitoring time,  $N$  is the number of animals,  $\text{me}_t^x$  is the overall monitoring error rate of all  $N$  animals' conditions of attribute  $x$  at time  $t$ ,  $\text{eo}_t^x(j)$  and  $\text{gt}_t^x(j)$  are the estimated and ground truth observation of animal  $j$ 's condition in  $x$  attribute at time  $t$ , respectively.

$$\mathcal{OL} = \frac{1}{T} \sum_{t \in T} \frac{rq_t^f}{rq_t}, \quad (5.3)$$

where  $T$  is the total monitoring time,  $rq_t^f$  and  $rq_t$  are the numbers of failed requests and total requests at time  $t$ , respectively. Determining an optimal update policy to achieve multiple objectives is non-trivial given the complexity involved in solving a multi-objective optimization problem [30]. This is discussed in detail based on the experimental results discussed in Section 5.7.

### 5.3 Key Contributions

To tackle this problem, we made the following **key contributions** in this work:

- We propose an energy-adaptive monitoring system for WSN-based smart farms with solar-powered sensor nodes attached to cattle. This is the first work that considers how WSN-based smart farms can maintain high monitoring quality under limited and

fluctuating energy availability due to solar energy harvesting in the smart farm.

- We develop uncertainty-aware DRL algorithms based on *Deep Reinforcement Learning* (DRL) [45] and *Subjective Logic* [80] (SL) to minimize the overall monitoring error and system overload for sensor nodes. The uncertainty is measured in two dimensions based on SL, i.e., *vacuity* due to a lack of evidence and *dissonance* due to conflicting evidence.
- We provide mathematical proof that can justify how SL's uncertainty maximization technique contributes to reducing monitoring errors. From the theoretical analysis, we found that using the uncertainty maximization technique can lead to using more recent evidence, reflecting recent network dynamics more appropriately.
- We consider a comprehensive attack model including seven attack types to evaluate the robustness of the proposed uncertainty-aware DRL-based monitoring system for the smart farm.
- We devise a novel *monitoring error rate metric* that can evaluate the monitoring quality independent of monitoring data distributions. The developed monitoring error rate metric enables our proposed monitoring system to handle multi-dimensional heterogeneous data simultaneously.
- We identify an optimal deployment setting of LoRa gateways on which DRL agents run to maximize the chances for solar sensors to deliver their sensed data within the gateway wireless radio range.
- We provide the asymptotic complexity analysis of our proposed uncertainty-aware DRL-based algorithms. This analysis reveals a critical tradeoff between robustness/-effectiveness vs. efficiency.

- We validate the performance of the proposed uncertainty-aware DRL-based monitoring system using real datasets obtained from Virginia Tech’s Smart Farm Innovation Network. Furthermore, we design a framework where healthy sensor nodes generate synthetic datasets similar to real datasets, and compromised sensor nodes are modeled as attackers following the attack model for testing the robustness of our uncertainty-aware DRL-based algorithms against adversarial attacks.
- We conduct extensive sensitivity analyses to investigate the effect of key designs and environmental factors on performance of two proposed uncertainty-aware DRL-based algorithms (deep Q-learning, DQN, and multi-agent proximal policy optimization, MAPPO) against three baseline models (greedy, random, DM), including the attack severity, the initial sensor node energy level, the number of solar sensors, and the chance for sensor nodes to be exposed to the sun.

## 5.4 System Model

This section discusses the network, node, and attack models.

### 5.4.1 Network Model

Our target WSN consists of solar sensors attached to the cattle, that continuously measure the bio-metric information and transmit the sensed data to the LoRa gateway, which then aggregates and transmits the clustered data to the cloud. Given the relatively low cost of transmitting data over long ranges (LoRa) via the standard IP protocol for IoT devices, the LoRa gateways act as the optimal intermediary between the sensor nodes and the cloud server. In the given smart farm network (see Fig. 5.1), using BLE (Bluetooth Low Energy)

each low-battery sensors (LBS) can relay its sensed data to one of the high-battery sensors (HBS). The high-battery sensor can then send the received sensed data along with its own data to the LoRa gateway via LoRa. The overall data flow is shown in Fig. 5.2. We assume that each sensor has a Microchip SAM R34/35 microcontroller with an embedded LoRa radio, which dissipates 170 *mW* during transmission, while the microcontroller itself dissipates only 8 *mW* in active mode. The LoRa protocol is ideal for long distance communication with a distance coverage of several *kms* and a data rate of 27 *kbps*. Contrarily, the BLE protocol is purposed for short distance communication with a radius coverage of 100 *meters* and a data rate of 2 *Mbps*. Furthermore, the BLE protocol drains considerably less power than the LoRa protocol. For example, only 11 *mW* of power is dissipated during transmission for a Texas Instruments CC2640R2F micro-controller chip with an embedded BLE radio [149]. Therefore, sending a single bit of data takes about 1,100 times less energy for the Texas Instruments CC2640R2F micro-controller chip when compared with the LoRa radio of the SAM R34/35 microcontroller chip. We assume that the initial battery level of each deployed sensor is 5 *kWs*, which is equivalent to a full charge. Outdoor solar has a power density of about 10 *mW/cm<sup>2</sup>* whereas indoor light has a power density of 0.1 *mW/cm<sup>2</sup>* [109]. For a solar panel of 5 *cm*, the maximum harvestable power for indoor light is about 2 *mW* and outdoor light is about 200 *mW*. Fig. 5.1 describes the considered network model in this work, describing a smart farm environment with solar sensors attached to the cattle.

With the main objective of minimizing the monitoring error rate and system overload, a DRL agent is deployed at every LoRa gateway to shortlist, select and prioritize which animal's sensed data is required, at regular intervals. The process of identifying the important data, by the DRL agent is described in Section 5.5. We assume that for energy saving there is no encryption when the sensors communicate with each other via BLE and hence, malicious entities can intercept the data in transmission and modify/forge data. Additionally, an

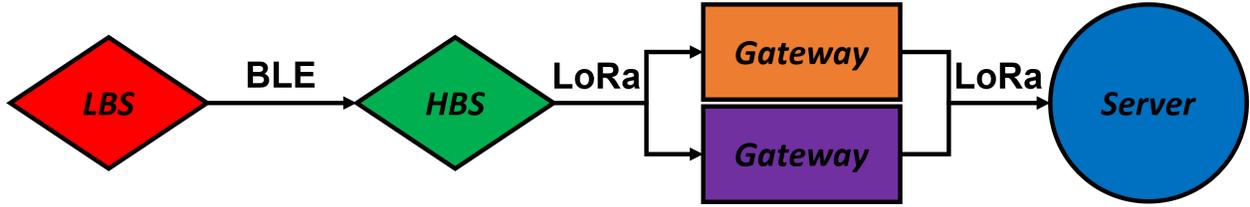


Figure 5.2: The overall data flow of the smart farm network.

attacker can imitate a sensor by using its authentication key with the gateway and sending false data for the sensor itself as well as for other low-battery sensors. We assume that the communication between the LoRa gateway and the cloud server is secure and encrypted based on traditional secret cryptography. As shown in Fig. 5.1, multiple LoRa gateways each running a DRL agent can collaborate with each other in sharing collected sensed data received from sensors.

## 5.4.2 Node Model

Sensor nodes in a given smart environment are assumed solar-powered and deployed as implants and can transmit data on request. Depending upon the animal’s movement and the varying weather condition from day to day, the battery levels of the sensor may fluctuate throughout the day. Therefore, it is essential to utilize the energy wisely for high availability, consistency, and sustenance. Each sensor node  $i$  is characterized by  $sn_i^t = [\text{temp}_i^t, \text{hb}_i^t, \text{ma}_i^t, \text{bl}_i^t]$ , where  $\text{temp}_i^t$  refers to sensor node  $i$ ’s temperature at time  $t$  in Celsius,  $\text{hb}_i^t$  is the number of  $i$ ’s heartbeat at time  $t$ ,  $\text{ma}_i^t$  is  $i$ ’s speed at time  $t$  and  $\text{bl}_i^t$  is  $i$ ’s battery life at time  $t$  scaled in  $[0, 100]$  in percent. Most of the sensor’s energy will be used to transmit the sensed data to the LoRa gateway. In contrast, considerably less battery will be used for communication between the sensor and other nearby sensors via BLE as it consumes roughly 1,100 times less (see Section 5.4.1).

Utilizing the data reported by the sensors to the LoRa gateway, each DRL agent will try to

maximize the monitoring quality by selecting what data is needed with priority to accurately estimate the well-being of all the animals on the farm. To this end, the sensor nodes in the WSN is categorized into high battery-level sensors (HBS) and low battery-level sensors (LBS) based on the recommended battery level  $T_M$ . Since we are only interested in transmissions from LBS to HBS, we model the sensor network as a directed bipartite graph. Section 5.5 describes the actions performed by the DRL agent running on every LoRa gateway. The end-user will get the efficient monitoring results of the smart farm from the cloud server, which aggregates data on individual animals from various LoRa gateways. This work aims to evaluate how the DRL agent on LoRa gateways can enhance the quality of animal monitoring in the presence of cyberattacks and fluctuating sensor energy levels.

### 5.4.3 Attack Model

This work considers the two classes of attacks in terms of cyber attacks and adversarial examples. First, we consider the following **cyber attack** behaviors:

- *Non-compliance to the protocol:* A sensor node can be compromised and exhibit non-compliant behavior to the request by the DRL agents on LoRa gateways. For example, when an animal  $A$ 's sensed data is requested by a DRL agent, the attacker can either not send  $A$ 's data or send another sensor's data to the LoRa gateway. We model this with the attacker's non-compliance probability,  $P_{NCA}$ .
- *False data injection:* An attacker (e.g., a compromised sensor) can transmit forged or modified data or inject false data to gateways. In addition, man-in-the-middle attackers (MIMAs) can intercept data being transmitted in the middle and replace it with forged or modified data. The attackers can inject false data during the training phase (i.e., poisonous attacks) or the testing phase (i.e., evasion attacks). We call the

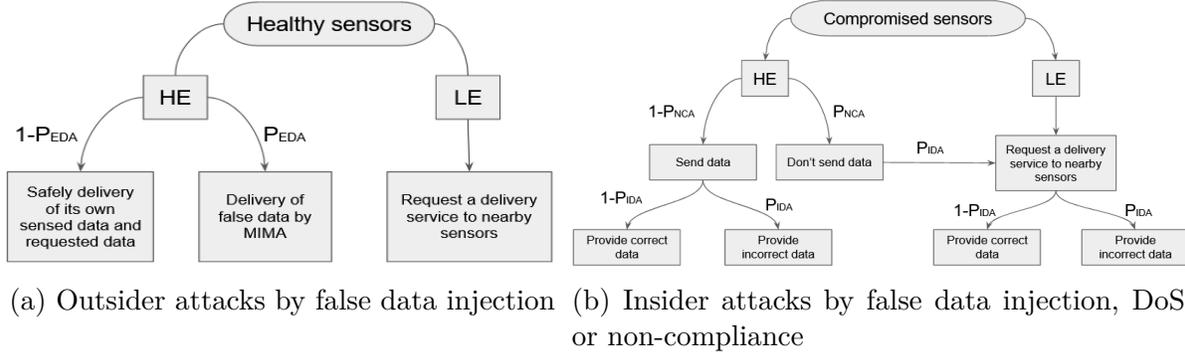


Figure 5.3: Attack scenarios by both outsider and insider attackers. Recall that  $P_{EDA}$  is the probability of an external attacker performing false data injection attacks,  $P_{NCA}$  is the probability of an internal attacker performing non-compliance attacks, and  $P_{IDA}$  is the probability of an inside attacker performing false data injection attacks or DoS attacks.

compromised sensors *internal attackers* while calling the external attackers intercepting sensed data for forgery or modification or injecting false data *external attackers*. These attacks are modeled by the forging or modifying data attacks an internal or external attacker can launch,  $P_{IDA}$  and  $P_{EDA}$ , respectively.

- *Denial-of-Service (DoS)*: A compromised high-battery sensor can send a request to nearby sensors requesting them to send its fake sensed data. As it is a type of internal attack, we also model this DoS attack probability by  $P_{IDA}$ . This can drain other sensors' energy levels quickly but is wasted in sending false data. To avoid an infinite loop, we assume the attacker will request sending its fake sensed data to legitimate sensors.

We summarize the above three types of attacks in Fig. 5.3.

We also consider **adversarial examples** aiming to disrupt DRL agents' operations as follows:

- *Fast Gradient Sign Method (FGSM)*: This state manipulation attack model is firstly proposed in [58] to generate adversarial examples in image classification tasks. To

apply it in the context of DRL-based algorithm execution, we generalize DRL settings by considering actions as class labels. To make a fair comparison, we use the original loss function of each DRL algorithm to compute the gradients. Since we have multiple DRL agents in our smart farm system, we assume the attack will happen in both local agent states and global agent observations. We define the attack probability as  $P_{FGS}$ .

- *Momentum Iterative Method (MIM)* [46]: MIM is a gradient-based adversarial attack to boost the success rate of the generated adversarial examples. Unlike other gradient-based methods to maximize the loss function, MIM manipulates a velocity vector for the gradient direction in the loss function at each step as an iterative attack method. We apply the MIM attack to each of our DRL schemes generating adversarial states using the loss function. We model the attack probability by  $P_{MIM}$ .
- *Projected Gradient Descent (PGD)* [106]: This attack is a multi-step variant FGSM and uses a negative loss function to maximize the inner part of the saddle point formulation. For each DRL agent, we compute the gradient and generate corresponding adversarial examples using their loss functions. This attack probability is modeled by  $P_{PGD}$ .
- *Basic Iterative Method (BIM)* [90]: This attack is a variant of the fast method to constrain each adversarial example in the  $\epsilon$ -neighborhood of the original example. The adversarial examples are computed by applying the fast method multiple times with a small step size, allowing the low computational cost to perform BIM. This attack is applied with adversarial states impacting the original loss function where the attack probability is modeled by  $P_{BIM}$ .

## 5.5 Proposed Approach: DRL-based Animal Monitoring

In this section, we provide a detailed description of our proposed uncertainty-aware DRL-based algorithms for smart farm animal monitoring.

### 5.5.1 Uncertainty-Aware Animal Monitoring

First, based on received data from sensors in the past, a gateway can estimate uncertainty in each animal's condition, such as heart rate, average temperature, minimum/maximum temperature, average activity, battery level of a sensor worn by the animal, and timestamp. Recall that each LBS will send its sensed data to a HBS within 100 meters. Hence, we distinguish direct sensed data from indirect sensed data in terms of whether a sensor sent its own sensed data or another sensor's sensed data. If the sensor with high energy transmitted other sensor's sensed data, it is treated as indirect sensed data. Otherwise, it is direct sensed data. The gateway periodically reports its collected data from sensors to the cloud server. Since the given network has multiple gateways, the corresponding multiple DRL agents will share information about sensed data and estimate each animal's conditions (see Table 5.1) and associated confidence level on each observation item. A database on the gateway keeps recording all animals' condition data where sensed data by sensor  $i$  (i.e., ID) for an animal are stored.

Each observation item's condition (e.g., average temperature) will be reported as one of the  $K$  classes of the range, e.g., for the temperature reading,  $K = 5$  meaning that there are 5 classes of ranges: 35 or below, 36-37, 38-39, 40-41, 42 or above. The end user can then easily determine if the temperature is normal based on the cloud server's received data. Since

Table 5.1: EVD dataset description

Metric	Description
Serial	A unique animal identifier
Heart rate	Heart bits per min.
Average-temperature	Average body temperature in Celsius
Min-temperature	Minimum temperature in Celsius
Max-temperature	Maximum temperature in Celsius
Average-activity	Average activity recorded by the number of steps taken
Battery-level	Residual battery life
Timestamp	Date and time of transmission

the gateway will periodically report average conditions for all animals to the cloud, it will aggregate sensed data from sensor nodes and measure their average with the probability of a condition being with  $K$  classes and multiple types of uncertainty values. To utilize the concept of uncertainty, we will apply SL [81] to compute an opinion on each animal's condition in a given attribute (e.g., temperature, heartbeats, activity, or battery level).

### SL-based Formulation of a Multinomial Opinion

SL can explicitly express uncertainty caused by a lack of evidence, called *vacuity* in its opinion representation. In addition, SL can consider base rates as prior probabilities in a Bayesian way to formulate a second-order opinion and corresponding uncertainty estimates, where a second-order opinion is represented by Dirichlet distribution. We will use a Dirichlet probability density function (PDF) to model the distribution of class probabilities and corresponding uncertainty masses. In SL, given a random variable  $X$  and its sample space  $\mathbb{X}$ , a multinomial opinion of  $X$  is represented by  $\omega_X = (\mathbf{b}_X, u_X, \mathbf{a}_X)$ . We call  $x \in \mathbb{X}$  as a belief mass and  $|\mathbb{X}|$  is the number of belief masses. The additivity requirement of  $\omega_X$  is given as  $\sum_{x \in \mathbb{X}} \mathbf{b}_X(x) + u_X = 1$ . To be specific, each parameter indicates,

- $\mathbf{b}_X$ : belief mass distribution over  $\mathbb{X}$ ;

- $u_X$ : *uncertainty mass* representing *vacuity of evidence*;
- $\mathbf{a}_X$ : *base rate distribution* over  $\mathbb{X}$ .

The projected probability distribution of multinomial opinions is given by:

$$\mathbf{P}_X(x) = \mathbf{b}_X(x) + \mathbf{a}_X(x) \cdot u_X, \quad \forall x \in \mathbb{X} \quad (5.4)$$

The base rate for belief  $\mathbf{b}_X(x_i)$ , which is  $\mathbf{a}_X(x_i)$ , means the prior preference over the  $x_i$  belief (e.g., a class). If no preference is given, we consider the base rate equally for each belief mass, i.e.,  $\mathbf{a}_X(x_i) = 1/|\mathbb{X}|$  for any  $x_i$ .

Given the amount of evidence supporting belief  $x_i$  is  $\mathbf{r}(x_i)$ , the observed evidence in the Dirichlet PDF can be mapped to the multinomial opinions as:

$$\mathbf{b}_X(x) = \frac{\mathbf{r}(x)}{W + \sum_{x_i \in \mathbb{X}} \mathbf{r}(x_i)}, \quad u_X = \frac{W}{W + \sum_{x_i \in \mathbb{X}} \mathbf{r}(x_i)}, \quad (5.5)$$

where  $W$  refers to the amount of uncertain evidence. Commonly,  $W = |\mathbb{X}|$ .

### Estimation of Multiple Types of Uncertainty

SL categorizes uncertainty into two primary sources [81]: (1) basic belief uncertainty derived from single belief masses, and (2) intra-belief uncertainty based on the relationships between different belief masses. These two sources of uncertainty can categorize the two uncertainty types, *vacuity* and *dissonance*, respectively, that correspond to vacuous beliefs and contradicting beliefs. In particular, the vacuity of an opinion  $\omega_X$  is captured by uncertainty mass

$u_X$  while dissonance of an opinion,  $\mathbf{b}_X^{\text{Diss}}$ , is formulated by [80]:

$$\mathbf{b}_X^{\text{Diss}} = \sum_{x_i \in \mathbb{X}} \left( \frac{\mathbf{b}_X(x_i) \sum_{x_j \in \mathbb{X} \setminus x_i} \mathbf{b}_X(x_j) \text{Bal}(x_j, x_i)}{\sum_{x_j \in \mathbb{X} \setminus x_i} \mathbf{b}_X(x_j)} \right), \quad (5.6)$$

where the relative mass balance between a pair of belief masses  $\mathbf{b}_X(x_j)$  and  $\mathbf{b}_X(x_i)$  is expressed by:

$$\text{Bal}(x_j, x_i) = 1 - \frac{|\mathbf{b}_X(x_j) - \mathbf{b}_X(x_i)|}{\mathbf{b}_X(x_j) + \mathbf{b}_X(x_i)}. \quad (5.7)$$

The dissonance estimation is useful to measure the *inconclusiveness* of an opinion even under a large amount of evidence that almost equally supports each singleton belief.

In this work, we regard each reported data from sensor nodes to a gateway as evidence. For instance, in a temperature report, if 38 C is reported,  $b_2$  (i.e.,  $b_1 =$  lower than normal,  $b_2 =$  normal,  $b_3 =$  higher than normal) should be updated based on Eq. (5.5). When the uncertainty mass becomes zero, an opinion will not be updated anymore, which means new evidence cannot be properly utilized in the latest opinion. To avoid this, we will deploy the *uncertainty maximization* technique [81] to reduce the impact of conflicting evidence while transforming the amount of conflicting evidence into the vacuity of an opinion.

Given opinion  $\omega_X = (\mathbf{b}_X, u_X, \mathbf{a}_X)$  where  $\mathbf{P}_X = \mathbf{b}_X + \mathbf{a}_X \cdot u_X$  for a domain  $X$ , the corresponding vacuity-maximized opinion is denoted by  $\ddot{\omega}_X = (\ddot{\mathbf{b}}_X, \ddot{u}_X, \mathbf{a}_X)$  where  $\ddot{u}_X$  and  $\ddot{\mathbf{b}}_X$  are computed by:

$$\ddot{u}_X = \min_i \left[ \frac{\mathbf{P}_X(x_i)}{\mathbf{a}_X(x_i)} \right], \quad (5.8)$$

$$\ddot{\mathbf{b}}_X(x_i) = \mathbf{P}_X(x_i) - \mathbf{a}_X(x_i) \cdot \ddot{u}, \quad \text{for } x_i \in X.$$

We use a threshold  $\rho$  to trigger the vacuity maximization. That is, Eq. (5.8) above can be

triggered only when  $u_X < \rho$  where  $\rho$  is sufficiently low (e.g., 0.05). The purpose of updating  $\omega_X$  to  $\ddot{\omega}_X$  is to allow the opinion to be further updated by receiving new evidence or being combined with other opinions, which is possible only when  $u_X > 0$ .

In a given category  $X$ , the animal condition is estimated as an opinion,  $\omega_X$ , where the corresponding uncertainty types, vacuity and dissonance are estimated, respectively, at the gateways that aggregate sensed data and transmit the average condition value in a given category along with the belief masses and uncertainty masses associated with  $\omega_X$ .

### 5.5.2 Data Aggregation at LoRa Gateways

For each sensor node, it will send its sensed data to LoRa gateways or a high-energy sensor close to it, as the information shown in Table 5.1. After receiving the reports from all sensor nodes capable of transmitting their data, a LoRa gateway will compute an opinion based on the received data. The opinion is composed of belief and uncertainty in terms of vacuity and dissonance masses. We define the opinion as a *monitoring opinion* (MO) and denote it as  $\Delta$  below. Specifically, given time step  $t$ , for the agent  $i$  and animal  $j$ , we define the attribute sets  $\mathbb{S}_{ij}^t = \{\text{temp}_{ij}^t, \text{hb}_{ij}^t, \text{ma}_{ij}^t\}$ , where  $\text{temp}_{ij}^t, \text{hb}_{ij}^t, \text{ma}_{ij}^t$  are temperature, the number of heart-beat, speed of animal  $j$  recorded by agent  $i$  at time  $t$  respectively. We let the random variable  $X_{ij}^t$  be the animal condition status of animal  $j$  recorded by agent  $i$  at time  $t$  and define the corresponding sample space  $\mathbb{X}_{ij}^t = \{\text{below normal range, normal range, above normal range}\}$ . The for each sensor report  $x_{ij}^t$  about an attribute  $s \in \mathbb{S}_{ij}$ ,  $x_{ij}^t \in \mathbb{X}_{ij}$  and  $\Delta(i, j, t) = \Delta(\omega_{X_{ij}^t}) = (u_{X_{ij}^t}, \mathbf{b}_{X_{ij}^t}^{\text{Diss}})$ . When the amount of sensed data becomes large enough, the MO may not be effectively updated from new sensed data because vacuity approaches zero based on Eq. (5.5). To update the MO properly from received new evidence, we will deploy the vacuity (uncertainty) maximization technique in Eq. (5.8). We use a threshold  $\rho$  (i.e.,  $0 < \rho < 1$ ) to

determine when to update the MO based on Eq. (5.8). That is, if  $u_X < \rho$ , this evidence will update an opinion based on Eq. (5.8). To evaluate the system monitoring error based on Eq. (5.2), we also introduce a system database to collect the latest monitoring opinions for each animal among all gateways.

### 5.5.3 DRL-based Monitoring Update

This section describes how DRL agent's states, actions, and immediate reward are formulated in this work.

#### State Space ( $\mathcal{S}_t$ )

We assume a partially observable environment where each DRL agent can only access information in the dataset of the local gateway it is running on. We formulate the state space of each agent  $i$  at time  $t$  indicating the total number of local reports for the duration of  $t$ ,  $\{\ell_{i,1}, \ell_{i,2}, \dots, \ell_{i,t-1}, \ell_{i,t}\}$ , where  $\ell_{i,t^*}$  refers to the number of local reports in  $[t^* - 1, t^*]$ . To be specific, assume  $t \in [0, T]$ , the overall state space is given by  $\mathcal{S}_t = \{s_{1,t}, \dots, s_{2,t}, \dots, s_{m,t}\}$ , where  $m$  is the number of DRL agents (i.e., LoRa gateways) and  $s_{i,t}$  is the state space for agent  $i$  at time  $t$ , which is given by  $s_{i,t} = \{\ell_{i,1}, \ell_{i,2}, \dots, \ell_{i,t}, 0, \dots, 0\}$ .

#### Action Space ( $\mathcal{A}_t$ )

For each DRL agent, it will choose  $k$  animals whose data is more helpful in improving monitoring quality and reducing system overload. Note that a certain amount of redundant information is desired since there is a possible situation that sensors fail to transmit data due to limitations of their energy level or topology. For the agent  $i$  and animal  $j$ , the utility

of animal  $j$  is given by:

$$\text{utility}_{ij} = (1 - u_{X_{ij}^t}) + (1 - \mathbf{b}_{X_{ij}^t}^{\text{Diss}}) + \text{fr}_{ij}^t + f(\text{bl}_{ij}^t), \quad (5.9)$$

where  $u_{X_{ij}^t}$ ,  $\mathbf{b}_{X_{ij}^t}^{\text{Diss}}$ , and  $\text{fr}_{ij}^t$  are vacancy, dissonance, and degree of freshness of animal  $j$ 's sensed data at time  $t$  by DRL agent  $i$ .  $\text{fr}_{ij}^t$  is formulated by  $\text{fr}_{ij}^t = e^{-\phi\mathbb{T}(t)}$ , where  $\mathbb{T}(t)$  is the time elapsed from the last update and  $\phi$  is a constant to normalize the freshness.  $f(x)$  is defined by  $f(x) = -(x - T_M)^2$  where  $x$  is set to  $\text{bl}_{ij}^t$ , the battery life of sensor  $j$  at time  $t$  by DRL agent  $i$ . By scaling  $u_{X_{ij}^t}$ ,  $\mathbf{b}_{X_{ij}^t}^{\text{Diss}}$ ,  $\text{bl}_{ij}^t$ , and  $\text{fr}_{ij}^t$  in  $[0, 1]$ , we set each component of  $\text{utility}_{ij}$  to  $[0, 1]$  as a real number. Here  $T_M$  denotes the recommended level that the battery of a sensor node should be maintained. A list of animal IDs will be calculated based on Eq. (5.9) in ascending order, so each agent will request data for the top  $k$  animals. The action space has three actions selecting the first  $k$  animal IDs such that  $k \in \{0, \lfloor \frac{nl}{2} \rfloor, nl\}$ , where  $nl$  is how many LBS nodes are in the current environment. In this way, the size of action space is not dependent on  $nl$  (i.e., 3), which is able to reduce the computation load raised by infinite action spaces and make this monitoring system possible for applying to larger-scale sensor networks as a generalization. For a lower  $k$ , it may impact the monitoring quality. At the same time, a higher  $k$  will obtain a larger amount of unnecessary data transmission and result in a system overload. Therefore, the DRL agent aims to identify the best action, which is the optimal  $k$  for this setting.

### Immediate Reward ( $r_t$ )

This is formulated by  $r_t^i = f(g_t^i(k_1^i, k_2^i, \dots, k_t^i), g_t)$  based on  $f(g_t^i, g_t) = -\mathcal{M}\mathcal{E}(g_t^i) - \mathcal{O}\mathcal{L}(g_t)$  given in Eq. (5.1), where  $g_t^i$  and  $k_t^i$  are the local sensor network and action with respect to gateway  $i$  at time step  $t$ .

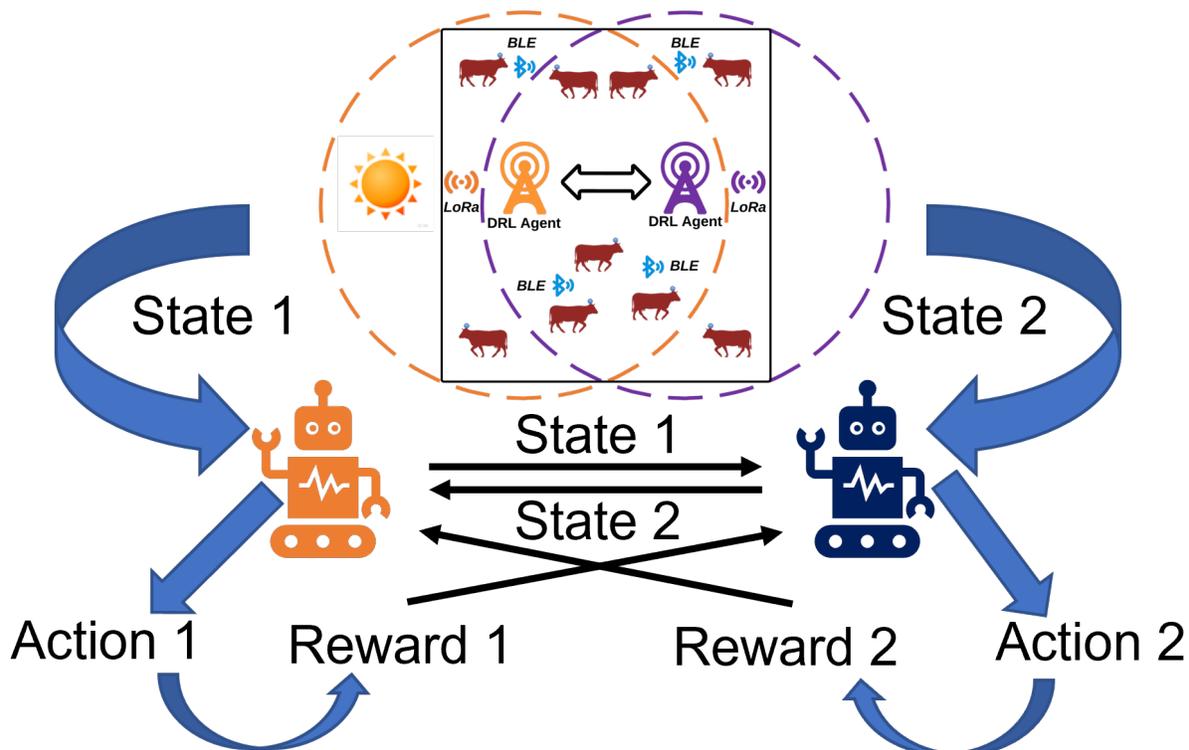


Figure 5.4: The proposed Multi-Agent Deep Reinforcement Learning (MADRL) framework.

In this work, we consider a cooperative framework where multiple DRL agents share their states and rewards to obtain the global observation of the whole farm area. Fig. 5.4 provides an overview of the proposed Multi-Agent Deep Reinforcement Learning (MADRL) framework.

#### 5.5.4 Mathematical Proof of Effectiveness Using Uncertainty Maximization

In this section, we formally prove the effectiveness of the *uncertainty maximization* technique [81] by mathematical proof. We observe that uncertainty mass and the monitoring error rate can be viewed as functions of evidence. Based on Eq. (5.5), the uncertainty (*vacuity*) drops when the amount of received evidence increases. Furthermore, given Eq. (5.6),

*dissonance* solely depends on the distribution of belief masses without vacuity being involved. Thus, the dissonance can be viewed as a constant when there is enough evidence from the same distribution.

The uncertainty (vacuity) maximization in Eq. (5.8) reinitializes the vacuity by transforming previous evidence from the belief masses to the uncertainty mass. Given the following [81],

$$j = \arg \min_i \left[ \frac{\mathbf{P}_X(x_i)}{\mathbf{a}_X(x_i)} \right], \quad (5.10)$$

where  $\mathbf{P}_X(x_i)$  is the projected probability of having  $x_i$  and  $\mathbf{a}_X(x_i)$  is the base rate (i.e., prior belief) that supports a belief mass  $x_i$ . Then, we have the updated belief masses and the uncertainty (vacuity) mass given by:

$$\ddot{\mathbf{b}}_X(x_k) = \frac{\mathbf{r}(x_k) - \mathbf{r}(x_j)}{W + \sum_{x_i \in \mathbb{X}} \mathbf{r}(x_i)}, \quad \ddot{u}_X = \frac{W + K\mathbf{r}(x_j)}{W + \sum_{x_i \in \mathbb{X}} \mathbf{r}(x_i)}, \quad (5.11)$$

where  $W$  is the amount of non-informed evidence (i.e., uncertain evidence),  $K$  is the number of belief masses (e.g., classes), and  $\mathbf{r}(x)$  is the amount of evidence supporting belief mass  $x$ . Since the amount of non-informed evidence,  $W$ , increases to  $W + K\mathbf{r}(x_j)$ , we replace  $W$  with  $W + K\mathbf{r}(x_j)$  in the denominators of both  $\ddot{\mathbf{b}}_X(x_k)$  and  $\ddot{u}_X$  as

$$\ddot{\mathbf{b}}_X(x_k) = \frac{\mathbf{r}(x_k) - \mathbf{r}(x_j)}{(W + K\mathbf{r}(x_j)) + \sum_{x_i \in \mathbb{X}} (\mathbf{r}(x_i) - \mathbf{r}(x_j))}, \quad (5.12)$$

$$\ddot{u}_X = \frac{W + K\mathbf{r}(x_j)}{(W + K\mathbf{r}(x_j)) + \sum_{x_i \in \mathbb{X}} (\mathbf{r}(x_i) - \mathbf{r}(x_j))}. \quad (5.13)$$

The above implies that the updated vacuity only considers partial history evidence, indicating more recent evidence than past evidence.

As shown in Eq. (5.2), the monitoring error rate is closely related to the amount of evidence. Assume that at each time step  $t \in [0, T]$ , the information of  $n_t$  animal is being updated

and each animal  $j$ 's information is updated  $m_j$  times in total. Hence, we have the expected monitoring error rate given by,

$$\begin{aligned}
E(\mathcal{ME}) &= \frac{E(\sum_{t \in T} \sum_{x \in X} \text{me}_t^x)}{NT|X|} & (5.14) \\
&= \frac{\sum_{t \in T} \sum_{x \in X} E(\sum_{j=1}^n D(\text{eo}_t^x(j), \text{gt}_t^x(j)))}{NT|X|} \\
&= \frac{\sum_{x \in X, t \in T} E(\sum_{j=1}^{n_t} 0 + \sum_{j=n_t+1}^n 1)}{NT|X|} \\
&= \frac{\sum_{x \in X} \sum_{t \in T} (N - n_t)}{NT|X|} \\
&= 1 - \frac{\sum_{x \in X} \sum_{j=1}^n m_j}{NT|X|}.
\end{aligned}$$

Here  $T$  is the total monitoring time,  $N$  is the number of solar sensors attached to cows,  $\text{me}_t^x$  is the overall monitoring error rate of all  $N$  cows' conditions of attribute  $x$  at time  $t$ ,  $\text{eo}_t^x(j)$  and  $\text{gt}_t^x(j)$  is the estimated and ground truth observation of cow  $j$ 's condition in  $x$  attribute at time  $t$ , respectively. The above derivation proves that  $E(\mathcal{ME})$  is only dependent upon  $n_t$  and it increases when  $n_t$  decreases. In addition, each animal  $j$ 's monitoring error rate is only related to the amount of corresponding evidence,  $m_j$ . Therefore, we prove that the order of any pair of sensors remains invariant under the partial order relations of vacuity and monitoring error rate.

## 5.6 Experimental Setup

This section describes the datasets, parameterization, DRL schemes, and performance metrics used for the experiments conducted in this work.

### 5.6.1 Datasets

At Virginia Tech, we have a collection of interconnected data collection and analysis hubs called the *SmartFarm Innovation Network* (TM), which is designed to facilitate the testing and demonstration of emerging technologies throughout the state. From the smart farm, we obtained sample datasets collected from four different sensors, namely, EmbediVet Implantable Temperature Device (EVD), Halter Sensor, Heart Rate Sensor, and Implantable Temperature Sensor. The dataset from the EVD consists of 8 components as described in Table 5.1. We consider 6 components out of them, except the serial number and timestamp, as the sensed data to represent the physical conditions of animals. The temperature and the heart rate sensor provide us with temperature in Celsius and heart rate in beats per minute (*bpm*), respectively. The Halter sensor could identify each animal's geolocation and assess its motion and posture to report its activity level. Since the existing dataset obtained from Virginia Tech's smart farm does not include any data compromised by attackers, we designed a framework where each sensor could generate synthetic datasets similar to real datasets and some compromised sensors are modeled as attackers following the attack model described in Section 5.4.3.

### 5.6.2 Parameterization

We consider 20 cows within a 40 acres ( $\sim 160\text{K}$  square meters) square farm area ( $A$ ) with 402 meters in length ( $a$ ). We consider two gateways with the same circular coverage. We further assume that these gateways could cover the farm area and each of them is covered by the other. In general, for a given number of  $m$  gateways with the same radius  $r$ , we aim to find the minimum radius  $r_m$  such that the farm area is fully covered by the total gateway coverage and each gateway is covered by other gateways to enable mutual communications.

To this end, we solve the following optimization problem to identify the minimum radius  $r_m$ :

$$r_m = \min_r \{r : \exists P_i = (x_i, y_i) \in \mathbb{R}^2\}, \quad (5.15)$$

$$\text{where } 1 \leq i \leq m, \quad \text{s.t. } \forall P = (x, y) \in [-a/2, a/2]^2,$$

$$(\min_i d(P, P_i) \leq r) \wedge \forall (i, j) \in [1, m]^2, \quad d(P_i, P_j) \leq r,$$

where  $a$  is the length of farm side,  $P_i$  is the center of gateway, and  $d(\cdot, \cdot)$  is the Euclidean distance function. We only consider the case when  $m = 2$ , where gateways locate in  $(-\frac{a}{4}, 0)$  and  $(\frac{a}{4}, 0)$  respectively with the same radius  $\frac{\sqrt{5}a}{4}$ . Fig. 5.1 shows how two gateways are optimally deployed with the corresponding wireless radio ranges used in our smart farm network environment.

To model the availability of solar energy based on the sun's movement in a day, we define a charging probability distribution  $P(x, y, t)$  over the farm area as the probability of being charged if a sensor is located at  $(x, y)$  at time  $t$ . For simplicity, we assume that  $P(x, y, t)$  has a quadratic form at time  $t$  and is represented by  $P(x, y, t) = \max\{0, -\frac{1}{6}(t - t_{xy})^2 + 1\}$ , where  $t_{xy}$  is a function of location  $(x, y)$  based on the farm's direction. We consider a square farm with its center at the origin and  $x$  axis towards the west. Thus,  $t_{xy}$  is formulated as  $t_{xy} = \frac{t_0}{a} \times (x - \frac{a}{2}) + 12$ , where  $t_0$  is a hyper-parameter. In general, to model different weather conditions, we can use a weight  $\alpha$  to discount the charging probability as  $\alpha P(x, y, t)$  with  $0 \leq \alpha \leq 1$ .

Each cow's attributes are collected by an attached solar-powered sensor. We adopt normal distributions  $\mathcal{N}(38, 1^2)$  and  $\mathcal{N}(1.5, 0.1^2)$  to describe a cow's temperature [38] and velocity [94] respectively. The cow's heartbeat is modeled as two uniform distributions:  $\mathcal{U}(60, 84)$  when it moves or  $\mathcal{U}(48, 60)$  when it does not move [38]. We use  $P_i^{mv}$  for cow  $i$ 's moving probability.

For an opinion about a cow's attributes, we will simply categorize based on three beliefs, i.e., lower than normal, normal, and higher than normal. The normal ranges of a cow's temperature, heart rate, and moving activity are given [37.8, 39.2] Celsius, [48, 84] number of beats per min., and [1, 2] meters per sec., respectively. We consider the number of uncertain evidence to be three where each belief mass has the same base rate (i.e., 1/3) [81].

We consider 24 consecutive hours as the total monitoring session. For every  $T_a = 60$  sec, each gateway would take an action to identify an optimal monitoring strategy. We assume 5 HBS with a full initial battery level and 15 LBS with random initial battery levels based on  $T_L$ . When the battery level is below  $T_M$ , LBS could only broadcast their own data to HBS via BLE. Each sensor can broadcast at most two sets of sensed data to each LoRa gateway within the wireless range per  $T_u = 30$  sec. In this way, each HBS can send its own data and another set of data requested by the LBS. The monitoring system would derive the consolidated priority list of update lists from gateways. Then the system would leverage the Hopcroft–Karp algorithm [69] to solve the maximum matching problem in bipartite sensor networks. In this way, the system could ensure the maximum number of transmissions being executed.

As for energy consumption, message transmissions need 170  $mW$  per sec and the sleep mode costs 2  $mW$  per sec. We assume a sensor is only activated for message transmissions. A sensor can be charged by the outdoor solar with 200  $mW$  per sec. In this way, a sensor can be charged  $200 \text{ mW} \times 6 \text{ h} = 4.32 \text{ kW}s$  under 6 hours of the sun.

We set all attack probabilities to  $P_A$ . For inside attackers, we initially pick them among the total number of sensors at random. For outside attackers (i.e., MIMAs), we pick a set of nodes at random transmitting messages intercepted by MIMAs with  $P_{EDA}(P_A)$ . The attackers launch attacks based on Fig. 5.3. Finally, we consider FGSM as a state manipulation attack to disturb the DRL training phase. Table 5.2 summarizes the key design parameters, their

meanings, and default values.

Table 5.2: Key design parameters, their meanings and default values

Param.	Meaning	Value
$m$	The number of gateways	2
$N$	The number of sensors(cows)	20
$T_M$	A minimum battery level to transmit sensed data by a sensor	30%
$LBS/HBS$	Low/High battery level sensors	/
$P_i^{mv}$	Cow $i$ 's probability to move	[0.3, 0.7]
$P_A$ [173]	Probability for an attacker or a compromised node to perform an attack (i.e., $P_{NCA}$ , $P_{IDA}$ , $P_{EDA}$ , $P_{FGS}$ )	0.3
$P_A$ [176]	Probability for an attacker or a compromised node to perform a common attack (i.e., $P_{NCA}$ , $P_{IDA}$ , $P_{EDA}$ )	0.3
$P_{ADV}$	Probability for an attacker to perform an adversarial attack (i.e., $P_{FGS}$ , $P_{MIM}$ , $P_{PGD}$ , $P_{BIM}$ )	0
$A$	Area of a given smart farm	40 acres
$a$	length of a given smart farm	402 m
$\rho$	Uncertainty maximization threshold	0.05
$t_0$	Hyper-parameter used in sun model	0.2
$T_u$	Time interval for a sensor to send sensed data	30 s
$T_a$	Time interval for a gateway to take an action to adjust $k$	60 s
$T_L$	Initial battery level for low battery level sensors	30%
$\alpha$	The probability for a cow to be exposed to the sun depending on its position when the sun is available	1

### 5.6.3 Metrics

We consider the following metrics for our experiments:

- **Accumulated reward ( $\mathcal{R}$ ):** This metric represents the sum of the mean accumulated reward for all DRL agents through all simulation runs.
- **Monitoring error rate ( $\mathcal{ME}$ ):** This metric is measured based on the mean difference between the latest data of each animal's condition from all gateways and the ground truth data of the corresponding animal's condition. We measure  $\mathcal{ME}$  by Eq. 5.2

introduced in Section 5.2:

- **Overload ( $\mathcal{OL}$ ):** This metric evaluates the system overload by the mean fraction of the failed requests over all sent requests from LBS. We measure  $\mathcal{OL}$  following Eq. 5.3 in Section 5.2.
- **Uncertainty ( $\mathcal{U}$ ):** Given  $m$  LoRa gateways and  $n$  animals, this metric refers to the average uncertainty observed from sensed data at time  $t$  and is estimated by:

$$\mathcal{U}_t = \frac{\sum_{i=1}^m \sum_{j=1}^n (u_{X_{ij}^t} + \mathbf{b}_{X_{ij}^t}^{\text{Diss}})}{2mn}, \quad (5.16)$$

where  $u_{X_{ij}^t}$  and  $\mathbf{b}_{X_{ij}^t}^{\text{Diss}}$  are vacuity and dissonance of animal  $j$ 's sensed data at time  $t$  by DRL agent  $i$ .

- **Battery maintenance level ( $\mathcal{BML}$ ):** Given  $m$  LoRa gateways and  $n$  animals, this metric measures the distance between the recommended energy level and the current battery level by:

$$\mathcal{BML}_t = \frac{\sum_{i=1}^m \sum_{j=1}^n f(\text{bl}_{ij}^t)}{2mn}$$

where  $f(\text{bl}_{ij}^t)$  is defined in Eq. 5.9.

- **Freshness ( $\mathcal{FR}$ ):** Given  $m$  LoRa gateways and  $n$  animals, this metric measures the overall sensor data freshness by:

$$\mathcal{FR}_t = \frac{\sum_{i=1}^m \sum_{j=1}^n \text{fr}_{ij}^t}{2mn}$$

where  $\text{fr}_{ij}^t$  is defined in Eq. 5.9. The unrecorded data has freshness 0.

### 5.6.4 Comparing Schemes

We develop two uncertainty-aware DRL algorithms (MADQN and MAPPO) whose performance is compared against three baseline schemes (Greedy, Random, and DM), described as follows:

- **Multi-Agent Deep Q-Learning (MADQN)** [111]: DRL agents learn a state-value Q function to select the optimal actions. In a multi-agent scenario, we extend DQN to MADQN, where each DRL agent learns an independent local Q function. We consider two variants of MADQN using UM or not and name them MADQN-UM and MADQN-NUM, respectively.
- **Multi-Agent Proximal Policy Optimization (MAPPO)**: MAPPO extends the PPO [138] to a multi-agent environment to mitigate non-stationarity by adopting a global critic value function to guide each local actor value function. We consider two variants of MAPPO with and without using uncertainty maximization. We name them MAPPO-UM and MAPPO-NUM, respectively.
- **Greedy Algorithm**: DRL agents make heuristic choices by enumerating all actions at each step and choosing the one with the optimal reward.
- **Random**: DRL agents randomly select an action, e.g.,  $k$  animal IDs to receive their sensed data.
- **Data Mitigation (DM)** [78]: DRL agents randomly select an action and reduce the redundant requests by restricting the communications between sensors and gateways. Each LBS is only allowed to communicate to one gateway at one time.

## 5.7 Numerical Results & Analysis

### 5.7.1 Algorithmic Complexity Analysis

We first analyze the algorithmic complexity of the five DRL schemes described in Section 5.6.4. Table 5.3 shows the asymptotic complexities in Big- $O$  notation for the five schemes

Table 5.3: Asymptotic complexity analysis of the considered schemes

Scheme	Complexity
MADQN/MAPPO	$O(n_e \times t_{train})$
Greedy	$O(n_{action})$
Random	$O(1)$
DM	$O(1)$

discussed in Section 5.6.4. We notice that the cost of MADQN/MAPPO only depends on the training episode  $n_e$  and training time per episode  $t_{train}$ . Greedy needs to enumerate the total action space and thus its complexity depends on the action space size  $n_{action}$ . When the action space is large enough, greedy can incur more cost than MADQN/MAPPO. Table 5.3 shows that the Random and DM are the most efficient algorithm among all while showing the worst performance (to be discussed further in Section 5.7).

### 5.7.2 Comparative Analyses

We conducted all experiments with 100 simulations on randomly generated farm environments based on Section 5.6, where each data point represents the average of the simulation runs. For MADQN, we use 500 as batch size, 0.02 as learning rate. For MAPPO, we use 500 as batch size, 0.08 and 0.008 as learning rates for critic and actor networks, respectively.

Fig. 5.5 compares the six schemes with respect to DRL training episodes. Greedy and random schemes have flat learning curves, since they are non-DRL schemes. We observe that MAPPO-UM and MADQN-UM outperform their corresponding NUM counterparts

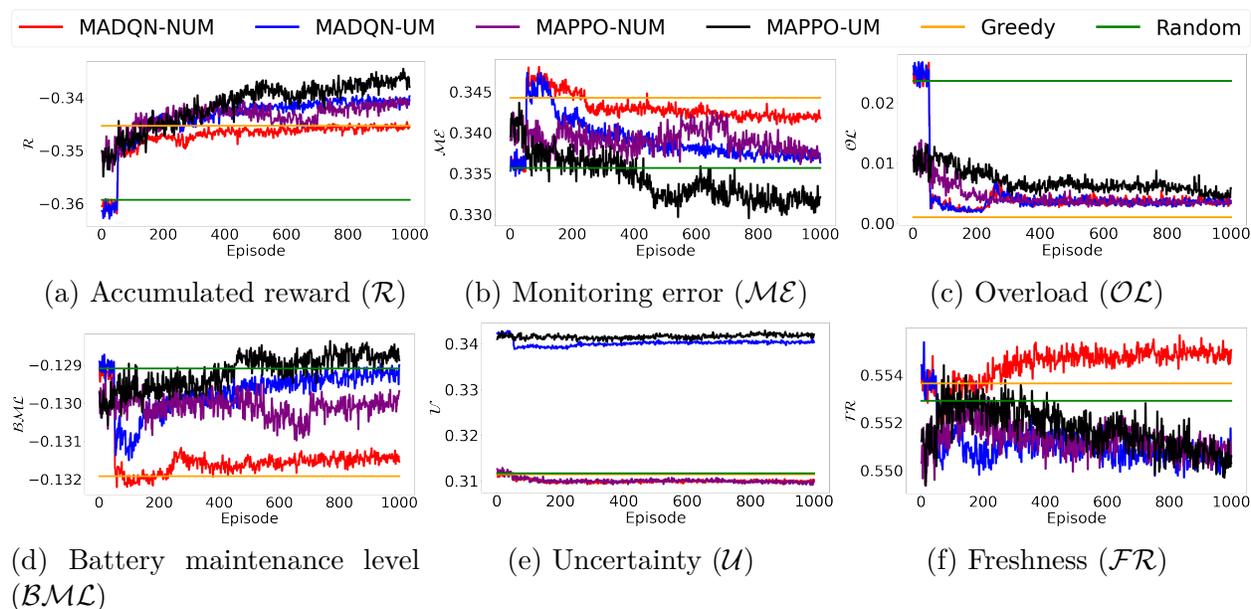


Figure 5.5: Performance of comparing schemes with respect to training episodes.

with respect to the accumulated reward ( $\mathcal{R}$ ), monitoring error ( $\mathcal{ME}$ ), and overload ( $\mathcal{OL}$ ). This is because uncertainty maximization (UM) can update the uncertainty information from time to time, which reflects the sensor network status in a timely manner. As a result, MAPPO-UM and MADQN-UM also have the highest uncertainties ( $\mathcal{U}$ ) in Fig. 5.5 (e). Our proposed uncertainty measures can estimate the update priority by considering the number of history updates for each sensor in a sensor network with acceptable dynamics. The proposed MADQN-based and greedy schemes fail to learn the effective monitoring policies. This is due to the non-stationary environment in the training phase of each agent [147]. Our proposed MAPPO-UM leverages uncertainty information in the stationary decision process and achieves the best performance among all comparing schemes. Note that MAPPO-UM also achieves the best battery efficiency ( $\mathcal{BML}$ ) compared to other schemes in Fig. 5.5 (d). The overall performance order of the considered schemes is: MAPPO-UM  $\geq$  MADQN-UM  $\approx$  MAPPO-NUM  $\geq$  MADQN-NUM  $\geq$  Greedy  $\geq$  Random.

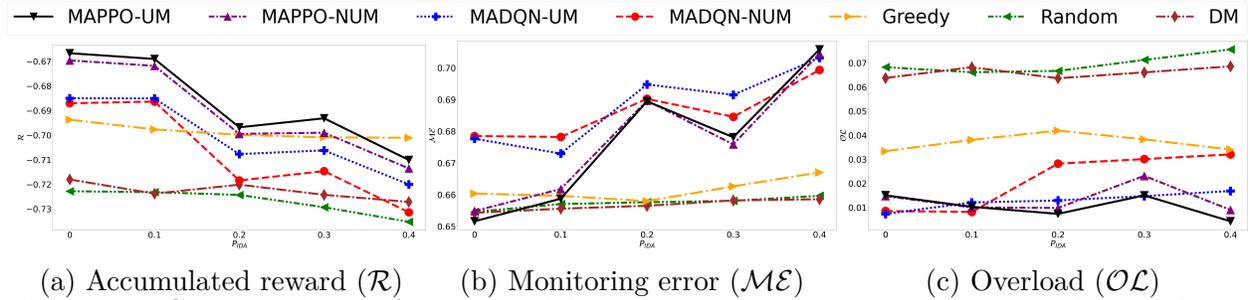
Since our multi-objective function has two conflicting goals, different schemes could have very

Table 5.4: Sensitivity analysis for adversarial attacks

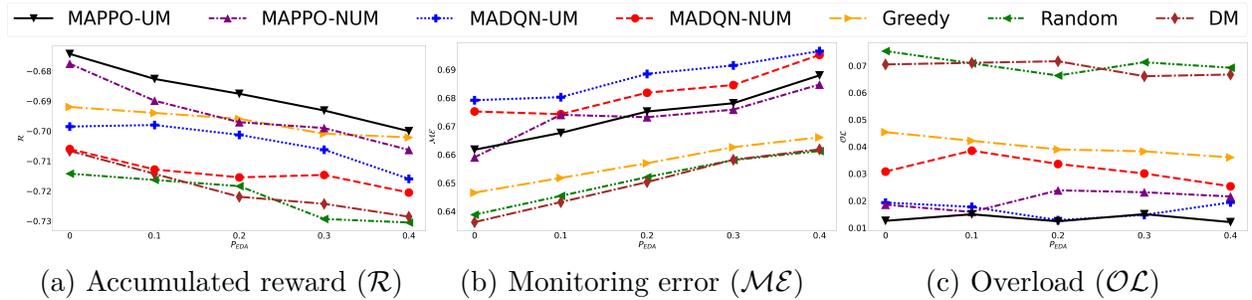
$P_{ADV}$ \ Attack	<i>FGS</i>	<i>MIM</i>	<i>PGD</i>	<i>BIM</i>
0.1	2	2	8	3
0.2	7	9	14	9
0.3	12	8	16	8
0.4	13	13	21	9

different policies. For example, MADQN-NUM and Greedy choose to minimize overload as their primary goals, as shown in Fig. 5.5 (b) and (c). Thus, they have the lowest overloads and highest monitoring error. Since low freshness records only come from LBS, they also have the highest freshness ( $\mathcal{FR}$ ) in Fig. 5.5 (f) due to minimum transmissions from LBS to HBS, as in Fig. 5.5 (d).

### 5.7.3 Sensitivity Analyses

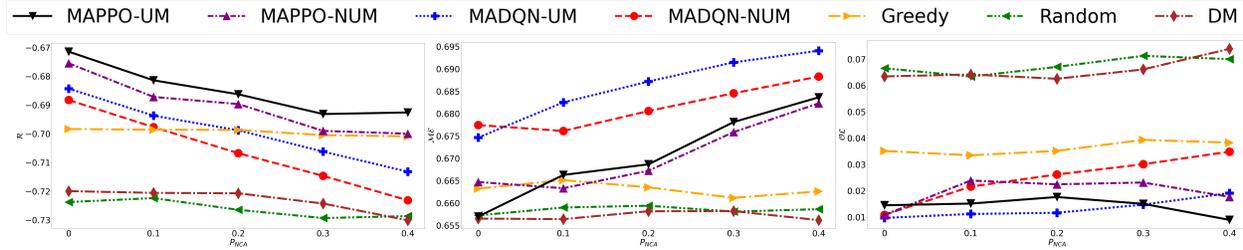


(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
Figure 5.6: Comparative performance with respect to varying the internal attack probability ( $P_{IDA}$ ).

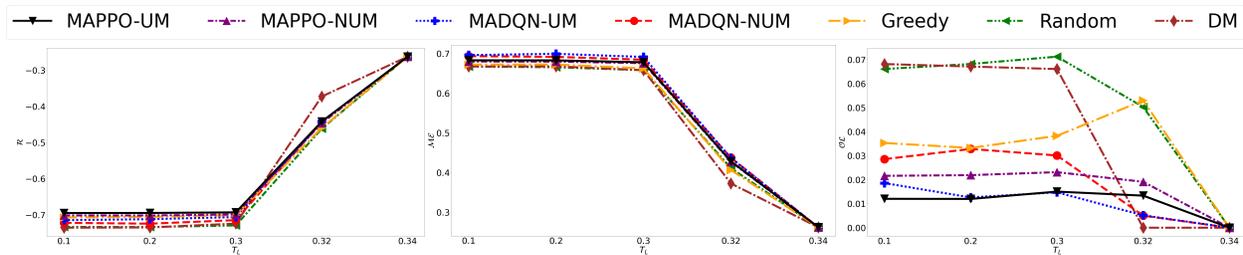


(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
Figure 5.7: Comparative performance with respect to varying the external attack probability ( $P_{EDA}$ ).

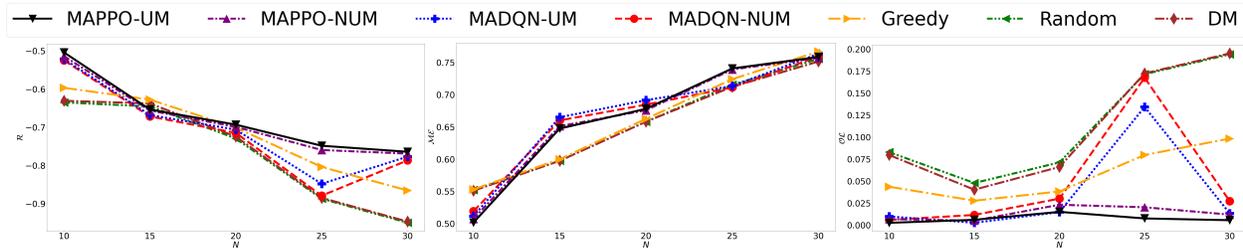
Below we conduct in-depth sensitivity analyses of the two baseline schemes (Greedy and



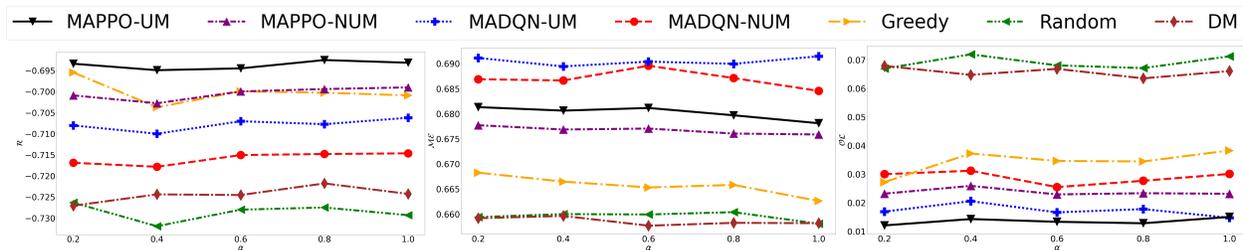
(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
 Figure 5.8: Comparative performance with respect to varying the attacker's non-compliance probability ( $P_{NCA}$ ).



(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
 Figure 5.9: Comparative performance with respect to varying the initial low battery level ( $T_L$ ).



(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
 Figure 5.10: Comparative performance with respect to varying the number of solar sensors ( $N$ ) attached to cows.



(a) Accumulated reward ( $\mathcal{R}$ ) (b) Monitoring error ( $\mathcal{ME}$ ) (c) Overload ( $\mathcal{OL}$ )  
 Figure 5.11: Comparative performance with respect to the different levels of sun exposure ( $\alpha$ ).

Random) and the two uncertainty-aware DRL schemes (i.e., MADQN, MAPPO) with uncertainty maximization (i.e., MADQN-UM, MAPPO-UM) vs. without uncertainty maxi-

mization (i.e., MADQN-NUM, MAPPO-NUM) over a wide range of the common attack probability,  $P_A$  (i.e.,  $P_{NCA}$ ,  $P_{IDA}$ ,  $P_{EDA}$ ), the initial low battery level  $T_L$ , the number of cows (sensors)  $N$ , and the chance for a cow to be exposed to the sun,  $\alpha$ . We also conduct the sensitivity analyses of MAPPO-UM with respect to four different adversarial attacks.

### Effect of Varying the Adversarial Attack Severity

Table 5.4 shows the effect of varying the adversarial attack probability,  $P_{ADV}$ , on the performance of MAPPO-UM in terms of the accumulated reward. For a clear comparison, we show the relative performance decline in % with respect to the default setting where  $P_{ADV} = 0$ . We observe that MAPPO-UM shows resilience towards all four attacks as they only decrease performance a little. Among all considered attacks, PGD is the strongest while BIM is the weakest.

### Effect of Varying the Common Attack Severity

Fig. 5.6–5.8 show the effect of varying the attack probability,  $P_A$ , on the performance of the seven schemes in terms of the three metrics in the network. We observe that increasing  $P_A$  decreases  $\mathcal{R}$  while increasing  $\mathcal{ME}$  and  $\mathcal{OL}$ . When  $P_A$  increases, the monitoring system gets severely compromised, thus the payoff per monitoring update would drop. MAPPO can successfully identify this change in the payoff and achieve better performances than other schemes. The overall performance order with respect to the three metrics is: MAPPO-UM  $\geq$  MAPPO-NUM  $\geq$  Greedy  $\geq$  MADQN-UM  $\geq$  MADQN-NUM  $\geq$  DA  $\geq$  Random.

### Effect of Varying the Initial Battery Levels ( $T_L$ )

Fig. 5.9 shows the effect of varying the initial battery level assigned to LBSs,  $T_L$ , on the performance of the seven schemes in terms of the three metrics in the network. We observe that increasing attack probability ( $T_L$ ) increases the accumulated reward ( $\mathcal{R}$ ) while decreasing the monitoring error rate ( $\mathcal{ME}$ ) and the degree of overload ( $\mathcal{OL}$ ). We also observe that when  $T_L$  increases, all three metrics converge to one point due to the decreased number of LBS. Monitoring policies only apply to LBS and thus negligible differences are observed under high  $T_L$ . Note that DA performs the best when  $T_L = 0.32$  since it can only effectively save energy of LBS with relative high number of LBS and high initial battery level of LBS. This results in more successful reports of LBS and thus less monitoring error. Overall, our proposed MAPPO-based schemes can achieve a low monitoring error rate with the lowest overload.

### Effect of Varying the Node Density ( $N$ )

Fig. 5.10 shows the effect of varying the number of solar sensors,  $N$ , on the performance of the seven schemes in terms of the three metrics in the network. We observe that increasing the number of sensor nodes ( $N$ ) decreases the accumulated reward ( $\mathcal{R}$ ) while introducing higher  $\mathcal{ME}$  and  $\mathcal{OL}$ . When  $N$  increases, there are not enough HBS (high battery level sensors) to transmit data for LBS. Consequently, the update requests from LBS may mostly fail. Our proposed MAPPO-UM scheme achieves the lowest monitoring error rate when  $N$  is low and the lowest overload when  $N$  is high, revealing the tradeoff that a lower monitoring error rate can incur a higher system overload.

### Effect of Varying the Degree of Sun Exposure ( $\alpha$ )

Fig. 5.11 shows the effect of  $\alpha$  (the probability for a cow to be exposed to the sun for energy harvesting) on the performance of the seven schemes. We observe that a higher  $\alpha$  contributes to boosting  $\mathcal{R}$  while reducing  $\mathcal{ME}$ . Moreover,  $\mathcal{OL}$  is not sensitive to varying  $\alpha$  because the energy harvesting speed exceeds the battery consumption speed. We also observe that  $\mathcal{ME}$  is equally reduced for every monitoring policy. Thus, when  $\alpha$  is high, small changes in monitoring policies lead to insensitive  $\mathcal{OL}$  while decreasing  $\mathcal{ME}$ .

Summarizing above, MAPPO-UM performs the best among all schemes, the effect of which is especially pronounced when the system is under high-stress situations, such as high attack severity, low initial battery energy, low node density, and low sun exposure. The reason is that MAPPO-UM applies not only an actor-critic framework (from *Deep Reinforcement Learning* [45]) but also uncertainty maximization (from *Subjective Logic* [80]) to derive stable monitoring policy updates based on new evidence, thus maximizing monitoring quality in terms of  $\mathcal{R}$  and  $\mathcal{ME}$  while reducing energy consumption in terms of  $\mathcal{OL}$ . This allows LoRa gateways to obtain more accurate sensed data from solar sensors having limited and fluctuating energy and to maximize uncertainty-aware monitoring quality.

## 5.8 Key Findings

From this study, we obtain the following **key findings**:

- The system overload does not always increase the monitoring error rate. Our proposed MAPPO-UM scheme can find monitoring policies that can minimize both the monitoring error rate and the system overload.
- The payoffs to monitoring updates are vastly different under different scenarios. This

discrepancy can result in different optimal monitoring policies being identified and applied in different scenarios.

- Our proposed MAPPO-UM scheme is shown to have an acceptable time complexity for which the major complexity comes from the training time and the size of the action space. MAPPO-UM outperformed other counterparts with an 4% reduction in both the monitoring error rate and the system overload.
- Among all schemes considered, MAPPO-UM can best adapt to different scenarios and identify the best monitoring policies for minimizing the monitoring error rate and the system overload.
- Our proposed MAPPO-UM scheme also showed strong robustness, particularly under harsh environments as demonstrated via extensive sensitivity analyses.

# Chapter 6

## Intrusion Response System for Autonomous Driving

This chapter addresses the **Recoverability Task**. This chapter is based on the paper “Intrusion Response System Integrated for Deep Reinforcement Learning-based Autonomous Driving” to be submitted to The Second Symposium on Vehicle Security and Privacy (VehicleSec 2024), Feb. 2024 [177].

### 6.1 Research Goal & Motivation

Security and safety are two primary critical concerns in autonomous driving research and applications [86]. The higher popularity of self-driving autonomous vehicles has introduced more potential risks that may endanger the lives of passengers or other road users due to their potential improper operations. Therefore, countering these risks by ensuring the security and safety of self-driving vehicles has been a must in developing autonomous driving technologies.

Deep reinforcement learning (DRL) has been widely adopted to solve sequential decision problems for autonomous driving. These tasks include lane keeping [137], lane changing [159], ramp merging [158], overtaking [119], and motion planning [84]. There are also end-to-end autonomous driving approaches targeting multiple driving tasks [21, 181]. However, direct deployment of the trained DRL model into autonomous vehicles in real environments may

introduce high risks because of a lack of learning in real-time by the DRL model. The current approaches have focused on additional components added to the DRL model, such as additional safe policy [168], safe constraints [140], and safe control [165]. As for secure autonomous driving, various intrusion detection systems (IDSs) for in-vehicle networks have been developed [79, 139, 164]. Although an intrusion response system (IRS) can improve vehicle protection by filtering false alarms and enhancing defense and security, it has been significantly less explored than well-developed IDS. Unlike IDSs, deploying an IRS often incurs more costs caused by the alert process, risk assessments, and response selection. The in-vehicle networks also have limited computation resources that can hardly meet the requirements of a heavyweight IRS. Specifically, a well-developed IDS can be easily deployed into different vehicle models. However, the current applications of the IRS are often limited by the equipped autonomous driving systems, including the autonomous driving algorithm and the customized response criteria. The system’s incompatibility may hinder the IRS’s adaptability to various vehicle models. Therefore, it is vital to develop a lightweight IRS solution that can be well-integrated into the DRL-based autonomous driving system.

**The aim of this study** is to develop a lightweight IRS for an in-vehicle control area network (CAN) bus system operating with DRL-based autonomous driving. To integrate the proposed IRS into existing autonomous driving systems, we evaluate the utility of a defense action based on autonomous driving performance and utilities of *control command controls* (CCCs), representing acceleration, throttle, and brake commands used to control the vehicle. Then, we compare the evaluated utilities to decide the appropriate response action. We name our proposed IRS approach Control Command Value-based IRS, or namely **CCV**. This work considers independent in-vehicle attacks targeting CAN messages without any collusion. Adversarial attacks targeting the perception components in autonomous driving are out of the scope studied in this work.

## 6.2 Problem Statement

This work aims to build an intrusion response system (IRS) for autonomous driving algorithms that can best identify the best response to the alert messages generated by the IDS. To evaluate the performance of the proposed IRS, given autonomous driving algorithm  $\mathcal{A}$ , IDS  $\mathcal{D}$ , and IRS  $\mathcal{R}$ , we aim to maximize the following objective by optimally deploying the IRS,  $\mathcal{R}$  via:

$$\begin{aligned} \arg \max_{\mathcal{R}} \quad & \mathcal{DS}_{\mathcal{A},\mathcal{D}}(\mathcal{R}) + \mathcal{RC}_{\mathcal{A},\mathcal{D}}(\mathcal{R}) \\ & + \mathcal{IP}_{\mathcal{A},\mathcal{D}}(\mathcal{R}) - \mathcal{DC}_{\mathcal{A},\mathcal{D}}(\mathcal{R}), \end{aligned} \quad (6.1)$$

where three driving performance metrics are driving score ( $\mathcal{DS}$ ), route completion ( $\mathcal{RC}$ ), and infraction penalty ( $\mathcal{IP}$ ) where we aim to maximize  $\mathcal{DS}$ ,  $\mathcal{RC}$ , and  $\mathcal{IP}$ . At the same time, the proposed IRS will aim to minimize defense costs ( $\mathcal{DC}$ ). We detail how to compute each metric in Section 6.6.

## 6.3 Key Contributions

We summarize our **key contributions** as follows:

- We propose a lightweight, integrated IRS solution called **CCV** for DRL-based autonomous driving. The proposed **CCV** strategically selects a defense action based on the control command utilities when a defense (i.e., encryption) is enabled or disabled in the presence or absence of false data injection attacks.
- We introduce a *confidence-score-based alert filter* to refine IDS-generated alerts. The enhanced **CCV** with the refined alerts achieves a higher driving performance and lower

defense cost.

- We design a novel IDS based on a fine-tuned bicycle forward model. The proposed IDS can generate alerts solely based on vehicle information from global positioning system (GPS) signals. This is deployable independently of the data transmission protocols.
- We consider five false data injection (FDI) attacks, modeled as outside attackers modifying the CAN messages in the in-vehicle network. We simulate these attacks with existing benchmarks to evaluate our proposed IRS, *CCV*. We prove our *CCV* demonstrates higher resilience against these attacks than the considered existing counterparts.
- Via extensive experiments, we analyze the performance of *CCV* under two DRL-based autonomous driving algorithms, *Rails* [21] and *Roach* [181]. Our experiment proves that *CCV* outperforms existing counterparts in driving performance and defense cost.

## 6.4 System Model

This section discusses the network, node, and attack models.

### 6.4.1 Network Model

We consider an in-vehicle network with a Controller Area Network (CAN bus) as the vehicle bus standard. The CAN bus is connected to mobile devices through multiple mobile applications. This includes shared Global Positioning System (GPS) navigation systems through wireless Bluetooth connections and wired USB connections. In this network environment, the attacker can infiltrate the CAN bus through these connections via mobile applications.

We simulate the vehicle and in-vehicle network by the CARLA simulator [47] and *python-*

*can package* [131]. CARLA was developed from the ground up to support the development, training, and validation of autonomous driving systems (ADSs). CARLA is shipped with an autonomous driving leaderboard [16] to evaluate autonomous driving algorithms (ADAs). In this work, we adopt metrics from this benchmark (see Section 6.6) to evaluate the performance of the proposed ADS with or without the proposed IRS. The *python-can package* provides a suite of utilities for sending and receiving messages on a can bus. We use this package to simulate in-vehicle communications.

Fig. 6.1 shows an overview of the network model. Attackers use mobile phones as entry points to hack into the in-vehicle network through wireless and wired ports, e.g., WiFi, Bluetooth, or USB. Then they breach into the central gateway ECU to navigate attacks to vehicle control ECUs. The IDS can request vehicle information from the central gateway ECU and generate alerts for the IRS. The IRS outputs the defense response based on the alert received. Finally, the central gateway ECU executes the defense actions for all vehicle control ECUs.

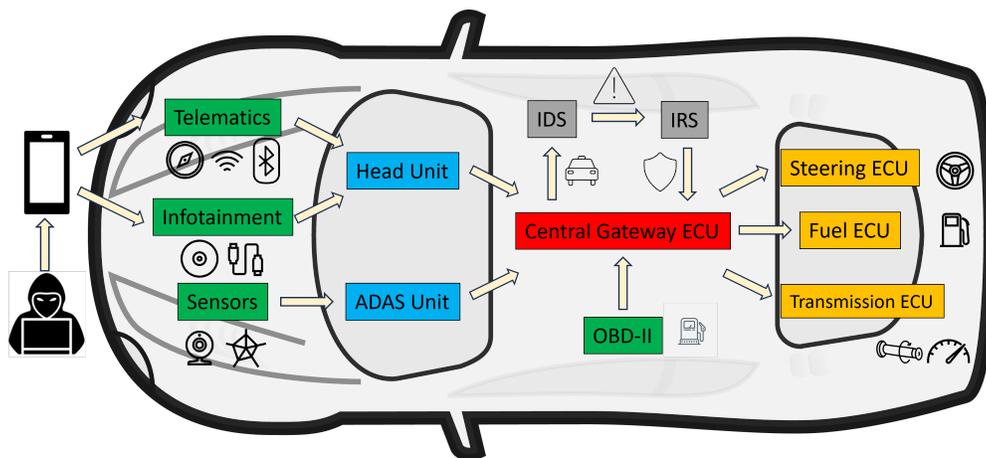


Figure 6.1: An overview of the network model.

### 6.4.2 Node Model

Each node in the CAN bus network consists of a central processing unit (CPU), a CAN controller, and a transceiver. Sensors attached to the CPU generate a message for a complete data transmission. Then, the CPU processes the data and transmits it to the CAN controller. Next, the transceiver converts the data from the CAN controller and sends it to the CAN bus network. To receive a message from the CAN bus, the transceiver converts the data and sends it to the CAN controller. The CAN controller records the data as a complete message and sends it to the CPU. Lastly, the CPU decodes the message.

All CAN bus messages follow the CAN frames. There are two lengths of frames: standard (CAN 2.0A) and extended (CAN 2.0B). The standard frame has an 11-bit identifier, and the extended one has a 29-bit identifier. We consider the data field under the standard frame. The data field consists of 0-8 bytes. We use 8 bytes to record an entire message. Specifically, we only consider control command messages. Fig. 6.2 demonstrates each control command and its corresponding location in the data field.

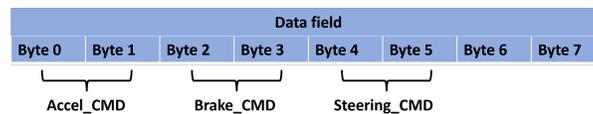


Figure 6.2: Control commands in a CAN message.

### 6.4.3 Attack Model

We consider the following independent false data injection attacks based on a car-hacking dataset [14]:

- *Denial-of-Service (DoS) attack* injects two consecutive bytes of zeros into random positions in the data field.

- *Fuzzy attack* injects totally random values into every position in the data field.
- *Spoofing gear attack* injects random values into positions related to gear information in the data field.
- *Spoofing steering attack* injects random values into positions related to steering information in the data field.
- *Spoofing RPM attack* injects random values into positions related to RPM information in the data field.

For simplicity for the experimental settings used in in-depth sensitivity analyses in Section 6.7, we will consider the same probability for each attack to occur. The setting is detailed in Section 6.6. Attack-specific defense actions need a real in-vehicle network implemented to simplify the defense scenarios with simulators. Thus, we assume false data injection (FDI) attacks can be directly blocked by an extra encrypted communication channel in the in-vehicle network. The encrypted communication channel is assumed to incur a cost  $C$  per usage. We describe the details of the proposed IRS (i.e., CCV), including this defense, in Section 6.5.2. The security protection methods for a CAN bus mainly fall into two categories: encryption-based protection and IDS-based protection. Unlike IDS-based protection, encryption can be applied as a general defense mechanism against various attacks [55]. Thus, we adopt encryption as the defense in our work. Similar to [3], existing authenticated encryption schemes like AEGIS [163] (i.e., an authenticated encryption algorithm) can be used to provide secure encryption protocols. For the key management, we assumed that each ECU is preloaded with two long-term symmetric keys during the manufacturing phase. Each ECU shares these keys with the central gateway ECU only. Further, these keys are used to generate individual session keys using existing protocols like AKEP2 [12]. Finally, the group session key is generated by taking a hash of all individual keys on the same ECU

channel.

## 6.5 Proposed Approach: CCV-based IRS

In this section, we provide a detailed description of our proposed control command value-based intrusion response system for DRL-based autonomous driving systems. Our proposed CCV-based IRS runs by processing the attacks detected by the IDS. Therefore, this section details the proposed CCV-based IRS with the designs of the IDS.

### 6.5.1 Intrusion Detection System for In-Vehicle Networks

We build the IDS based on a forward model and a classifier. The IDS only relies on vehicle information. Thus, it differs from traditional approaches dealing with in-vehicle network traffic data. Since our approach does not depend on the network traffic data, it can be easily generalized to vehicles with different data transmission protocols.

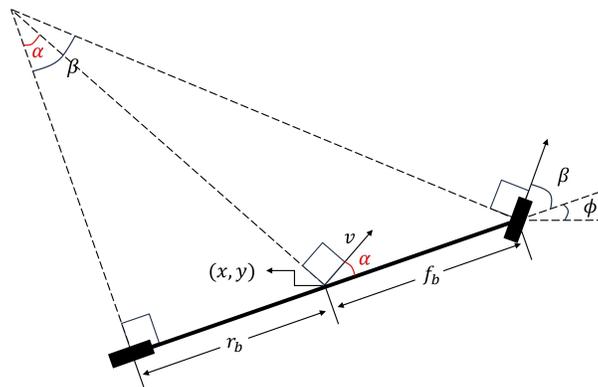


Figure 6.3: The kinematic bicycle model.

## Forward Model

The forward model is based on the kinematic bicycle model [128], denoted by  $\mathcal{F}$ . We denote the vehicle location as two-dimensional coordinates  $(x, y)$ . The vehicle speed, acceleration, and orientation are represented by  $v$ ,  $a$ , and  $\phi$ . The vehicle wheelbases are expressed by  $r_b$  and  $f_b$ , and  $s$ ,  $t$ , and  $b$  are steer, throttle, and brake, respectively. Where  $(\dot{\cdot})$  denotes the derivative of a vector, we formulate the derivative of each notation by:

$$\begin{aligned} \dot{x} &= v \cos(\phi + \alpha), \quad \dot{y} = v \sin(\phi + \alpha), \quad \dot{v} = a, \\ r_b \dot{\phi} &= v \sin(\alpha), \quad (f_b + r_b) \tan(\alpha) = r_b \tan(\beta) \\ \beta &= \omega_s s, \quad a = \begin{cases} \omega_b, & \text{if } b = 1 \\ \omega_t t, & \text{otherwise.} \end{cases} \end{aligned} \quad (6.2)$$

Here  $\beta$  is the steering angle, and  $\alpha$  is the slip angle. We assume the vehicle control is applied at the center of gravity. Then,  $\alpha$  gives the derivation from the current orientation at the center of gravity due to steering.  $\beta$  gives the derivation from the current orientation at the front wheel due to steering.  $r_b / \sin(\alpha)$  gives the radius of rotation due to steering. We model  $\beta$  as a function of  $s$ , and  $a$  as a function of  $b$  and  $t$ . The forward model  $\mathcal{F}$  is parameterized by  $\omega_s$ ,  $\omega_b$ , and  $\omega_t$ . Fig. 6.3 shows an overview of the underlying kinematic bicycle model of  $\mathcal{F}$ . Forward model  $\mathcal{F}$  is represented by:

$$\begin{aligned} \mathcal{F} &= \mathcal{F}_{\omega_s, \omega_b, \omega_t}, \\ x_{t+1}, y_{t+1}, \phi_{t+1}, v_{t+1} &= \mathcal{F}(x_t, y_t, \phi_t, v_t, a_t), \end{aligned} \quad (6.3)$$

where  $a_t = (s_t, t_t, b_t)$  and  $s_t$ ,  $t_t$ , and  $b_t$  are steer, throttle, and brake at time  $t$ , respectively.

We train  $\mathcal{F}$  with stochastic gradient descent with L1 loss:

$$L = \sum_t^T |\cos(\phi_t) - \cos(\hat{\phi}_t)| + |\sin(\phi_t) - \sin(\hat{\phi}_t)| \quad (6.4)$$

$$+ \sum_t^T |x_t - \hat{x}_t| + |y_t - \hat{y}_t|,$$

where  $(\hat{\cdot})$  means the ground truth.

### Classifier

We use the outputs of the forward model  $\mathcal{F}$  to train a classifier  $\mathcal{C}$ . Specifically, we use  $\mathcal{F}$  at step  $t - 1$  to predict vehicle information (i.e.,  $x_t, y_t, \theta_t$ , and  $v_t$ ) and record ground truth vehicle information (i.e.,  $\hat{x}_t, \hat{y}_t, \hat{\theta}_t$ , and  $\hat{v}_t$ ) at step  $t$ . We use XGBoost [26] to train a classifier  $\mathcal{C}$  such that  $l_t = \mathcal{C}(x_t, y_t, \theta_t, v_t, \hat{x}_t, \hat{y}_t, \hat{\theta}_t, \hat{v}_t)$ . We chose it among conventional ML classifiers as it has shown the best performance. Testing various ML classifiers is beyond the scope of our work. We consider a binary classifier, thus  $l_t \in \{0, 1\}$ , and our IDS generates alert  $l_t$  without enabled encrypted communication channels.

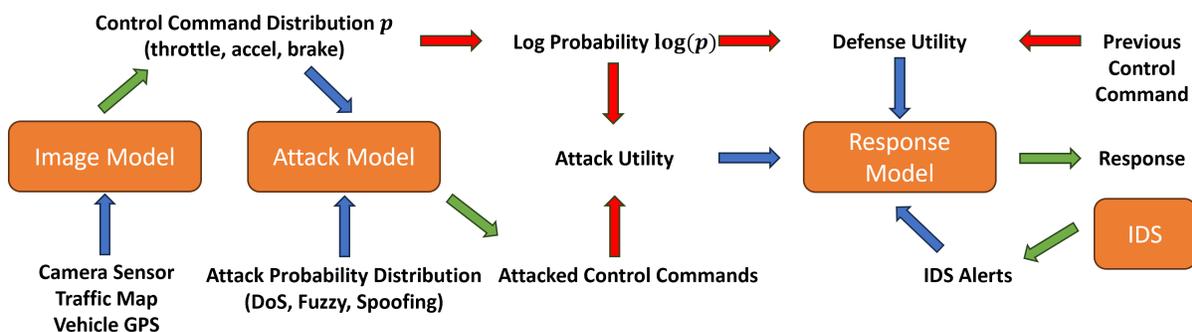


Figure 6.4: An overview of the proposed control command value (CCV)-based IRS.

### 6.5.2 Control Command Value (CCV)-based IRS

We build the IRS based on existing DRL-based autonomous driving algorithms. We call our proposed IRS a *Control Command Value (CCV)-based IRS*. Fig. 6.4 shows an overview of the proposed CCV-based IRS. Specifically, we consider both discrete and continuous action spaces to evaluate a defense action and generate a response correspondingly.

#### Defense Actions

We consider binary defense actions, using either encryption or not, i.e.,  $e \in \{0, 1\}$ .  $e$  represents whether or not the encrypted communication channel is used, i.e.,  $e = 1$  means the IRS activates the channel;  $e = 0$  otherwise. We assume the encrypted communication channel has a constant duration  $d$  once activated. Furthermore, activating an encrypted communication channel incurs a system latency that forces using the last control command that was not attacked.

#### Discrete Action Space

The discrete action space  $\mathcal{DA}$  consists of finite tuples of control commands  $(s, t, b)$ , where  $s$ ,  $t$ , and  $b$  are steer, throttle, and brake, respectively. To evaluate the utility of a defense action at time  $t$ , we calculate  $u_{1,t}$  and  $u_{0,t}$  as the expected utilities of defense actions  $e_t = 1$  and  $e_t = 0$ , respectively, based on the utilities of control commands under each defense action (i.e., enabled with  $e_t = 1$  and disabled with  $e_t = 0$ ). The DRL algorithms output a probability distribution  $\mathbf{p}_t$  over  $\mathcal{DA}$ . Since  $\mathbf{p}_t$  is calculated by the *softmax* function, we use the reverse operation of *softmax* to get  $\log(\mathbf{p}_t)$  from  $\mathbf{p}_t$  as the utility distribution over  $\mathcal{DA}$ . This means that for a given control command  $j$ , higher  $\mathbf{p}_t(j)$  means higher  $\log(\mathbf{p}_t(j))$ , producing higher utility in control command,  $j$ .

The considered FDI attacks (see Section 6.4.3) randomly modify the control commands to other variant commands regularly. We can construct a set  $T_t$  to enumerate those variants. Without loss of generality, we assume  $\mathcal{DA}$  has size  $n = n_s \times n_t \times n_b$  such that  $n_s$ ,  $n_t$ , and  $n_b$  are the numbers of steer, throttle, and brake levels, respectively. Then, the numbers of variants based on  $\mathcal{DA}$  for DoS, Fuzzy, Spoofing gear, Spoofing steering, and Spoofing RPM attacks are 4,  $n$ ,  $n_b$ ,  $n_s$ , and  $n_t$ , respectively. Therefore, the size of  $T_t$  equals  $4 + n + n_b + n_s + n_t$ . This allows us to evaluate the utility  $u_j$  of a variant  $j$  by a linear interpolation based on  $\log(\mathbf{p}_t)$ . Since the attack probability distribution  $\mathbf{p}_{a,t}$  at time  $t$  is given, we can derive the expected utility of an attacked control command by:

$$E_t(u_{atk}) = \sum_{at \in FDI} \sum_{j \in T_{at,t}} \frac{u_j \mathbf{p}_{a,t}(at)}{|T_{at,t}| \sum_{at \in FDI} \mathbf{p}_{a,t}(at)}, \quad (6.5)$$

where  $at$  is an attack type in the set of considered FDI attacks,  $FDI$ .  $T_{at,t}$  is the subset of  $T_t$  where  $T_t$  is the set of the variants of attacked control commands.  $T_{at,t}$  includes only different variants of control commands attacked by attack type  $at$ , and  $|\cdot|$  is the size of  $\cdot$ . To simplify the following formulas, we denote attack probability  $P_{a,t} = \sum_{at \in FDI} \mathbf{p}_{a,t}(at)$ . Accordingly, the control command utility at time  $t$  when defense is disabled with  $e_t = 0$ , denoted by  $u_{0,t}$ , is given by:

$$\begin{aligned} u_{0,t} &= \underbrace{E_t(u_{atk})}_{\text{current expected utility}} + \underbrace{((d-1)(1-P_{a,t}))u_t^*}_{\text{expected utility under no attack for the next } (d-1) \text{ duration}} \\ &\quad + \underbrace{((d-1)P_{a,t})E(u_{atk})}_{\text{expected utility under attack for the next } (d-1) \text{ duration}} \\ &= u_t^*(d-1)(1-P_{a,t}) + E_t(u_{atk})(1+(d-1)P_{a,t}), \end{aligned} \quad (6.6)$$

where  $u_t^*$  is the utility of the original control command output of the DRL algorithm with respect to  $\log(\mathbf{p}_t)$ , and  $d$  is the pre-defined duration of an encrypted communication channel

(e.g.,  $d = 5$ ). Similarly, we formulate the control command utility at time  $t$ ,  $u_{1,t}$ , when defense is enabled with  $e_t = 1$  by:

$$\begin{aligned}
 u_{1,t} = & \underbrace{u'_t}_{\text{current expected utility}} + \underbrace{E_t(u_{atk})((d-1)(1-p_d)P_{a,t})}_{\text{expected utility under attack for the next } (d-1) \text{ duration}} \\
 & + \underbrace{u_t^*(1 - (d-1)(1-p_d)P_{a,t})}_{\text{expected utility under no attack for the next } (d-1) \text{ duration}}, \tag{6.7}
 \end{aligned}$$

where  $u'_t$  is the utility of the previous output of the DRL algorithm with respect to  $\log(\mathbf{p}_t)$ , and is updated per step when  $l_t = 0$  and  $p_d$  is the successful defense probability. We assume the encrypted communication channel is highly reliable such that  $p_d \approx 1$ . The simplified  $u_{0,t}$  is:

$$u_{1,t} = u'_t + u_t^*(1 - (d-1)). \tag{6.8}$$

The proposed CCV-based IRS will return  $e_t = 1$  when  $u_{1,t} > u_{0,t}$  and  $e_t = 0$  otherwise.

## Continuous Action Space

The continuous action space  $\mathcal{CA}$  consists of infinite tuples of control commands  $(s, t, b)$ , where  $s \in \{-1, 1\}$ ,  $t \in \{0, 1\}$ ,  $b \in \{0, 1\}$ . We normalize the scales of these actions for fair evaluation. Similar to discrete actions, we use the control command utilities,  $\mathbf{p}_t$  and  $\log(\mathbf{p}_t)$ , provided by the DRL algorithms to calculate the utility when a defense is enabled or disabled, denoted by  $u_{0,t}$  and  $u_{1,t}$  (see Eqs. (6.6) and (6.8)), respectively. We directly evaluate a control command because  $\log(\mathbf{p}_t)$  is continuous.

### Confidence Score-based Alert Filtering

For each detection at time  $t$ , IDS outputs a prediction class label  $l_t$  and a corresponding probability distribution  $\mathbf{p}_t$ , representing the distribution of confidence scores in each attack scenario,  $a_i$  (i.e., attack  $a_i = 1$  or no attack  $a_i = 0$  in this study). To minimize the process of false alarms, which can waste defense costs, our IRS will consider threshold  $\lambda$  to filter out false alarms generated by the IDS based on the confidence score of each attack scenario,  $\mathbf{p}_t(a_i)$ . We define an alert confidence level  $\tau(a_i)$  based on the  $\mathbf{p}_t(a_i)$  by:

$$\tau(a_i) = \max[(\mathbf{p}_t(a_i) - \lambda)/\lambda, 0]. \quad (6.9)$$

The IDS would output  $a_i = 1$  with probability  $\min[\tau(a_i), 1]$  in the proposed IRS.  $\tau$  allows for stochastic filtering and can significantly reduce unnecessary defense costs introduced by processing false alarms.

## 6.6 Experimental Setup

This section describes the DRL algorithms, datasets, comparing schemes, and performance metrics used for the experiments conducted in this work.

**DRL-based Autonomous Driving Algorithms.** We consider *Rails* [21] and *Roach* [181], which are pre-trained for autonomous driving algorithms in the absence of attacks. We leverage them to build and evaluate the proposed IRS with the proposed pre-trained IDS.

**Datasets.** We adopt the *CARLA NoCrash-dense benchmark* [48] as the experiment environment. The datasets include the route scenarios under various weather conditions. Route scenarios include freeways, urban areas, residential districts, and rural settings. Weather

conditions include daylight scenes, sunset, rain, fog, and night. All experimental results in this work are the average 50 scenarios in the benchmark. This includes Table 6.1, Fig. 6.5, and Fig. 6.6.

**Comparing Schemes.** We evaluate the following schemes:

- **CCV-IRS-CSF** is the autonomous driving algorithm (ADA) with control command value-based intrusion responses (CCV-IRS) and confidence score-based alert filtering (CSF).
- **DS-Ranking** [141] is the ADA with optimal intrusion responses based on dynamic-speed (DS) rankings at each step.
- **Random** is the ADA with randomly selected intrusion responses based on IDS alarm probabilities.
- **No-Defense** is the ADA without intrusion responses (i.e., no defense).
- **Defense** is the ADA with “always defense response” to a detected intrusion.

**Metrics.** We consider the following metrics for validation:

- **Driving score (DS)** [16] is the main metric of the leader-board and is measured by the product of the route completion ( $\mathcal{RC}$ ) and infractions penalty ( $\mathcal{RC}$ ) by:

$$DS = \mathcal{RC} \times IP. \quad (6.10)$$

- **Route completion ( $\mathcal{RC}$ )** [16] refers to the percentage of the route distance completed by a DRL agent:

$$\mathcal{RC} = \frac{L_c}{L}, \quad (6.11)$$

where  $L_c$  and  $L$  are the completed and the total route lengths, respectively.

- **Infraction penalty ( $\mathcal{IP}$ )** [16] aggregates all infractions triggered by the DRL agent as a geometric series where the leader-board tracks several types of infractions. The agent starts with an ideal 1.0 base score, which is reduced for each type to which an infraction is committed:

$$\mathcal{IP} = \prod_{j \in I} (p_j)^{\#\text{infractions}_j}, \quad (6.12)$$

where  $I$  is the set of infractions, i.e.,  $I = \{\text{running a red light, running a stop sign, collisions with pedestrians, collisions with other vehicles, collisions with static elements, scenario timeout, failure to maintain minimum speed, failure to yield to emergency vehicle}\}$ ,  $p_j$  is a predefined value ranged in  $(0, 1)$  for infraction penalty type  $j$ , and  $\#\text{infractions}_j$  is the number of type  $j$  infractions.

- **Defense cost ( $\mathcal{DC}$ )** is the aggregated times to take defenses with a constant defense cost per usage and given by:

$$\mathcal{DC} = \sum_t e_t, \quad (6.13)$$

where  $e_t$  is a binary value indicating whether IRS activates the encrypted communication channel or not at time  $t$ .

## 6.7 Numerical Results & Analysis

Below, we conduct in-depth sensitivity analyses of the five schemes in Section 6.6 over a wide range of the attack probability,  $P_a$  where  $P_a$  is defined as the sum of the probabilities of attacks considered in Section 6.4.3 for simplicity. That is, we define  $P_a = P_{a,t} = \sum_{at \in FDI} \mathbf{p}_{a,t}(at)$  since  $\mathbf{p}_{a,t}$  is fixed in our work. We also conduct sensitivity analyses with respect to two different autonomous driving algorithms, *Rails* and *Roach*. Further, we in-

investigate the effect of the confidence-score-based filtering (CSF) mechanism proposed in the CCV-based IRS.

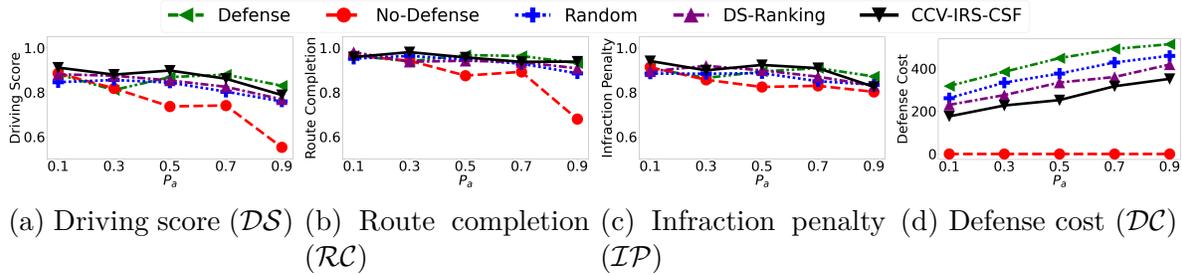


Figure 6.5: Comparative performance of the *Rails* with respect to varying the attack probability ( $P_a$ )

**Effect of the attack severity with Rails.** Fig. 6.5 shows the effect of varying the attack probability,  $P_a$ , on the performance of the *Rails* under five schemes in terms of the four metrics in Section 6.6. Our results show that increasing  $P_a$  decreases  $\mathcal{DS}$ ,  $\mathcal{RC}$ , and  $\mathcal{IP}$  while increasing  $\mathcal{DC}$ . Every defense action comes along with the control command latency. This introduces the inherent trade-off between performance and security. The scheme *Defense* (i.e., always enable defense, which is an encryption communication) reflects this trade-off, achieving the best under the highest attack probability but performing worse than other schemes when the attack probability drops. As in Fig. 6.5, *Defense* has the highest defense cost, and *No-Defense* has the lowest defense cost. Our proposed CCV-IRS-CSF scheme has the second-lowest defense cost with the best driving score overall.

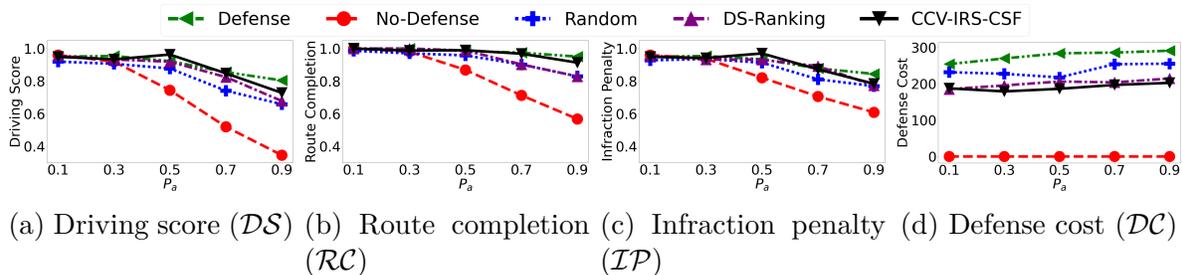


Figure 6.6: Comparative performance of the *Roach* with respect to varying the attack probability ( $P_a$ )

**Effect of the attack severity with Roach.** Fig. 6.6 shows the effect of varying the attack probability,  $P_a$ , on the performance of the Roach under five schemes. Similar to Fig. 6.5,  $DS$ ,  $\mathcal{RC}$ , and  $\mathcal{IP}$  decreases while  $DC$  increases as  $P_a$  becomes higher. In terms of driving scores ( $DS$ ), the margin between *Defense* and *No-Defense* becomes 38% larger under high  $P_a = 0.9$ , compared to Fig. 6.5. This implies that defense actions are more effective for Roach in the presence of attacks. *No-Defense* also performs 8% better with low  $P_a = 0.1$  and performs 37% worse with high  $P_a = 0.9$  compared to the ones observed in Fig. 6.5. This means Roach performs better than Rails in the absence of attacks but less stable than Rails in the presence of attacks. The overall defense costs for Roach are lower than Rails, especially under high  $P_a$ . This means IDS performs worse for Roach than it performs for Rails. Thus, *Defense*, *Random*, *DS-Ranking*, and *CCV-IRS-CSF* perform worse in high  $P_a$  cases compared to their performance under the Rails. The observed results are because Roach, a model-free approach, explores more solution spaces than Rails to determine the optimal policy in the absence of attacks. Thus, the exploration prioritizes the actions with high rewards. However, without enough training under attack scenarios, the learned policy cannot maintain its high performance, and IDS can hardly distinguish the low-performance policy from the attacked policy. Therefore, Roach becomes less stable than the mode-based approach Rails. On the other hand, Rails, which is a model-based approach, considers all possible traffic scenarios equally, within a deviation from the desired driving behaviors. This allows Rails to have a more balanced exploration policy than Roach. This policy may hurt the performance of Rails in the absence of attacks, but it could also help Rails maintain its performance in the presence of attacks. In conclusion, our proposed CCV-IRS-CSF scheme has the second-lowest defense cost while showing the best driving score overall.

**Effect of the confidence score-based filtering (CSF) in the CCV-based IRS.** To demonstrate the effectiveness of the proposed CSF in the proposed IRS, we can compare

Table 6.1: CCV performance W/O confidence score-based filtering (CSF)

Metrics	CCV with CSF	CCV without CSF
Driving score	0.90	0.82
Route completion	0.92	0.87
Infraction penalty	0.96	0.92
Defense cost	252.16	418.64

the performance of our IRS and IDS with or without CSF. Table 6.1 shows the performance of our proposed CCV-based IRS with or without CSF. We evaluated the results where Rails is used as a DRL algorithm under attack probability  $P_a = 0.5$ . As shown in Table 6.1, *CCV with CSF* means the metric value with CSF, and *CCV without CSF* means the metric value without CSF. This result shows that using CSF significantly saves the defense cost and improves driving performance.

## 6.8 Key Findings

From this study, we obtain the following **key findings**:

- Our proposed CCV-based IRS shows strong resilience against attacks by achieving the best driving performance with sufficiently low defense cost (i.e., the second best among all).
- The confidence score-based alert filtering (CSF) further restricts the alert generation and significantly improves the performance of our proposed CCV-based IRS with lower defense costs.
- Our proposed CCV-based IRS offers a generalizable approach by adapting to different state-of-the-art DRL-based autonomous driving algorithms.

# Chapter 7

## Conclusion

### 7.1 Completed Work

During my Ph.D. study, I have the following papers published, accepted, or submitted:

- **Qisheng Zhang**, Abdullah Zubair Mohammed, Zelin Wan, Jin-Hee Cho, and Terrence J. Moore, “Diversity-by-Design for Dependable and Secure Cyber-Physical Systems: A Survey,” in *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 706-728, March 2022.
- **Qisheng Zhang**, Jin-Hee Cho, Terrence J. Moore, and Ing-Ray Chen, “Vulnerability-Aware Resilient Networks: Software Diversity-Based Network Adaptation,” in *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3154-3169, Sept. 2021.
- **Qisheng Zhang**, Jin-Hee Cho, and Terrence J. Moore, “Network Resilience Under Epidemic Attacks: Deep Reinforcement Learning Network Topology Adaptations,” in *2021 IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain, 2021, pp. 1-7.
- **Qisheng Zhang**, Jin-Hee Cho, Terrence J. Moore, and Frederica Free Nelson, “DREVAN: Deep Reinforcement Learning-based Vulnerability-Aware Network Adaptations for Re-

silient Networks,” in *2021 IEEE Conference on Communications and Network Security (CNS)*, Tempe, AZ, USA, 2021, pp. 137-145.

- **Qisheng Zhang**, Yash Mahajan, Ing-Ray Chen, Dong Sam Ha, and Jin-Hee Cho, “An Attack-Resilient and Energy-Adaptive Monitoring System for Smart Farms,” in *2022 IEEE Global Communications Conference (GLOBECOM)*, Rio de Janeiro, Brazil, 2022, pp. 2776-2781.
- **Qisheng Zhang**, Zhen Guo, Audun Jøsang, Lance Kaplan, Feng Chen, Dong Hyun Jeong, Jin-Hee Cho, “PPO-UE: Proximal Policy Optimization via Uncertainty-Aware Exploration”, in *The 1st AAAI Workshop on Uncertainty Reasoning and Quantification in Decision Making*, 2023.
- **Qisheng Zhang**, Jin-Hee Cho, Terrence J. Moore, Dongseong Kim, Hyuk Lim, and Frederica F. Nelson, “EVADE: Efficient Moving Target Defense for Autonomous Network Topology Shuffling Using Deep Reinforcement Learning”, *The 21st International Conference on Applied Cryptography and Network Security (ACNS)*, Jun. 2023.
- **Qisheng Zhang**, Dian Chen, Y. Mahajan, Ing-Ray Chen, Dong S. Ha, and Jin-Hee Cho, “Attack-Resistant, Energy-Adaptive Monitoring for Smart Farms: Uncertainty-Aware Deep Reinforcement Learning Approach”, *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14254-14268, Aug. 2023.
- Zhen Guo, Qi Zhang, Xinwei An, **Qisheng Zhang**, Audun Jøsang, Lance Kaplan, Feng Chen, Dong Hyun Jeong, Jin-Hee Cho, “Uncertainty-Aware Reward-based Deep Reinforcement Learning for Intent Analysis of Social Media Information”, in *The 1st AAAI Workshop on Uncertainty Reasoning and Quantification in Decision Making*, 2023.

- Zhen Guo, Zelin Wan, **Qisheng Zhang**, Xujiang Zhao, Feng Chen, Jin-Hee Cho, Qi Zhang, Lance Kaplan, Dong Hyun Jeong, and Audun Jøsang, “A Survey on Uncertainty Reasoning and Quantification for Decision Making: Belief Theory Meets Deep Learning”, *Information Fusion*, Volume 101, 2024, 101987, Sep. 2023.
- Lei Zhang, **Qisheng Zhang**, Zhiqian Chen, Yanshen Sun, Chang-Tien Lu, Liang Zhao, “Infinitely Deep Attributed Graph Transformation Learning”, *The 23rd IEEE International Conference on Data Mining (ICDM 2023)*, Dec. 2023.
- Zhen Guo, Qi Zhang, **Qisheng Zhang**, Lance Kaplan, Audun Jøsang, Feng Chen, Dongseong Jeong, and Jin-Hee Cho, “Detecting Intents of Fake News Using Uncertainty-Aware Deep Reinforcement Learning,” *The International Conference on Web Services (ICWS)*, Jul. 2023.
- **Qisheng Zhang**, Terrence J. Moore, Dongseong Kim, Hyuk Lim, Seunghyun Yoon, Frederica F. Nelson, and Jin-Hee Cho, “Intrusion Response System Integrated for Deep Reinforcement Learning-based Autonomous Driving”, submitted to *The Second Symposium on Vehicle Security and Privacy (VehicleSec 2024)*, Dec. 2023.
- Zhen Guo, **Qisheng Zhang**, Qi Zhang, Lance Kaplan, Audun Jøsang, Feng Chen, Dongseong Jeong, and Jin-Hee Cho, “mudRIA: Multidimensional Uncertainty-Aware Deep Reinforcement Learning-based Intent Analysis of Fake News Spreaders”, submitted to *The 38th Annual AAAI Conference on Artificial Intelligence (AAAI)*, Aug. 2023.
- Dian Chen, **Qisheng Zhang**, Ing-Ray Chen, Dong Ha, and Jin-Hee Cho, “Energy-Adaptive, Robust Monitoring for Solar Sensor-based Smart Farms Under Adversarial Attacks,” submitted to *IEEE Transactions on Sustainable Computing*, Apr. 2023.

- Dian Chen, **Qisheng Zhang**, Ing-Ray Chen, Dong Ha, and Jin-Hee Cho, “Robust Learning-based Monitoring Protocol for Sustainable Smart Farms,” submitted to *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 2024)*, Sep. 2023.

## 7.2 Future Work

My work has the following research directions to be explored in the future:

**Adaptability Task:** We assume a distributed network environment where multiple independent networks are managed together. For each independent network, an adaptation budget is given as a variant. The total sum of budgets is given as a constant. Then we can build a hierarchical structure in the following way. Multiple DRL agents can be deployed into those independent networks, and each of them is trained to find a corresponding optimal adaptation strategy. The overall adaptation strategy for the distributed network can be determined by integrating all those strategies for independent networks. This design can allow the proposed DRL-based approach to be easily generalizable to multi-agent cases for better applicability in distributed network environments.

**Fault-Tolerance Task:** The current approach models the sun’s movement, the weather conditions, and the cow’s behavior. In a real smart farm environment, these factors may be more complex and dynamic. We also measure the system latency from the computation overhead. In a real smart farm system, the latency can be directly measured as a time delay. By incorporating these modifications, we can evaluate the performance of our proposed monitoring system in a real system. This could bring more

insights for potential improvements. The monitoring system can be implemented in a real smart farm for further case studies.

**Recoverability Task:** The current work in Chapter 6 uses a CARLA simulator to model the communications between the ego vehicle and the traffic environment. The CARLA server takes all the computation tasks. This is not a standard desirable way as the computation overhead causes the traffic simulation delay in CARLA. To address this, we use a robotic operating system (ROS) to simulate these communications. In general, an autonomous driving architecture on the ROS includes four types of ROS nodes: perception nodes, decision-making nodes, control nodes, and utility nodes. These nodes correspond to perception, planning, and control tasks performed in autonomous driving. The ROS bridge package in CARLA provides the interface for ROS nodes to access built-in functions and data structures in CARLA. To this end, the proposed IDS can be implemented in a real autonomous driving system via a ROS for better adaptability under real attacks.

# Bibliography

- [1] “Common Vulnerability Scoring System (CVSS).” [Online]. Available: <https://www.first.org/cvss/>
- [2] S. Achleitner, T. L. Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, “Deceiving network reconnaissance using SDN-based virtual topologies,” *IEEE Transactions on Network and Service Management*, vol. 14, pp. 1098–1112, Dec. 2017.
- [3] M. Agrawal, T. Huang, J. Zhou, and D. Chang, “Can-fd-sec: improving security of can-fd protocol,” in *Security and Safety Interplay of Intelligent Software Systems: ESORICS 2018 International Workshops, ISSA 2018 and CSITS 2018, Barcelona, Spain, September 6–7, 2018, Revised Selected Papers*. Springer, 2019, pp. 77–93.
- [4] S. O. Al-Mamory and H. Zhang, “Intrusion detection alarms reduction using root cause analysis and clustering,” *Computer Communications*, vol. 32, no. 2, pp. 419–430, 2009.
- [5] —, “New data mining technique to enhance IDS alarms quality,” *Journal in Computer Virology*, vol. 6, no. 1, pp. 43–55, 2010.
- [6] R. Albert, H. Jeong, and A. Barabási, “Error and attack tolerance of complex networks,” *nature*, vol. 406, no. 6794, pp. 378–382, 2000.
- [7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

- [8] A. Avizienis, “On the implementation of N-version programming for software fault tolerance during execution,” *Proc. COMPSAC*, pp. 149–155, 1977.
- [9] —, “The N-version approach to fault-tolerant software,” *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1491–1501, Dec. 1985.
- [10] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.
- [11] A.-L. Barabási, *Network Science*, 1st ed. Cambridge University Press, 2016.
- [12] M. Bellare and P. Rogaway, “Entity authentication and key distribution,” in *Annual international cryptology conference*. Springer, 1993, pp. 232–249.
- [13] S. Benferhat, A. Boudjelida, K. Tabia, and H. Drias, “An intrusion detection and alert correlation approach based on revising probabilistic classifiers using expert knowledge,” *Applied Intelligence*, vol. 38, no. 4, pp. 520–540, 2013.
- [14] Car-Hacking Dataset. (2016). [Online]. Available: <http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset>
- [15] J. L. Cardy and P. Grassberger, “Epidemic models and percolation,” *Journal of Physics A: Mathematical and General*, vol. 18, no. 6, p. L267, 1985.
- [16] CARLA autonomous driving leaderboard. (2023). [Online]. Available: <https://leaderboard.carla.org/>
- [17] C. G. Cassandras, “Smart cities as cyber-physical social systems,” *Engineering*, vol. 2, no. 2, pp. 156–158, 2016.

- [18] R. Chaâri, F. Ellouze, A. Koubâa, B. Qureshi, N. Pereira, H. Youssef, and E. Tovar, “Cyber-physical systems clouds: A survey,” *Computer Networks*, vol. 108, pp. 260–278, 2016.
- [19] X. Chai, Y. Wang, C. Yan, Y. Zhao, W. Chen, and X. Wang, “DQ-MOTAG: Deep reinforcement learning-based moving target defense against DDoS attacks,” in *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2020, pp. 375–379.
- [20] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.
- [21] D. Chen, V. Koltun, and P. Krähenbühl, “Learning to drive from a world on rails,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 590–15 599.
- [22] G. Chen, Y. Zhan, Y. Chen, L. Xiao, Y. Wang, and N. An, “Reinforcement learning based power control for in-body sensors in WBANs against jamming,” *IEEE Access*, vol. 6, pp. 37 403–37 412, 2018.
- [23] G. Chen, Y. Zhan, G. Sheng, L. Xiao, and Y. Wang, “Reinforcement learning-based sensor access control for wbans,” *IEEE Access*, vol. 7, pp. 8483–8494, 2019.
- [24] H. Chen, X. Li, and F. Zhao, “A reinforcement learning-based sleep scheduling algorithm for desired area coverage in solar-powered wireless sensor networks,” *IEEE Sensors Journal*, vol. 16, no. 8, pp. 2763–2774, 2016.
- [25] L. Chen and J. H. R. May, “A diversity model based on failure distribution and its application in safety cases,” *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1149–1162, Sept. 2016.

- [26] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: <http://doi.acm.org/10.1145/2939672.2939785>
- [27] Y. Chen, B. Boehm, and L. Sheppard, “Value driven security threat modeling based on attack path analysis,” in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS’07)*, 2007, pp. 280a–280a.
- [28] Q. Cheng, C. Wu, B. Hu, D. Kong, and B. Zhou, “Think that attackers think: Using first-order theory of mind in intrusion response system,” in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
- [29] J.-H. Cho, I.-R. Chen, and D.-C. Wang, “Performance optimization of region-based group key management in mobile ad hoc networks,” *Performance Evaluation*, vol. 65, no. 5, pp. 319–344, 2008.
- [30] J.-H. Cho, Y. Wang, I.-R. Chen, K. S. Chan, and A. Swami, “A survey on modeling and optimizing multi-objective systems,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1867–1901, 2017.
- [31] J.-H. Cho, S. Xu, P. M. Hurley, M. Mackay, T. Benjamin, and M. Beaumont, “STRAM: Measuring the trustworthiness of computer-based systems,” *ACM Computing Surveys*, vol. 51, no. 6, Feb. 2019.
- [32] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, “Toward proactive, adaptive defense: A survey on moving target defense,” *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

- [33] C. Colbourn, “Network Resilience,” *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 3, pp. 404–409, 1987.
- [34] M. Colezea, G. Musat, F. Pop, C. Negru, A. Dumitrascu, and M. Mocanu, “CLUeFARM: Integrated web-service platform for smart farms,” *Computers and Electronics in Agriculture*, vol. 154, pp. 134–154, 2018. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0168169917305112>
- [35] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, “Adversarial attack on graph structured data,” *Proceedings of Machine Learning Research (PMLR)*, vol. 80, pp. 1115–1124, 2018.
- [36] V.-A. Darvari, S. Hailes, and M. Musolesi, “Goal-directed graph construction using reinforcement learning,” *Proceedings of the Royal Society A*, vol. 477, no. 2254, p. 20210168, 2021.
- [37] D. Das, “A fuzzy multiobjective approach for network reconfiguration of distribution systems,” *IEEE Transactions on Power Delivery*, vol. 21, no. 1, pp. 202–209, 2005.
- [38] M. Denwood, “Clinican examination of the cow,” Oct. 2012. [Online]. Available: <https://www.gla.ac.uk/t4/~vet/files/teaching/clinicalexam/examination/info/temperatures.html>
- [39] A. Desai and S. Milner, “Autonomous reconfiguration in free-space optical sensor networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 8, pp. 1556–1563, 2005.
- [40] C. Dey, *Reducing IDS false positives using incremental stream clustering (ISC) algorithm*. Skolan för informations-och kommunikationsteknik, Kungliga Tekniska högskolan, 2009.

- [41] Z. Dezsö and A.-L. Barabási, “Halting viruses in scale-free networks,” *Physical Review E*, vol. 65, May 2002.
- [42] D. Ding, Q. Han, Z. Wang, and X. Ge, “A survey on model-based distributed control and filtering for industrial cyber-physical systems,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 5, pp. 2483–2499, 2019.
- [43] Q. Do, B. Martini, and K.-K. R. Choo, “Exfiltrating data from Android devices,” *Computers & Security*, vol. 48, pp. 74–91, 2015.
- [44] J. Dobaj, M. Krisper, and G. Macher, “Towards cyber-physical infrastructure as-a-service (cpiaas) in the era of industry 4.0,” in *European Conference on Software Process Improvement*. Springer, 2019, pp. 310–321.
- [45] H. Dong, Z. Ding, and S. Zhang, Eds., *Deep Reinforcement Learning Fundamentals, Research and Applications*. Springer, 2020.
- [46] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, “Boosting adversarial attacks with momentum,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9185–9193.
- [47] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
- [48] —, “CARLA: An open urban driving simulator,” in *Conference on robot learning*. PMLR, 2017, pp. 1–16.
- [49] H. T. Elshoush and I. M. Osman, “An improved framework for intrusion alert correlation,” in *Proceedings of the World Congress on Engineering*, vol. 1, 2012, pp. 1–6.

- [50] P. Erdős and A. Rényi, “On the evolution of random graphs,” in *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 1960, pp. 17–61.
- [51] FAO. (2020) The food and agriculture organization (fao) of the united nations. [Online]. Available: <http://www.fao.org/home/en/>
- [52] N. Fernández, R. J. D. Barroso, D. Siracusa, A. Francescon, I. de Miguel, E. Salvadori, J. C. Aguado, and R. M. Lorenzo, “Virtual topology reconfiguration in optical networks by means of cognition: Evaluation and experimental validation [invited],” *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 1, pp. A162–A173, 2015.
- [53] “Common Vulnerabilities and Exposures (CVE),” Forum of Incident Response and Security Teams. [Online]. Available: <https://cve.mitre.org/>
- [54] M. Franz, “E unibus pluram: Massive-scale software diversity as a defense mechanism,” in *Proceedings of the 2010 New Security Paradigms Workshop*, ser. NSPW ’10. New York, NY, USA: ACM, 2010, pp. 7–16.
- [55] C. Gao, G. Wang, W. Shi, Z. Wang, and Y. Chen, “Autonomous driving security: State of the art and challenges,” *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7572–7595, 2021.
- [56] M. Ge, J.-H. Cho, D. Kim, G. Dixit, and I.-R. Chen, “Proactive defense for internet-of-things: Moving target defense with cyberdeception,” *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 1, pp. 1–31, 2021.
- [57] R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter, “Information modeling for intrusion report aggregation,” in *Proceed-*

- ings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, vol. 1. IEEE, 2001, pp. 329–342.
- [58] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015. [Online]. Available: <http://arxiv.org/abs/1412.6572>
- [59] G. Grimmett, “Percolation and disordered systems,” in *Lectures on Probability and Statistics*. Springer Berlin Heidelberg, 1997, pp. 153–300.
- [60] C. Guo, Y. Ping, N. Liu, and S.-S. Luo, “A two-level hybrid approach for intrusion detection,” *Neurocomputing*, vol. 214, pp. 391–400, 2016.
- [61] M. Gupta, M. Abdelsalam, S. Khorsandroo, and S. Mittal, “Security and privacy in smart farming: Challenges and opportunities,” *IEEE Access*, vol. 8, pp. 34 564–34 584, 2020.
- [62] M. Hamad, M. Tsantekidis, and V. Prevelakis, “Intrusion response system for vehicles: Challenges and vision,” in *Smart Cities, Green Technologies and Intelligent Transport Systems*. Springer, 2019, pp. 321–341.
- [63] Y. Hamada, M. Inoue, N. Adachi, H. Ueda, Y. Miyashita, and Y. Hata, “Intrusion detection system for in-vehicle networks,” *SEI Tech. Rev.*, vol. 88, pp. 76–81, 2019.
- [64] K. J. Hole, “Diversity reduces the impact of malware,” *IEEE Security Privacy*, vol. 13, no. 3, pp. 48–54, May 2015.
- [65] —, “Toward anti-fragility: A malware-halting technique,” *IEEE Security Privacy*, vol. 13, no. 4, pp. 40–46, July 2015.

- [66] A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, and M. Franz, “Large-scale automated software diversity–program evolution redux,” *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 158–171, March 2017.
- [67] J. B. Hong and D. S. Kim, “Assessing the effectiveness of moving target defenses using security models,” *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 163–177, Mar. 2016.
- [68] J. B. Hong, S. Yoon, H. Lim, and D. S. Kim, “Optimal network reconfiguration for software defined networks using shuffle-based online mtd,” in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2017, pp. 234–243.
- [69] J. E. Hopcroft and R. M. Karp, “An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs,” *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.
- [70] S. Hosseini and M. A. Azgomi, “The dynamics of an SEIRS-QV malware propagation model in heterogeneous networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 512, pp. 803–817, 2018.
- [71] C. Huang, S. Zhu, and R. Erbacher, *Toward Software Diversity in Heterogeneous Networked Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, July 2014, pp. 114–129.
- [72] C. Huang, S. Zhu, Q. Guan, and Y. He, “A software assignment algorithm for minimizing worm damage in networked systems,” *Journal of Information Security and Applications*, vol. 35, no. Supplement C, pp. 55–67, 2017.
- [73] K. Hughes, K. McLaughlin, and S. Sezer, “A model-free approach to intrusion response systems,” *Journal of Information Security and Applications*, vol. 66, p. 103150, 2022.

- [74] A. Humayed, J. Lin, F. Li, and B. Luo, “Cyber-physical systems security—a survey,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.
- [75] S. Iannucci, V. Cardellini, O. D. Barba, and I. Banicescu, “A hybrid model-free approach for the near-optimal intrusion response control of non-stationary systems,” *Future Generation Computer Systems*, vol. 109, pp. 111–124, 2020.
- [76] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar, “Intrusion response systems: Foundations, design, and challenges,” *Journal of Network and Computer Applications*, vol. 62, pp. 53–74, 2016.
- [77] A. Izaddoost, E. Ogodó, and S. Prasai, “Enhanced data transmission platform in smart farms,” in *ACM Proceedings of the International Conference on Omni-Layer Intelligent Systems (COINS)*, New York, NY, USA, 2019, pp. 58–61.
- [78] H. M. Jawad, R. Nordin, S. K. Gharghan, A. M. Jawad, and M. Ismail, “Energy-efficient wireless sensor networks for precision agriculture: A review,” *Sensors*, vol. 17, no. 8, p. 1781, 2017.
- [79] S. Jin, J.-G. Chung, and Y. Xu, “Signature-based intrusion detection system (IDS) for in-vehicle CAN bus network,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.
- [80] A. Jøsang, J. Cho, and F. Chen, “Uncertainty characteristics of subjective opinions,” in *2018 21st International Conference on Information Fusion (FUSION)*, July 2018, pp. 1998–2005.
- [81] A. Jøsang, *Subjective Logic: A Formalism for Reasoning Under Uncertainty*. Springer Publishing Company, 2016.

- [82] T. Kaur and J. Baek, "A strategic deployment and cluster-header selection for wireless sensor networks," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1890–1897, 2009.
- [83] M. Keramati, A. Akbari, and M. Keramati, "CVSS-based security metrics for quantitative analysis of attack graphs," in *ICCKE 2013*, 2013, pp. 178–183.
- [84] A. Keselman, S. Ten, A. Ghazali, and M. Jubeh, "Reinforcement learning with A\* and a deep heuristic," *arXiv preprint arXiv:1811.07745*, 2018.
- [85] V. Kharchenko, "Diversity for safety and security of embedded and cyber physical systems: Fundamentals review and industrial cases," in *2016 15th Biennial Baltic Electronics Conference (BEC)*. IEEE, 2016, pp. 17–26.
- [86] H. A. Kholidy, "Autonomous mitigation of cyber risks in the cyber-physical systems," *Future Generation Computer Systems*, vol. 115, pp. 171–187, 2021.
- [87] F. Kiani, "Reinforcement learning based routing protocol for wireless body sensor networks," in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, 2017, pp. 71–78.
- [88] J. Knight, J. Davidson, A. Nguyen-Tuong, J. Hiser, and M. Co, "Diversity in cybersecurity," *Computer*, vol. 49, no. 4, pp. 94–98, Apr. 2016.
- [89] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
- [90] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2017.

- [91] H. Kwon, S. Lee, J. Choi, and B.-h. Chung, “Mitigation mechanism against in-vehicle network intrusion by reconfiguring ecu and disabling attack packet,” in *2018 International Conference on Information Technology (InCIT)*, 2018, pp. 1–5.
- [92] P. Larsen, S. Brunthaler, and M. Franz, “Security through diversity: Are we there yet?” *IEEE Security & Privacy*, vol. 12, no. 2, pp. 28–35, Mar. 2014.
- [93] —, “Automatic software diversity,” *IEEE Security & Privacy*, vol. 13, no. 2, pp. 30–37, Mar. 2015.
- [94] P. Laube and R. S. Purves, “How fast is a cow? cross-scale analysis of movement data,” *Transactions in GIS*, vol. 15, no. 3, pp. 401–418, 2011.
- [95] K. H. Law *et al.*, “IDS false alarm filtering using knn classifier,” in *International Workshop on Information Security Applications*. Springer, 2004, pp. 114–121.
- [96] A. S. Leong, D. E. Quevedo, A. Ahlén, and K. H. Johansson, “On network topology reconfiguration for remote state estimation,” *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3842–3856, 2016.
- [97] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” <http://snap.stanford.edu/data>, June 2014.
- [98] J. Leskovec and J. Mcauley, “Learning to discover social circles in ego networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [99] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations,” in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 177–187.

- [100] D. Li, Q. Zhang, E. Zio, S. Havlin, and R. Kang, “Network reliability analysis based on percolation theory,” *Reliability Engineering & System Safety*, vol. 142, pp. 556–562, 2015.
- [101] H. Li, K. Ota, and M. Dong, “Learning IoT in edge: Deep learning for the internet of things with edge computing,” *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.
- [102] T. Li, C. Feng, and C. Hankin, “Improving ICS cyber resilience through optimal diversification of network resources,” *CoRR*, vol. abs/1811.00142, 2018. [Online]. Available: <http://arxiv.org/abs/1811.00142>
- [103] A. X. Liu and M. G. Gouda, “Diverse firewall design,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1237–1251, 2008.
- [104] M. R. Lyu and L. K. Lau, “Firewall security: Policies, testing and performance evaluation,” in *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000*. IEEE, 2000, pp. 116–121.
- [105] W. Ma, “Analysis of anomaly detection method for internet of things based on deep learning,” *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 12, p. e3893, 2020.
- [106] A. Mađry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *stat*, vol. 1050, p. 9, 2017.
- [107] P. K. Manadhata and J. M. Wing, *A Formal Model for a System’s Attack Surface*. New York, NY: Springer New York, 2011, pp. 1–28.
- [108] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz, “A data mining analysis of rtid alarms,” *Computer Networks*, vol. 34, no. 4, pp. 571–577, 2000.

- [109] C. O. Mathuna, T. O'Donnell, R. V. Martinez-Catala, J. Rohan, and B. O'Flynn, "Energy scavenging for long-term deployable wireless sensor networks," *Talanta*, vol. 75, no. 3, pp. 613–623, 2008.
- [110] S. Mavoungou, G. Kaddoum, M. Taha, and G. Matar, "Survey on threats and attacks on mobile networks," *IEEE Access*, vol. 4, pp. 4543–4572, 2016.
- [111] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [112] C. Moore and M. E. Newman, "Epidemics and percolation in small-world networks," *Physical Review E*, vol. 61, no. 5, p. 5678, 2000.
- [113] Q. Mou, H. Ye, and Y. Liu, "Enabling highly efficient eigen-analysis of large delayed cyber-physical power systems by partial spectral discretization," *IEEE Transactions on Power Systems*, vol. 35, no. 2, pp. 1499–1508, Mar. 2020.
- [114] W. Najjar and J. L. Gaudiot, "Network resilience: a measure of network fault tolerance," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 174–181, Feb. 1990.
- [115] F. Neri, "Comparing local search with respect to genetic evolution to detect intrusions in computer networks," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, 2000, pp. 238–243 vol.1.
- [116] P. Nespoli, F. Gomez Marmol, and G. Kambourakis, "AISGA: Multi-objective parameters optimization for countermeasures selection through genetic algorithm," in *The 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–8.

- [117] M. Newman and D. Watts, “Scaling and percolation in the small-world network model,” *Physical Review E*, vol. 60, no. 6, pp. 7332–7342, 1999.
- [118] M. E. J. Newman, *Networks: An Introduction*, 1st ed. Oxford University Press, 2010.
- [119] D. C. K. Ngai and N. H. C. Yung, “A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 509–522, 2011.
- [120] J. O, D. Noh, and Y. Sohn, “Empirical test of Wi-Fi environment stability for smart farm platform,” in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, 2017, pp. 1–5.
- [121] A. J. O’Donnell and H. Sethu, “On achieving software diversity for improved networksecurity using distributed coloring algorithms,” in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ser. CCS ’04, 2004, pp. 121–131.
- [122] U. of Washington, “Rocketfuel maps and data,” <http://www.cs.washington.edu/research/networking/rocketfuel/>, April 2003.
- [123] Winning the future with science and technology for 21st century smart systems. Accessed: Jan. 11, 2020. [Online]. Available: <https://www.nitrd.gov/pubs/CPS-OSTP-Response-Winning-The-Future.pdf>
- [124] D. Parikh and T. Chen, “Data fusion and cost minimization for intrusion detection,” *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 3, pp. 381–389, 2008.
- [125] R. Perdisci, G. Giacinto, and F. Roli, “Alarm clustering for intrusion detection systems

- in computer networks,” *Engineering Applications of Artificial Intelligence*, vol. 19, no. 4, pp. 429–438, 2006.
- [126] T. Pietraszek, “Using adaptive alert classification to reduce false positives in intrusion detection,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 102–124.
- [127] T. Pietraszek and A. Tanner, “Data mining and machine learning—towards reducing false positives in intrusion detection,” *Information Security Technical Report*, vol. 10, no. 3, pp. 169–183, 2005.
- [128] P. Polack, F. Altché, B. d’Andréa Novel, and A. de La Fortelle, “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 812–818.
- [129] Y. Prieto, N. Boettcher, S. E. Restrepo, and J. E. Pezoa, “Optimal multiculture network design for maximizing resilience in the face of multiple correlated failures,” *Applied Sciences*, vol. 9, no. 11, p. 2256, 2019.
- [130] C. Pu, A. P. Black, C. Cowan, J. Walpole, and C. Consel, “A specialization toolkit to increase the diversity of operating systems,” 1996.
- [131] python-can. (2023). [Online]. Available: <https://github.com/hardbyte/python-can>
- [132] N. Qi, K. Dai, F. Yi, X. Wang, Z. You, and J. Zhao, “An adaptive energy management strategy to extend battery lifetime of solar powered wireless sensor nodes,” *IEEE Access*, vol. 7, pp. 88 289–88 300, 2019.
- [133] R. S. Rao, K. Ravindra, K. Satish, and S. Narasimham, “Power loss minimization in distribution system using network reconfiguration in the presence of distributed generation,” *IEEE Transactions on Power Systems*, vol. 28, no. 1, pp. 317–325, 2012.

- [134] R. Roman, J. Zhou, and J. Lopez, “On the features and challenges of security and privacy in distributed internet of things,” *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.
- [135] M. Roser. (2020) Future population growth. [Online]. Available: <https://ourworldindata.org/future-population-growth>
- [136] K. Salako and L. Strigini, “When does ‘diversity’ in development reduce common failures? insights from probabilistic modeling,” *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 2, pp. 193–206, Mar. 2014.
- [137] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “End-to-end deep reinforcement learning for lane keeping assist,” *Proceedings of NIPS Workshop on Machine Learning for Intelligent Transportation Systems (MLITS)*, vol. 2, pp. 1–9, 2016.
- [138] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: <http://arxiv.org/abs/1707.06347>
- [139] E. Seo, H. M. Song, and H. K. Kim, “GIDS: Gan based intrusion detection system for in-vehicle network,” in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–6.
- [140] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” *arXiv preprint arXiv:1610.03295*, 2016.
- [141] A. Shameli-Sendi and M. Dagenais, “ORCEF: Online response cost evaluation framework for intrusion response system,” *Journal of Network and Computer Applications*, vol. 55, pp. 89–107, 2015.

- [142] M. S. Shin, E. H. Kim, and K. H. Ryu, “False alarm classification model for network-based intrusion detection system,” in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2004, pp. 259–265.
- [143] J. P. Sterbenz, D. Hutchison, E. K. Cetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, “Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines,” *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.
- [144] H. Sun, Q. Zhu, J. Ren, D. Barclay, and W. Thomson, “Combining image analysis and smart data mining for precision agriculture in livestock farming,” in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017, pp. 1065–1069.
- [145] S. C. Sundaramurthy, L. Zomlot, and X. Ou, “Practical IDS alert correlation in the face of dynamic threats,” in *Proceedings of the International Conference on Security and Management (SAM)*. Citeseer, 2011, p. 1.
- [146] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
- [147] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, “Multiagent cooperation and competition with deep reinforcement learning,” *PloS one*, vol. 12, no. 4, p. e0172395, 2017.
- [148] Technical Marketing Workgroup, “A technical overview of lora® and lorawan™,” LoRa Alliance®, Fremont, CA, USA, Nov. 2015.
- [149] *CC2640R2F SimpleLink™ Bluetooth® 5.1 Low Energy Wireless MCU*, Texas

- Instruments, 2016, rev. C. [Online]. Available: <https://www.ti.com/product/CC2640R2F>
- [150] E. Totel, F. Majorczyk, and L. Mé, “Cots diversity based intrusion detection and application to web servers,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2005, pp. 43–62.
- [151] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, “Dependability and security models,” in *2009 7th International Workshop on Design of Reliable Communication Networks*, 2009, pp. 11–20.
- [152] S. Ullah, S. Shelly, A. Hassanzadeh, A. Nayak, and K. Hasan, “On the effectiveness of intrusion response systems against persistent threats,” in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 415–421.
- [153] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, “Comprehensive approach to intrusion detection alert correlation,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 146–169, 2004.
- [154] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.
- [155] G. Vigna and R. A. Kemmerer, “NetSTAT: A network-based intrusion detection approach,” in *Proceedings 14th Annual Computer Security Applications Conference (Cat. No. 98EX217)*. IEEE, 1998, pp. 25–34.
- [156] B. Walker, A. Kinzig, and J. Langridge, “Original articles: plant attribute diversity,

- resilience, and ecosystem function: the nature and significance of dominant and minor species,” *Ecosystems*, vol. 2, no. 2, pp. 95–113, March 1999.
- [157] Z. Wan, Y. Mahajan, B. W. Kang, T. J. Moore, and J.-H. Cho, “A survey on centrality metrics and their network resilience analysis,” *IEEE Access*, vol. 9, pp. 104 773–104 819, 2021.
- [158] P. Wang and C.-Y. Chan, “Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–6.
- [159] P. Wang, C.-Y. Chan, and A. de La Fortelle, “A reinforcement learning based approach for automated lane change maneuvers,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1379–1384.
- [160] 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.
- [161] A. D. Wood and J. A. Stankovic, “Denial of service in sensor networks,” *Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.
- [162] M. Woolley, “The bluetooth low energy primer,” Bluetooth SIG, Kirkland, WA, USA, Jun. 2022.
- [163] H. Wu and B. Preneel, “Aegis: A fast authenticated encryption algorithm,” in *Selected Areas in Cryptography–SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers 20*. Springer, 2014, pp. 185–201.

- [164] R. Xing, Z. Su, N. Zhang, Y. Peng, H. Pu, and J. Luo, "Trust-evaluation-based intrusion detection and reinforcement learning in autonomous driving," *IEEE Network*, vol. 33, no. 5, pp. 54–60, 2019.
- [165] X. Xiong, J. Wang, F. Zhang, and K. Li, "Combining deep reinforcement learning and safety based control for autonomous driving," *arXiv preprint arXiv:1612.00147*, 2016.
- [166] Y. Yang, S. Zhu, and G. Cao, "Improving sensor network immunity under worm attacks: A software diversity approach," in *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '08, 2008, pp. 149–158.
- [167] —, "Improving sensor network immunity under worm attacks: A software diversity approach," *Ad Hoc Networks*, vol. 47, no. Supplement C, pp. 26–40, 2016.
- [168] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.
- [169] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: A security metric for evaluating the resilience of networks against zero-day attacks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 1071–1086, May 2016.
- [170] Q. Zhang, J.-H. Cho, and T. J. Moore, "Network resilience under epidemic attacks: Deep reinforcement learning network topology adaptations," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–7.
- [171] Q. Zhang, J.-H. Cho, T. J. Moore, and I.-R. Chen, "Vulnerability-aware resilient net-

- works: Software diversity-based network adaptation,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3154–3169, 2021.
- [172] Q. Zhang, J.-H. Cho, T. J. Moore, and F. F. Nelson, “Drevan: Deep reinforcement learning-based vulnerability-aware network adaptations for resilient networks,” in *2021 IEEE Conference on Communications and Network Security (CNS)*, 2021, pp. 137–145.
- [173] Q. Zhang, Y. Mahajan, I.-R. Chen, D. S. Ha, and J.-H. Cho, “An attack-resilient and energy-adaptive monitoring system for smart farms,” in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 2776–2781.
- [174] Q. Zhang, A. Z. Mohammed, Z. Wan, J.-H. Cho, and T. J. Moore, “Diversity-by-design for dependable and secure cyber-physical systems: A survey,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 706–728, 2022.
- [175] Q. Zhang, D. Chen, Y. Mahajan, I.-R. Chen, D. S. Ha, and J.-H. Cho, “Attack-resistant, energy-adaptive monitoring for smart farms: Uncertainty-aware deep reinforcement learning approach,” *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 254–14 268, 2023.
- [176] Q. Zhang, J.-H. Cho, T. J. Moore, D. Kim, H. Lim, and F. F. Nelson, “Evade: Efficient moving target defense for autonomous network topology shuffling using deep reinforcement learning,” in *Applied Cryptography and Network Security: 21th International Conference, ACNS 2023, Kyoto, Japan, June 19-22, 2023, Proceedings 21*. Springer, 2023.
- [177] Q. Zhang, T. J. Moore, D. Kim, H. Lim, S. Yoon, F. F. Nelson, and J.-H. Cho, “Intrusion response system integrated for deep reinforcement learning-based autonomous driving,” in *The Second Symposium on Vehicle Security and Privacy (VehicleSec 2024)*, 2024, under review.

- [178] T. Zhang, C. Xu, B. Zhang, X. Kuang, Y. Wang, S. Yang, and G.-M. Muntean, “Dq-rm: Deep reinforcement learning-based route mutation scheme for multimedia services,” in *2020 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2020, pp. 291–296.
- [179] Y. Zhang, H. Vin, L. Alvisi, W. Lee, and S. K. Dao, “Heterogeneous networking: A new survivability paradigm,” in *Proceedings of Network Security Paradigms Workshop (NSPW’01)*. Cloudcroft, New Mexico, USA: ACM, Sep. 2001.
- [180] Y. Zhang, M. Murata, H. Takagi, and Yusheng Ji, “Traffic-based reconfiguration for logical topologies in large-scale wdm optical networks,” *Journal of Lightwave Technology*, vol. 23, no. 10, pp. 2854–2867, 2005.
- [181] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, “End-to-end urban driving by imitating a reinforcement learning coach,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 222–15 232.
- [182] Z.-H. Zhang, H. Shen, and Y.-P. Sang, “An observation-centric analysis on the modeling of anomaly-based intrusion detection,” *International Journal of Network Security*, vol. 4, no. 3, pp. 292–305, 2007.
- [183] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, “Federated reinforcement learning,” *CoRR*, vol. abs/1901.08277, 2019. [Online]. Available: <http://arxiv.org/abs/1901.08277>

# Appendices

# Appendix A

## Appendix for Chapter 3

Here we attach the appendix for our TNSM paper [171].

### A.1 Algorithms

The following algorithms are referenced from the TNSM paper [171]. They are as follows:

- **Algorithm 3 (SDBA)**: Remove edges between two nodes with the same software package, which is Step 1 in the SDA algorithm (Algorithm 3).
- **Algorithm 4 (GenPV)**: Generate path vulnerabilities. This algorithm returns a vector  $\mathbf{PV}$  that contains the maximum attack path vulnerability where each attack path is a disjoint shortest path from node  $i$  to a node with the maximum  $k - 1$  hop distance.
- **Algorithm 5 (SetEAB)**: Set edge adaptation budget. This algorithm returns  $T^{global}$ , which is the total number of edges to be adapted based on a given fraction of edges to be adapted,  $\rho$ , and a vector of the number of edges to be adapted per node,  $\mathbf{T}^{local}$ .
- **Algorithm 6 (GEAC)**: Generate edge addition candidates. This algorithm returns a vector of edge candidates to be added based on the lowest software diversity reduction.

- **Algorithm 7 (GERC)**: Generate edge removal candidates. This algorithm returns a vector of edge candidates to be removed based on the highest software diversity increase.
- **Algorithm 8 (AdaptNT)**: Adapt network topology. Based on the above two algorithms, **GEAC** and **GERC**,  $T^{global}$  number of edges are adapted (removed or added) where each edge is adapted based on  $\mathbf{T}^{local}$  set in **SetEAB** above.
- **Algorithm 9 (Random-A)**: This algorithm is one of comparable counterpart schemes and compared with the proposed SDA-based schemes. This algorithm uses the same procedure in Step 1 as the SDA (i.e., **SDBA** in Algorithm 3). In Step 2, an edge addition is performed randomly for the number of nodes lost in Step 1 and it is only based on the condition that two nodes use different software packages.
- **Algorithm 10**: This algorithm describes how the epidemic attacks are modeled in this work.

---

**Algorithm 3** Software Diversity-based Basic Adaptation (SDBA)

---

```

1:  $N \leftarrow$  The total number of nodes in a network
2:  $\mathbf{DN} \leftarrow$  A vector containing the number of removed edges per node
3:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
4:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
5:  $\mathbf{A}' \leftarrow$  An adjacency matrix after edges are adapted
6:
7:  $\mathbf{A}' = \text{SDBA}(\mathbf{DN}, \mathbf{A}, \mathbf{S})$   $\triangleright$  Remove edges between two nodes with the same software package.
8: for  $i := 1$  to  $N$  do
9:   for  $j := 1$  to  $N$  do
10:    if  $(a_{ij} > 0) \wedge s_i == s_j$  then
11:       $a_{ij} = 0$ 
12:       $a_{ji} = 0$ 
13:       $\mathbf{DN}(i) = \mathbf{DN}(i) + 1$ 
14:       $\mathbf{DN}(j) = \mathbf{DN}(j) + 1$ 
15:    end if
16:  end for
17: end for
18: return  $\mathbf{A}'$ 

```

---



---

**Algorithm 4** Generate Path Vulnerabilities (GenPV)

---

```

1:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
2:  $\mathbf{APV} \leftarrow$  A vector of vulnerabilities of attack paths  $apv_{ij}$  where  $j = 1, \dots, 20$   $\triangleright$  For simplicity, consider maximum 20 shortest disjoint attack paths reachable to node  $i$ .
3:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
4:  $k \leftarrow$  A hop distance given in a node's local network
5:  $\mathbf{PV} \leftarrow$  A vector of estimated maximum path vulnerabilities for all nodes  $i$  where  $pv_i$  refers to the estimated maximum attack path vulnerability where the path consists of at most  $k - 1$ -hop distance from node  $i$   $\triangleright$  Count one hop from a node to any neighboring node and thus  $k$  is decremented by 1.
6:  $\mathbf{PV} = \text{GenPV}(\mathbf{A}, \mathbf{APV}, \mathbf{S}, k)$ 
7: for  $i := 1$  to  $N$  do
8:    $pv_i = \max_j apv_{ij}^{k-1}$ 
9: end for
10: return  $\mathbf{PV}$ 

```

---

## A.2 Real Network Topologies Used

This work develops a topology-aware notion of software diversity and, hence it makes sense to study the effect of different network topologies. To meet this objective, we considered

**Algorithm 5** Set Edge Adaptations Budget (**SetEAB**)

---

```

1:  $\mathbf{DN} \leftarrow$  A vector containing the number of removed edges per node
2:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
3:  $\rho \leftarrow$  A threshold referring to the fraction of edges to be adapted
4:  $\kappa_i \leftarrow$  Node  $i$ 's degree
5:  $T^{global} \leftarrow$  The total number of edges to be adapted
6:  $\mathbf{T}^{local} \leftarrow$  A vector of the number of edges that are allowed to be adapted per node where an
   element is denoted by  $T_i^{local}$  for node  $i$  for  $i = 1, \dots, N$ 
7:
8:  $\mathbf{T}^{local}, T^{global} = \mathbf{SetEAB}(\mathbf{DN}, \mathbf{A}, \rho)$ 
9:  $T^{global} = \lfloor \rho \frac{\text{sum}(\mathbf{DN})}{2} \rfloor$ 
10:  $\kappa = \frac{\text{sum}(\mathbf{A}) + T^{global}}{N}$   $\triangleright$  An expected average degree after  $T^{global}$  number of edges are adapted.
11: if  $\rho > 0$  then
12:    $T_i^{local} \leftarrow \max(0, \kappa - \kappa_i)$   $\triangleright T_i^{local}$  captures the number of edges that can be added.
13:    $N_{HD} = \sum_{i=1}^N \max(0, \kappa_i - \kappa) - T^{global}$ 
14: else
15:    $T_i^{local} \leftarrow \max(0, \kappa_i - \kappa)$   $\triangleright T_i^{local}$  captures the number of edges that can be removed.
16:    $N_{HD} = \sum_{i=1}^N \max(0, \kappa - \kappa_i) - T^{global}$ 
17: end if  $\triangleright$  Positive  $N_{HD}$  means that more nodes have higher (or lower)
   degrees than the expected average degree while negative  $N_{HD}$  implies fewer nodes have higher
   (lower) degrees than the average expected degree after edge adaptations. Hence, when  $N_{HD}$  is
   positive, edge adaptations should be restricted further.
18: while  $N_{HD} > 0$  do
19:   for  $i := 1$  to  $N$  do
20:     if  $T_i^{local} > 0 \wedge N_{HD} > 0$  then
21:        $T_i^{local} = T_i^{local} - 1$ 
22:        $N_{HD} = N_{HD} - 1$ 
23:     end if
24:   end for
25: end while
26: return  $T^{global}, \mathbf{T}^{local}$ 

```

---

three different network datasets, each with different levels of network density.

A dense network (DN) is generated from a Facebook ego network, which is a connected component from the one of the 10 ego networks provided by SNAP [97]. This network has density of approximately 0.05 with a mean degree near 52. A visualization of this network is shown in Fig. A.1 (a) and its degree distribution is shown in Fig. A.1 (d). The distribution is right-skewed despite the high mean.

---

**Algorithm 6** Generate Edge Addition Candidates (**GEAC**)
 

---

```

1:  $N \leftarrow$  The total number of nodes in a network
2:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
3:  $\mathbf{SD} \leftarrow$  A vector of software diversity values,  $sd_i$  for all nodes  $i = 1, \dots, N$ 
4:  $\mathbf{SV} \leftarrow$  A vector of software vulnerabilities,  $sv_i$  for all nodes  $i = 1, \dots, N$ 
5:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
6:  $\mathbf{PV} \leftarrow$  A vector of estimated maximum path vulnerabilities for all nodes  $i$ 
7:  $\mathbf{T}^{local} \leftarrow$  A vector of the number of edges that are allowed to be adapted per node where an
   element is denoted by  $T_i^{local}$  for node  $i$  for  $i = 1, \dots, N$ 
8:
9: add_candidate = GEAC( $\mathbf{A}, \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local}$ )
10:  $\mathbf{A}^* \leftarrow (\mathbf{A} + \mathbf{I})^{2k}$  where  $a_{ij}^*$  is 1 when nodes  $i$  and  $j$  belong to each other's local network or its
   neighbor's local networks; 0 otherwise.
11:  $r \leftarrow$  counter initialized at 0.
12: for  $i := 1$  to  $N$  do
13:   for  $j := i + 1$  to  $N$  do
14:     if  $a_{ij}^* > 0 \wedge a_{ij} == 0 \wedge s_i \neq s_j$  then
15:       sd_diff_sum( $r$ ) =  $(sd_i - sd'_i) + (sd_j - sd'_j)$ 
16:        $\triangleright$  Sum of the improved software diversity values
         by both node  $i$  and node  $j$  where the expected software diversity values of nodes  $i$  and  $j$  after
         edge addition adaptations are  $sd'_i = sd_i(1 - sv_i pv_j)$ ,  $sd'_j = sd_j(1 - sv_j pv_i)$  based on Eq. (4) of
         the main paper (i.e., software diversity metric).
17:        $r = r + 1$ 
18:     end if
19:   end for
20: end for
21: Rank sd_diff_sum in an ascending order and capture top  $\mathbf{T}^{local}$  edges in add_candidate
22: return add_candidate

```

---

A medium dense network (MN) is generated from a subset of the Enron email dataset on SNAP [97]. This medium dense network is generated by the largest connected component of the induced subgraph of nodes ranked between 501 and 1500 by degree. This was done to generate a network with a similar size to the dense network from an original network that starts with less density. The density is 0.016 with a mean degree near 16. A visualization of this network is shown in Fig. A.1 (b) and its degree distribution is shown in Fig. A.1 (e). The distribution has the shape close to a binomial distribution.

A sparse network (SN) is generated from an observation of the Internet at the autonomous

**Algorithm 7** Generate Edge Removal Candidates (**GERC**)

---

```

1:  $N \leftarrow$  The total number of nodes in a network
2:  $\mathbf{A} \leftarrow$  An adjacency matrix for a given network with element  $a_{ij}$  for  $i, j = 1, \dots, N$ 
3:  $\mathbf{SD} \leftarrow$  A vector of software diversity values,  $sd_i$  for all nodes  $i = 1, \dots, N$ 
4:  $\mathbf{SV} \leftarrow$  A vector of software vulnerabilities,  $sv_i$  for all nodes  $i = 1, \dots, N$ 
5:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
6:  $\mathbf{PV} \leftarrow$  A vector of estimated maximum path vulnerabilities for all nodes  $i$ 
7:  $\mathbf{T}^{local} \leftarrow$  A vector of the number of edges that are allowed to be adapted per node where an
   element is denoted by  $T_i^{local}$  for node  $i$  for  $i = 1, \dots, N$ 
8:
9: remove_candidate = GERC( $\mathbf{A}, \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local}$ )
10: for  $i := 1$  to  $N$  do
11:   for  $j := i + 1$  to  $N$  do
12:     if  $a_{ij} > 0$  then
13:       sd_diff_sum( $r$ ) =  $(sd'_i - sd_i) + (sd'_j - sd_j)$ 
14:        $\triangleright$  Sum of the improved software diversity values by both node
          $i$  and node  $j$  where the expected software diversity values of nodes  $i$  and  $j$  after edge removal
         adaptations are  $sd'_i = sd_i / (1 - sv_i pv_j)$ ,  $sd'_j = sd_j / (1 - sv_j pv_i)$  based on Eq. (3.6) and Eq. (3.8)
         (i.e., software diversity metrics).
15:        $r = r + 1$ 
16:     end if
17:   end for
18: end for
19: Rank sd_diff_sum in an descending order and capture top  $\mathbf{T}^{local}$  edges in
   remove_candidate
20: return remove_candidate

```

---

systems level [97]. Edges are removed from the recorded data to ensure a simple network. The density of this network is approximately 0.003 and the mean node degree is approximately 4.4. A visualization of this network is shown in Fig. A.1 (c) and its degree distribution is shown in Fig. A.1 (f). The distribution skews right and is close to a power-law shape.

## A.3 Experiments Under a Random Network

The work also considers a random network topology generated by an Erdős-Rényi (ER) random graph model  $G(n, p)$  where  $n$  is the number of nodes and  $p$  is the connection probability

---

**Algorithm 8** Adapt Network Topology (**AdaptNT**)
 

---

```

1:  $\mathbf{DN} \leftarrow$  a vector with # of disconnected edges per node
2:  $\mathbf{A} \leftarrow$  an adjacency matrix for a given network
3:  $\rho \leftarrow$  threshold of the fraction of edges adapted
4: candidate  $\leftarrow$  A vector of edge candidates, either add_candidate in Algorithm 6 or
   remove_candidate in Algorithm 7
5:  $\mathbf{T}^{local}, \mathbf{T}^{global} = \mathbf{SetEAB}(\mathbf{DN}, \mathbf{A}, \rho)$  based on Algorithm 5
6:  $\mathbf{A}' \leftarrow$  An adjacency network after edges are adapted
7:
8:  $\mathbf{A}' = \mathbf{AdaptNT}(\mathbf{A}, \mathbf{candidate}, \mathbf{T}^{local}, \mathbf{T}^{global}, \rho)$ 
9: for  $(i, j, \mathbf{sd\_diff\_sum})$  in candidate do
10:   if  $T_i^{local} T_j^{local} > 0$  then
11:     if  $\rho > 0$  then
12:        $a_{ij} = a_{ji} = 1$ 
13:     else
14:        $a_{ij} = a_{ji} = 0$ 
15:     end if
16:      $T_i^{local} = T_i^{local} - 1$ 
17:      $T_j^{local} = T_j^{local} - 1$ 
18:      $T^{global} = T^{global} - 1$ 
19:   end if
20: end for
21: for  $(i, j, \mathbf{sd\_diff\_sum})$  in candidate do
22:   if  $T^{global} > 0$  then
23:     if  $\rho > 0 \wedge a_{ij} == 0$  then
24:        $a_{ij} = a_{ji} = 1$ 
25:        $T^{global} = T^{global} - 1$ 
26:     else
27:       if  $\rho < 0 \wedge a_{ij} > 0$  then
28:          $a_{ij} = a_{ji} = 0$ 
29:          $T^{global} = T^{global} - 1$ 
30:       end if
31:     end if
32:   end if
33: end for
34: return  $\mathbf{A}'$ 

```

---

between any pair of nodes [118]. This model is ideal to study the effect of network density in a general manner by varying the connection probability  $p$  since the network has density approximately  $p$  for large  $n$ . It is used here for a sensitivity analysis and comparative analysis for the methods considered in the paper.

**Algorithm 9** Random Adaptation (Random-A)

---

```

1:  $N \leftarrow$  The total number of nodes in a network
2:  $\mathbf{DN} \leftarrow$  a vector with # of disconnected edges per node
3:  $\mathbf{A} \leftarrow$  an adjacency matrix for a given network
4:  $\mathbf{S} \leftarrow$  A vector of software packages installed over nodes with element  $s_i$  for  $i = 1, \dots, N$ 
5:  $\mathbf{A}' \leftarrow$  An adjacency matrix after edges are adapted
6:
7:  $\mathbf{A}' = \mathbf{Random-A}(\mathbf{DN}, \mathbf{A}, \mathbf{S})$ 
8: Step 1:  $\mathbf{A}' = \mathbf{SDBA}(\mathbf{DN}, \mathbf{A}, \mathbf{S})$   $\triangleright$  Remove edges between two nodes with the same software
   package (see Algorithm 3).
9:
10: Step 2: Add random edges between two disconnected nodes if their software packages are
    different
11: for  $i := 1$  to  $N$  do
12:   candidate  $\leftarrow$  a vector of nodes that can be connected with node  $i$  where  $a'_{ij} == 0 \wedge (na_i \cdot$ 
      $na_j > 0)$  for node  $j$ 
13:   visited  $\leftarrow$  a vector with  $length(\mathbf{candidate})$ 
14:   for  $j := 1$  to  $\mathbf{DN}(i)$  do
15:      $r \leftarrow$  A random integer selected from  $[0, length(\mathbf{candidate})]$  at random.
16:     if  $\mathbf{visited}(r) == 0 \wedge \mathbf{DN}(r) > 0$  then
17:        $a'_{ir} = 1$ 
18:        $a'_{ri} = 1$ 
19:        $\mathbf{DN}(i) = \mathbf{DN}(i) - 1$ 
20:        $\mathbf{DN}(r) = \mathbf{DN}(r) - 1$ 
21:        $\mathbf{visited}(r) = 1$ 
22:     else
23:        $j = j - 1$ 
24:     end if
25:     if  $sum(\mathbf{visited}) == length(\mathbf{visited})$  then
26:        $\triangleright$  All candidate nodes are selected.
27:       break
28:     end if
29:   end for
30: end for
31: return  $\mathbf{A}'$ 

```

---

**A.3.1 Visualization of an ER Network**

A visualization of the example ER network for  $n = 1000$  and  $p = 0.025$  is shown in Fig. A.2 (a). This realization of the ER model has density 0.02497, which is very close to  $p$ , and has the mean node degree 24.946, which is close to the expected value  $p(n - 1)$ . The degree

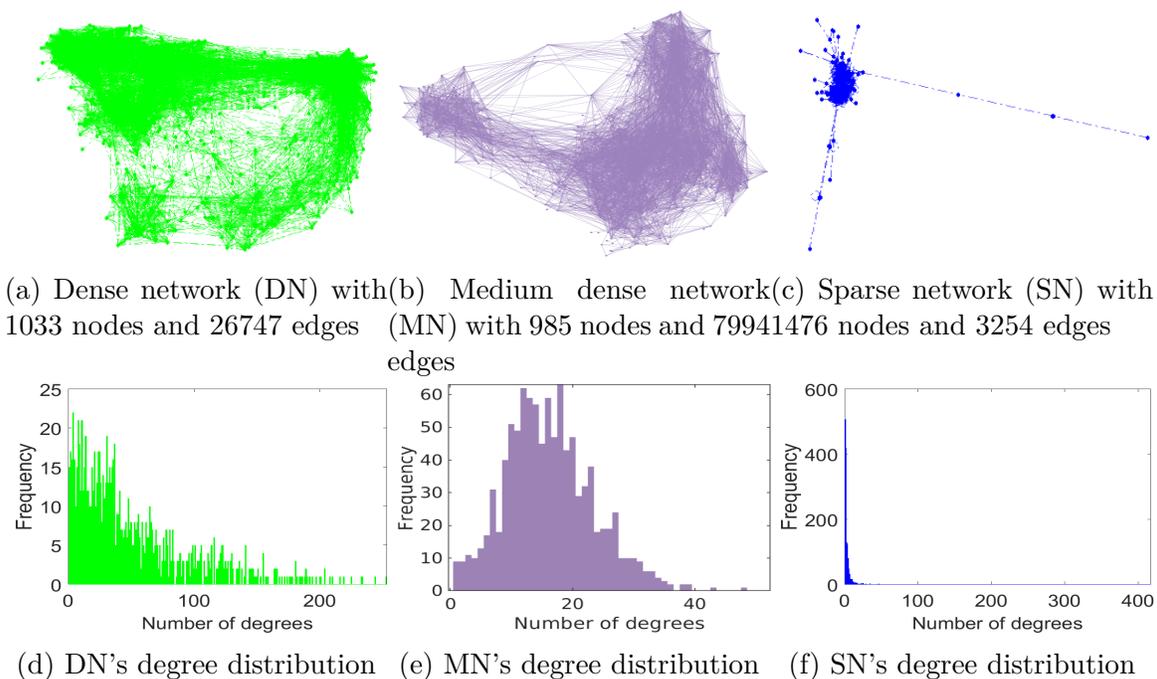


Figure A.1: The network topologies used for our experimental study and their degree distributions.

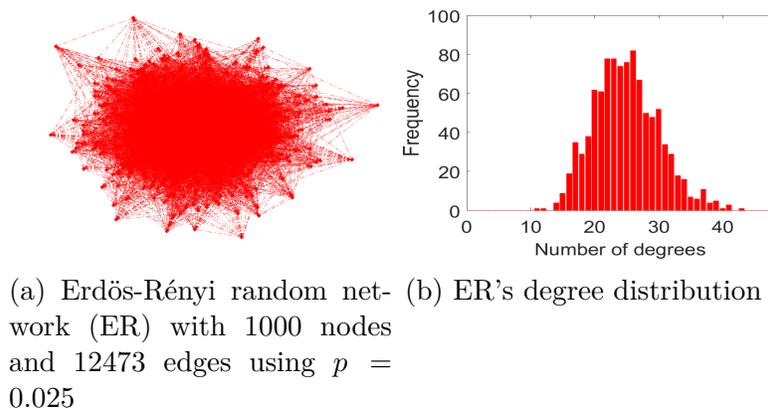
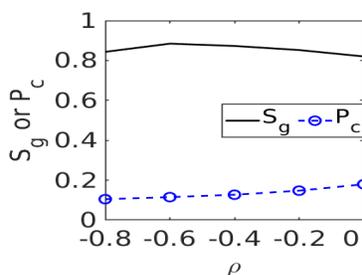


Figure A.2: The network topology of a random network used for our experimental study and its degree distribution.

distribution of this network realization, shown in Fig. A.2 (b), has is very close to a binomial distribution.

### A.3.2 Identification of an Optimal Fraction of Edges to be Adapted ( $\rho$ )

We conducted a sensitivity analysis of the effect of the fraction of edges to be adapted ( $\rho$ ) for the random network here. Fig. A.3 shows the size of the giant component ( $S_g$ ) and the fraction of compromised nodes ( $P_c$ ) versus varying  $\rho$ . We used  $P_a = 0.1$ ,  $N_s = 5$ , and  $p = 0.025$  for corresponding simulations. We observe that  $S_g$  attains its maximum when  $\rho = -0.6$ , which will be used as the optimal  $\rho$  for the comparative analysis conducted in Section A.3.3.



(a) Random Network

Figure A.3: Effect of the threshold of the fraction of edges adapted ( $\rho$ ) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ).

### A.3.3 Comparative Performance Analysis

This section details our comparative analysis of the six schemes introduced in Section 3.7 under a random network.

#### Effect of Network Density Under a Random Network

Fig. A.4 shows how the network density, controlled by the ER connectivity parameter  $p$ , affects the performance of the six schemes with respect to the four performance metrics listed

in Section 3.6.1 of the main paper. Overall, as the node connection probability  $p$  increases, the network is exposed to higher vulnerability with respect to epidemic attacks because the attackers have more neighbors and, hence, more potential nodes to compromise. Since the number of software packages available  $N_s$  is limited to 5, as density increases, an attacker has a better chance to come into contact with a neighboring node that has a software package shared or previously learned by the attacker. Since higher network connectivity naturally leads to higher vulnerability to epidemic attacks, as more nodes are connected, the attacks exhibit a greater impact. After attacks on the original or adapted network, the resulting outcome is a less connected network with a smaller size of the giant component. This post-attack network also has a lower software diversity value as a fewer number of nodes will be considered in the attack path to estimate the software diversity, as shown in Eq. (3.4) in Section 3.5.1. Therefore, when  $p$  is low, significant performance differences by all schemes are not observed. On the other hand, as  $p$  increases, schemes with lower adapted network density (i.e., SDA with  $\rho = 0$  and SDA with optimal  $\rho = -0.6$ ) outperform the other counterparts (i.e., No-A, Random-A, Random-Graph-C and SDA with  $\rho = 1$ ).

Overall the best performing method with respect to the three metrics is shown with SDA with  $\rho = -0.6$ . SDA with  $\rho = -0.6$  also shows significant resilience among all four performance metrics. The performance order in the fraction of compromised nodes ( $P_c$ ), the size of the giant component ( $S_g$ ) and software diversity ( $SD$ ) is: SDA with  $\rho = -0.6 \geq$  SDA with  $\rho = 0 \geq$  SDA with  $\rho = 1 \approx$  Random-A  $\geq$  Random-Graph-C  $\approx$  No-A. Although Random-Graph-C exhibits slightly better performance than No-A, both schemes perform nearly identically. This implies that the contribution of shuffling is minimal under the random network model. With respect to the defense cost ( $D_c$ ), the overall performance order follows: SDA with  $\rho = 0 \approx$  No-A  $\geq$  SDA with  $\rho = 1 \approx$  Random-A  $\geq$  SDA with  $\rho = -0.6 \geq$  Random-Graph-C. Random-Graph-C incurs the highest cost since the shuffling cost and the

cost caused by the IDS are combined when calculating the defense cost using Eq. (3.12) in Section 3.6.1 of the main paper. Although adaptation schemes incur a higher cost than No-A, as nodes are more connected with higher  $p$ , SDA with  $\rho = 0$  generates a significantly lower cost than the other existing counterparts. This is because well-adapted network topologies are less vulnerable to epidemic attacks. In these networks, a significantly fewer number of nodes become compromised and, hence, detected by the IDS, which disconnects all the edges from the detected and compromised nodes. On the other hand, non-adapted or poorly adapted network topologies result in more actions by the IDS, which will disconnect all the edges of nodes that are detected as compromised to protect the network itself.

### Effect of the Fraction of Initial Seeding Attackers ( $P_a$ ) under a Random Network

Fig. A.5 demonstrates how the different levels of attack density impact the performance of all comparing schemes with respect to the performance metrics in Section 3.6.1 under the random network topology. The overall trends of the performance as more seeding attackers are added in the network are as follows. An increase of the fraction of seeding attackers decreases software diversity and the size of the giant component, while at the same time increasing the fraction of compromised nodes and defense costs. This latter effect is because more nodes become compromised and accordingly more site percolation-based adaptations are required by the IDS (i.e., disconnecting all edges to a detected, compromised node). Also across the range of the attack density and with respect to all metrics except defense cost ( $D_c$ ), the overall performance order clearly follows: SDA with  $\rho = -0.6 \geq$  SDA with  $\rho = 0 \geq$  SDA with  $\rho = 1 \approx$  Random-A  $\geq$  Random-Graph-C  $\approx$  No-A. As for defense cost ( $D_c$ ), the overall performance order follows: SDA with  $\rho = 0 \approx$  No-A  $\geq$  SDA with  $\rho = 1 \approx$  Random-A  $\geq$  SDA with  $\rho = -0.6 \geq$  Random-Graph-C.

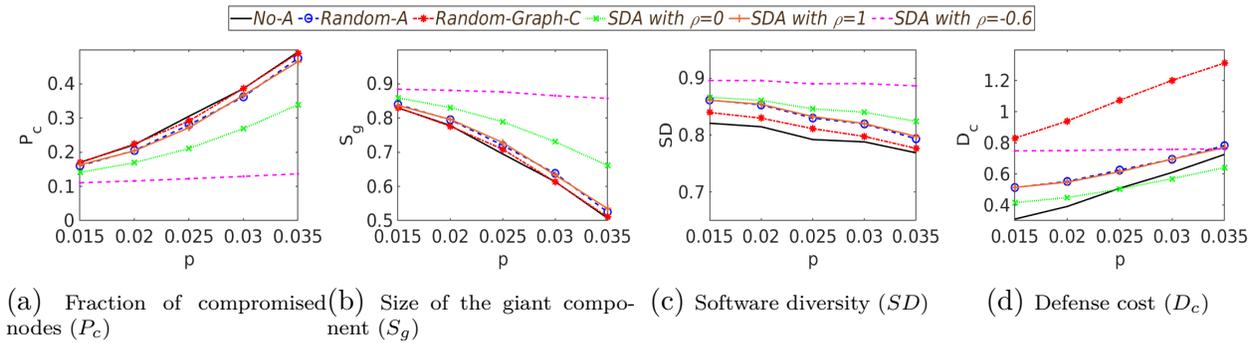


Figure A.4: Effect of the connection probability between two nodes ( $p$ ) under random networks.

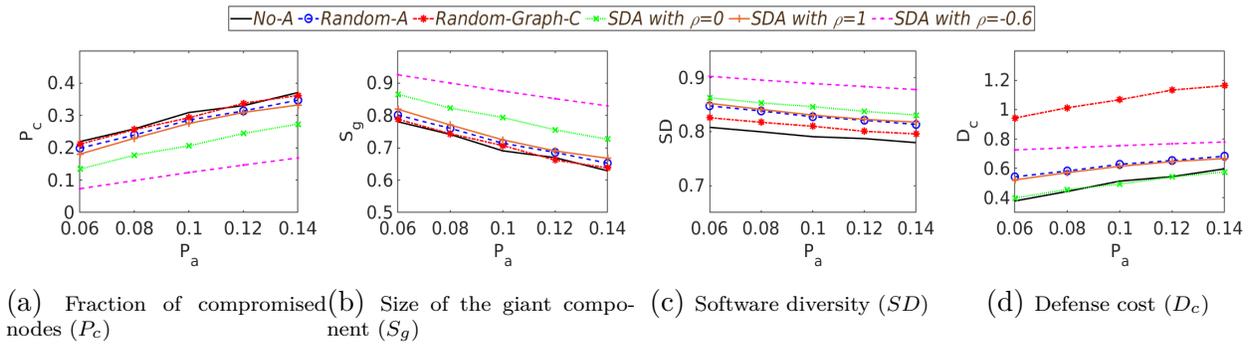


Figure A.5: Effect of varying the fraction of seeding attacks ( $P_a$ ) under a random network.

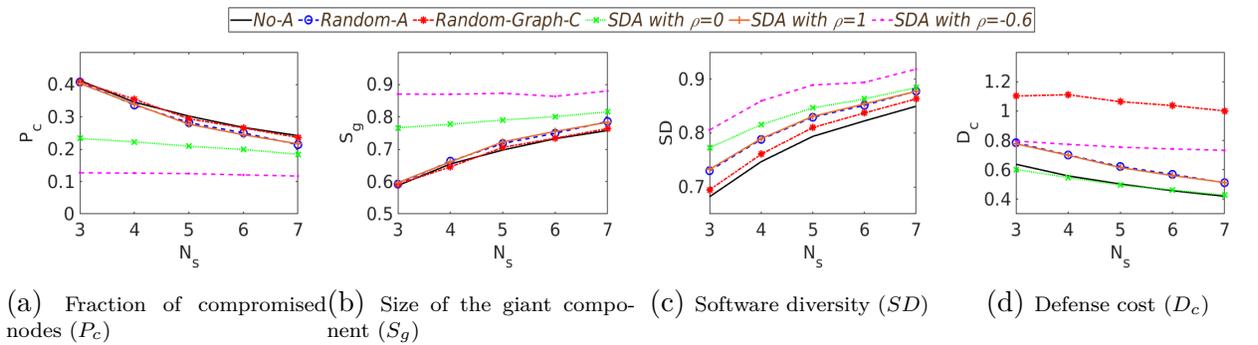


Figure A.6: Effect of the number of software packages ( $N_s$ ) under a random network.

### Effect of the Number of Software Packages ( $N_s$ ) Under a Random Network

Fig. A.6 shows how a different number of software packages ( $N_s$ ) available impacts the performance in terms of the metrics in Section 3.6.1 under the random network model.

Similar to the previous results displayed in Figs. A.4 and A.5, the degree of software diversity is well aligned with that of the size of the giant component. But, noticeably, SDA with optimal  $\rho = -0.6$  performs the best with high resiliency even under a very small  $N_s$ . In fact, this scheme shows steady performance in terms of all the metrics except software diversity ( $SD$ ) across the range of  $N_s$  considered in this work. This is due to the low network density and our intelligent edge adaptation. On the other hand, other five schemes exhibit improved performance when  $N_s$  is increased because the availability of more software packages can significantly contribute to increasing the degree of software diversity of each node. Unlike the previous results shown in Figs. A.4 (d) and A.5 (d), Random-Graph-A attains maximum defense cost when  $N_s = 4$ . This is due to two conflicting related factors: On the one hand, a node has more options to choose with more software packages available, i.e., a much lower chance to choose the same software package as the one the node currently has, thereby increasing the likelihood of shuffling and the shuffling cost. On the other hand, the network becomes more secure when more software packages are available, which results in lower IDS cost.

## A.4 Identification of Optimal $k$ under a Random Network

Fig. A.7 shows the sensitivity analysis when varying  $k$  used in the software diversity metric of the SDA with  $l = 1$  and  $\rho = -0.6$ . We observe that both  $S_g$  and  $P_c$  are not sensitive when  $k$  varies from 1 to 5. Thus, we use  $k = 1$  in our experiments to minimize computational complexity.

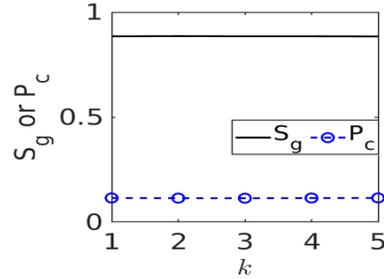


Figure A.7: Effect of varying  $k$  (a maximum hop distance) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ) under a random network.

## A.5 Identification of Optimal $l$ under a Random Network

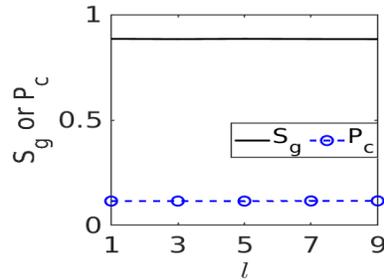


Figure A.8: Effect of varying  $l$  (a maximum number of attack paths) on network connectivity ( $S_g$ ) and security vulnerability ( $P_c$ ) under a random network.

Fig. A.8 shows the sensitivity analysis under varying  $l$  used in estimating the software diversity metric of the SDA when  $k = 1$  and  $\rho = -0.6$ . We observe that both  $S_g$  and  $P_c$  are not sensitive to varying  $l$  from 1 to 9. Thus, we set  $l = 1$  in our experiments to minimize computational complexity.

**Algorithm 10** Epidemic Attacks

---

```

1: Input:
2:  $\mathbf{A} \leftarrow$  an adjacency matrix
3:  $\mathbf{node} \leftarrow$  nodes' attributes defined in Eq. (3.2)
4:  $\sigma_i \leftarrow$  attacker  $i$ ' vector of exploitable software packages
5:  $N_s \leftarrow$  a number of software packages available
6:  $\gamma \leftarrow$  an intrusion detection probability
7: procedure PERFORMEPIDEMICATTACKS( $\mathbf{A}$ ,  $\mathbf{node}$ ,  $\sigma_j$ ,  $N_s$ ,  $\gamma$ )
8:    $spreadDone \leftarrow 0$ 
9:   spread: a list with length  $N$ , initialized at 0            $\triangleright$  To check if node  $i$  attempted to
   compromise its direct neighbors.
10:  while  $spreadDone == 0$  do
11:    for  $i := 1$  to  $N$  do
12:      if  $na_i > 0 \wedge nc_i > 0$  then            $\triangleright$  Check if  $i$  is an active attacker.
13:         $r_1 \leftarrow$  a random real number in  $[0, 1]$  based on uniform distribution
14:        if  $r_1 > \gamma \wedge \mathbf{spread}(i) < 2$  then
15:           $\mathbf{spread}(i) = \mathbf{spread}(i) + 1$ 
16:          for  $j := 1$  to  $N$  do
17:            if  $a_{ij} > 0 \wedge na_j > 0 \wedge nc_j == 0$  then            $\triangleright$  Check if  $j$  is susceptible.
18:              if  $\sigma_i$  includes  $s_j$  then            $\triangleright i$  knows  $s_j$ 's vulnerability.
19:                 $nc_j = 1$                                 $\triangleright j$  is compromised by  $i$ .
20:              else
21:                 $r_2 \leftarrow$  a random real number in  $[0, 1]$ 
22:                 $d \leftarrow sv_j$ 
23:                if  $r_2 < d$  then
24:                   $nc_j = 1$                                 $\triangleright j$  is compromised by  $i$ .
25:                   $\sigma_i(s_j) = 1$                         $\triangleright i$  learned  $s_j$ 's vulnerability.
26:                end if
27:              end if
28:            end if
29:          end for
30:        else
31:           $na_i = 0$             $\triangleright i$  is detected and deactivated for infecting behavior.
32:           $a_{ij} = 0, a_{ji} = 0$             $\triangleright$  IDS disconnects all edges adjacent to  $i$ .
33:        end if
34:      end if
35:    end for
36:    for  $k := 1$  to  $N$  do
37:      if  $na_i > 0 \wedge nc_i > 0 \wedge \mathbf{spread}(k) < 2$  then            $\triangleright$  Each node has two chances to
   compromise each of its direct neighbors.
38:         $spreadDone = 0$ 
39:        break
40:      else
41:         $spreadDone = 1$ 
42:      end if
43:    end for
44:  end while
45: end procedure

```

---