Universität Zürich^UZH

Department of Informatics

MERLINS: A Machine Learning-Driven Moving Target Defense (MTD) Framework for Secure 5G and Beyond 5G Networks

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Wissem Soussi
from
Sassuolo, Italy

approved in September 2025

at the request of
Prof. Dr. Burkhard Stiller
Dr. Gürkan Gür
Prof. Dr. Madhusanka Liyanage

Abstract

The transition from Physical Network Functions (PNF) to multi-cloud environments and service-based architectures in 5G and Beyond (5G/B5G) networks has led in highly distributed, virtualized, and interconnected edge services. While virtualization enables a flexible and scalable ecosystem, bridging the physical and virtual domains, it also significantly increases the size and complexity of these networks. This expansion inherently enlarges the attack surface, exposing networks to heightened risks of exploitation by malicious actors. Threats, such as intrusions, malware infections, and the formation of large-scale botnets become more prevalent. Thus, it is imperative to improve the security of such networks beyond conventional methods, using a layered in-depth security acting not only reactively but also proactively.

This PhD thesis addresses these emerging security challenges by introducing an additional security layer based on the Moving Target Defense (MTD) strategy. MTD leverages the inherent flexibility of modern networks to dynamically alter the attack surface of 5G/B5G, thereby complicating attackers' efforts to conduct reconnaissance and execute targeted attacks. The thesis proposes a novel approach, termed *MERLINS*, which generates and manages optimal MTD strategies for both proactive and reactive security. The *MERLINS* approach is built on a closed-loop, four-phase methodology, an MTD framework, and a suite of solutions that implement the framework. The methodology defines a cyclic process for selecting MTD operations based on near real-time monitoring and analysis of the network state, ensuring timely and context-aware enforcement in 5G networks and allowing future compatibility with B5G using the Network Virtualization Function (NFV) standard. The framework outlines a high-level architecture, detailing the components required to support the methodology's tasks, while the set of new solutions are implementations of such components providing *(a)* integration with 5G/B5G networks adhering to established specifications and standards, *(b)* the implementation of various MTD operations acting on different components of the network, *(c)* a cognitive component that optimizes MTD strategies by selecting the most effective operations based on the network's current state, and *(d)* a Federated Learning (FL) solution enabling multi-tenant collaboration to optimize MTD strategies while preserving data confidentiality.

Practical experiments conducted on a deployed 5G testbed demonstrate the feasibility, performance, and limitations of the proposed solutions, as well as their effectiveness in mitigating a range of security threats. The results observed in the experiments revealed that *1)* IP and port shuffling effectively increase the difficulty for attackers to identify and target services, *2)* stateless service re-instantiation and live migration is a powerful proactive security operation with limited overhead, mitigating Advanced Persistent Threats (APT) such as undetected malware infections and backdoors, *3)* the choice of communication protocol impacts MTD performance, with QUIC-based protocols reducing downtime significantly compared to TCP in traffic redirection scenarios, *4)* multi-objective deep Reinforcement Learning (deep-RL), despite being less commonly used, outperforms traditional single-objective deep-RL in optimizations requiring a balance between conflicting objectives, and *5)* accelerates the optimization of MTD strategies and potentially enhances performances while maintaining the confidentiality of participant data and models.

Collectively, this research advances the state of the art in securing 5G/B5G networks by providing a comprehensive, adaptive, and proactive MTD-based security framework. The proposed solutions not only address current security challenges but also pave the way to new research directions for resilient and secure future network architectures.

Kurzfassung

Der Übergang von physikalische Netzwerkfunktionen (PNF) zu Multi-Cloud-Umgebungen und dienstbasierten Architekturen in 5G und darüber hinausgehenden (5G/B5G) Netzwerken hat zu hochgradig verteilten, virtualisierten und miteinander verbundenen Edge-Services geführt. Während die Virtualisierung ein flexibles und skalierbares Ökosystem ermöglicht, das die physischen und virtuellen Domänen überbrückt, erhöht sie auch die Größe und Komplexität dieser Netzwerke erheblich. Diese Erweiterung vergrößern die Angriffsfläche und setzen die Netzwerke einem erhöhten Risiko der Ausnutzung durch böswillige Akteure aus. Bedrohungen wie Einbrüche, Malware-Infektionen und die Bildung groß angelegter Botnets werden immer häufiger. Daher ist es zwingend erforderlich, die Sicherheit solcher Netze über die herkömmlichen Methoden hinaus zu verbessern, indem eine mehrschichtige, tiefgreifende Sicherheit verwendet wird, die nicht nur reaktiv, sondern auch proaktiv wirkt.

Diese Doktorarbeit befasst sich mit neuen Sicherheitsherausforderungen, indem sie eine zusätzliche Sicherheitsebene einführt, die auf der Strategie der Moving Target Defense (MTD) basiert. MTD nutzt die inhärente Flexibilität moderner Netzwerke, um die Angriffsfläche von 5G/B5G dynamisch zu verändern und so die Bemühungen von Angreifern zu erschweren, Erkundungen durchzuführen und gezielte Angriffe auszuführen. In dieser Arbeit wird ein neuartiger Ansatz, names *MERLINS*, vorgeschlagen, der optimale MTD-Strategien sowohl für proaktive als auch reaktive Sicherheit generiert und verwaltet. Der *MERLINS*-Ansatz basiert auf einer geschlossenen, vierphasigen Methodik, einem MTD-Rahmenwerk und einer Reihe von Lösungen, die das Rahmenwerk implementieren. Die Methodik definiert einen zyklischen Prozess für die Auswahl von MTD-Operationen auf der Grundlage einer nahezu in Echtzeit erfolgenden Überwachung und Analyse des Netzwerkstatus, um eine zeitnahe und kontextbezogene Durchsetzung in 5G/B5G Netzwerken zu gewährleisten. Das Framework skizziert eine High-Level-Architektur und beschreibt die Komponenten, die zur Unterstützung der Methodik benötigt werden. Die neuen Lösungen sind Implementierungen dieser Komponenten und bieten: *(a)* Integration mit 5G- und darüber Netzen unter Einhaltung etablierter

Spezifikationen und Standards, *(b)* die Implementierung verschiedener MTD-Operationen, die auf verschiedene Komponenten des Netzwerks einwirken, *(c)* eine kognitive Komponente, die MTD-Strategien optimiert, indem sie die effektivsten Operationen auf der Grundlage des aktuellen Zustands des Netzwerks auswählt, und *(d)* eine Lösung für föderiertes Lernen (FL), die eine mandantenübergreifende Zusammenarbeit ermöglicht, um MTD-Strategien unter Wahrung der Vertraulichkeit der Daten zu optimieren. Praktische Experimente, die auf einem 5G-Testbed durchgeführt wurden, demonstrieren die Machbarkeit, Leistung und Grenzen der vorgesch-lagenen Lösungen sowie ihre Effektivität bei der Entschärfung einer Reihe von Sicherheitsbedrohungen. Die in den Experimenten beobachteten Ergebnisse zeigen, dass *1)* IP- und Port-Shuffling die Schwierigkeit für Angreifer, Dienste zu identifizieren und anzugreifen, effektiv erhöht, *2)* zustandslose Neuinstantiierung von Diensten und Live-Migration eine leistungsstarke proaktive Sicherheitsmaßnahme mit begrenztem Overhead ist, die fortgeschrittene anhaltende Bedrohungen (APTs) wie unentdeckte Malware-Infektionen und Backdoors abschwächt, *3)* die Wahl des Kommunikationsprotokolls die MTD- Leistung beeinflusst. Die Wahl des Kommunikationsprotokolls wirkt sich auf die MTD-Leistung aus, wobei QUIC-basierte Protokolle die Ausfallzeiten im Vergleich zu TCP in Verkehrsumleitungsszenarien erheblich reduzieren. *4)* Multi-Objective Deep Reinforcement Learning (deep-RL) wird zwar seltener eingesetzt, ist aber bei Optimierungen, die ein Gleichgewicht zwischen widersprüchlichen Zielen erfordern, besser als herkömmliches Deep-RL mit nur einem Ziel, und *5)* beschleunigt die Optimierung von MTD-Strategien und verbessert möglicherweise deren Leistung, während die Vertraulichkeit der Teilnehmerdaten und -modelle gewahrt bleibt. In der Gesamtheit betrachtet bringt diese Forschung den Stand der Technik bei der Sicherung von 5G/B5G Netzwerken voran, indem sie einen umfassenden, adaptiven und proaktiven MTD-basierten Sicherheitsrahmen bereitstellt. Die vorgeschlagenen Lösungen adressieren nicht nur aktuelle Sicherheitsherausforderungen, sondern ebnen auch den Weg für neue Forschungsrichtungen in belastbaren und sicheren zukünftigen Netzwerkarchitekturen.

Acknowledgments

With profound gratitude, I acknowledge the divine guide that has sustained me throughout my life, including this incredible journey of my doctoral studies. I am grateful and consider myself truly fortunate for having pursued research in a field that ignites my passion, surrounded by individuals who inspired me and who I got to put in high esteem.

First and foremost, this dissertation is dedicated to my family, whose long-term support was the cornerstone of my success. To my dear father, Houcine, who has been an inspirational role model in character and conduct for as long as I can remember. To my dear mother, Arbia, whose unconditional love and care have nourished me at every turn in a way that I would never be able to repay. I extend my most heartfelt appreciation to my dear wife, Sabrine, and my soul, my daughter Ilef, for their constant love and understanding. They have endured countless late nights and weeks of absence of mine due to work and travels for conferences and project meetings. Their support has been invaluable to my personal and professional growth throughout this journey.

I am deeply grateful to my supervisors, Dr. Gürkan Gür from ZHAW and Prof. Dr. Burkhard Stiller from UZH, for continuously supporting me during the various stages of my Ph.D. Working primarily at ZHAW, I have learned a lot from Gürkan, not only in terms of technical knowledge but also on how to be an excellent supervisor and project leader, leading by example with his actions, wisdom, and support on a day-to-day basis. Likewise, Burkhard's guidance, experience, and wisdom have been indispensable to my success. I am very fortunate to have had such outstanding supervisors over these four years.

I am grateful to colleagues, external collaborators, and EU project partners I had the pleasure to work with during my Ph.D., particularly Gökcan Cantali, Maria Christopoulous, Themis Anagnostopoulou, George Xilouris, Hữu Nghĩa Nguyễn, Beytullah Yiğit, and Ariane Trammell. I also want to thank the bachelor and master students at ZHAW whom I had the pleasure of cosupervising and collaborating with, especially the students Anthony Mamaril, Heeb Zeno, Michael Meier Azhari, Nicholas Mayones, Pascal Kunz, and Rinchen Kolodziejczyk.

Finally, I would like to express my deep appreciation to all my colleagues from both research groups, the ISE group at ZHAW and the CSG at UZH. The shared coffee breaks and camaraderie made my experience more enriching and enjoyable, and I am grateful for that.

Contents

1 Introduction

Telecommunication networks, such as 5G/B5G, provide various services that enable connected and digital applications in anytime-anywhere settings. These services, ranging from leisure human activities like live video streaming to mission-critical applications, such as smart grids and industrial Internet of Things (IoT), operate in a scalable, cost-efficient, and flexible manner via new technologies. Network slicing is a key technology in this regard, introduced with 5G telecommunication networks to create isolated, multi-tenant, and multi-domain networks on a single infrastructure, featuring an independent data plane, control plane, and management plane [1]. Each network slice handles these three planes differently, considering their different service level requirements (*e.g.*, communication latency, bandwidth, and power consumption) [2].

Concurrently, Virtual Network Operators (VNO) are deploying Multi-Access Edge Computing (MEC) [4], increasing the size and complexity of the infrastructure with intermediate data centers, cloud platforms, and edge nodes at different access points. This architectural setting allows mobile applications to utilize cloud computing platforms located in intermediate data centers and edge networks (such as those situated at cellular base stations) that are in close proximity to end-users. With this edge-to-cloud continuum, communication overload is greatly reduced, and end-user Quality of

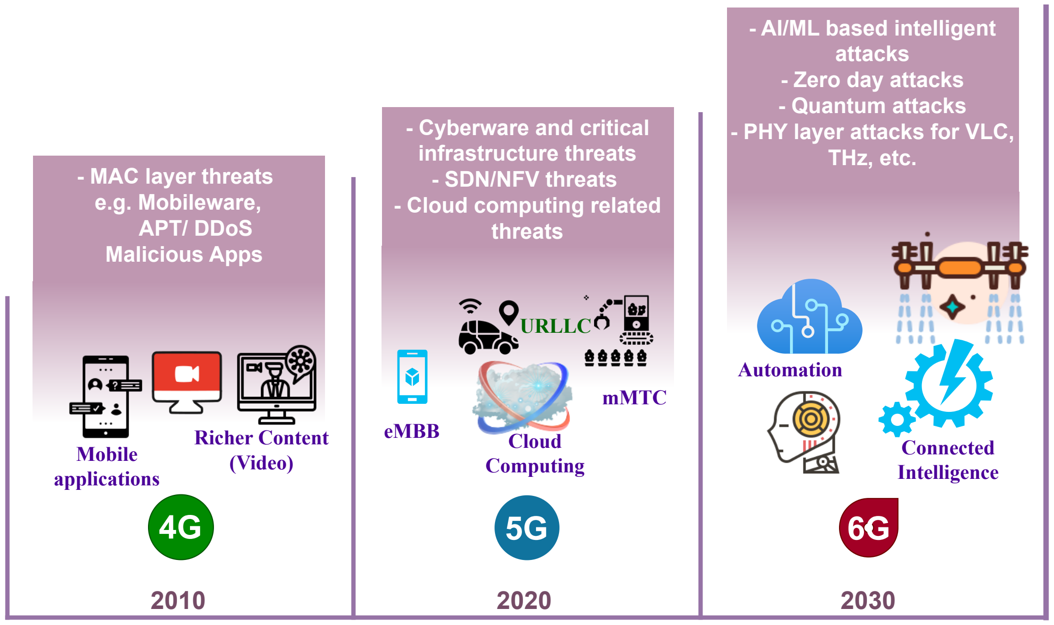


Figure 1.1: Threat landscape evolution with the evolving of telecommunication networks, based on [3].

Experience (QoE) is enhanced. However, the larger infrastructure that is eventually created also presents new challenges as it increases the network complexity. Nonetheless, network slicing is still possible with Software-Defined Networking (SDN) and Network Function Virtualization (NFV) in this complex setting [5]. These technologies improve the scalability and flexibility of network services by dynamic instantiation, deletion, and chaining of virtual resources and Virtual Network Functions (VNFs) encapsulated in virtual machines (VM) and/or container runtimes.

B5G systems will drastically extend network capabilities beyond this promise and realize a revolutionary set of services with unprecedented quality and reliability levels, such as Further-enhanced Mobile Broadband (FeMMB), Enhanced Ultra-Reliable Low-Latency Communication (ERLLC/eURLLC), and ultra-massive Machine Type Communication (umMTC). They are envisioned to support a peak data rate of 1 Tbps, an area traffic capacity of 1 Tbps/m^2, and microsecond-level latency. Expected to be deployed in the 2030s, they will unlock advanced traffic demands for more flexible, data-hungry, and tactile applications such as holographic communications, pervasive Virtual Reality (VR) and Augmented Reality (AR) applications, tactile Internet, smart massive autonomous systems like drone delivery service, 3D networking, and fully-automated network orchestration [6].

However, this environment's heterogeneity and increased large-scale nature also result in a greater attack surface of communication systems, *i.e.*, the set of network elements (hardware and software) that can be accessed by authorized or unauthorized entities to exploit them for cyberattacks.

Figure 1.1 depicts the increasing number and diversity of threats with the successive telco generations, based on [3]. Protecting a larger attack surface is more challenging as attacks have a greater range of targets and can propagate and affect multiple network domains and slices. Malicious actors, internal or external to a telecommunication network, can perform various attacks, such as reconnaissance/fingerprinting, in order to gather knowledge on the targeted system, followed by more disruptive attacks such as Denial-of-Service (DoS), Distributed DoS (DDoS), spoofing, intrusion, malware infection, or Man-in-the-Middle (MitM). Moreover, some attacks can propagate within the network through a single compromised device, leading to widespread security incidents, as observed with IoT botnets [7]. This situation is expected to become more critical with B5G networks, where Quality-of-Service (QoS) requirements are even more stringent, and services are more critical towards security, as in autonomous vehicle networks, Industry 4.0, and tactile remote surgery [8].

In-depth defense, composed of multiple layers of security, must be in place to secure such softwarized networks and services along the edge-to-cloud continuum. It becomes imperative to establish an automated system that monitors, detects, and adaptively mitigates threats by minimizing the attack surface, performing optimal and dynamic decisions, and considering a multitude of factors in the current state of the 5G network. These requirements extend beyond 5G networks and are also applicable in the context of future networks currently being discussed in academia and the research community, namely B5G, 6G, or NextG networks [3].

Moving Target Defense (MTD) [9] is a promising paradigm that extends network management automation to security, providing proactive and reactive protection of virtualized network assets. The US National Institute of Standards and Technology (NIST) defines MTD as "the concept of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity, and increase the costs of their probing and attack efforts". MTD leverages the new network's flexibility and heterogeneity properties to its advantage by shifting the virtual resources in time and space. This can drastically reduce the action space of attackers in the time dimension, mitigate attacks, and guarantee the availability of protected services.

The usage of MTD would result in an additional defensive layer for in-depth security of future networks, complementing conventional security approaches, such as firewalls, authentication, security protocols, and encryption, further increasing the hardening of a networked system. Thus, MTD does not replace foundational security measures; rather, it addresses the exponential increase in the attack surface by introducing uncertainty and resilience. By continuously shifting the system's configuration, MTD operations invalidate an attacker's reconnaissance and data collection efforts, forcing them to continually reassess their approach. This raises the cost for attackers and reduces their Attack Success Probability (ASP).

Nonetheless, the enforcement of MTD operations can also impact the network performance and come with additional operational costs and energy consumption. Therefore, smart and dynamic control of MTD following a cognitive paradigm (i.e., a closed loop of *observe, orient, plan, decide, act,* and *learn* tasks) considering security requirements, security gains, overhead, and feasibility, is crucial. This is attainable through the utilization of game-theoretical models, Artificial Intelligence (AI), and Machine Learning (ML) to enable and optimize the autonomous protection of virtualized network resources and services.

1.1 Problem statement

Despite the promise of MTD and AI/ML paradigms for network security, the goal of learning-based optimization of autonomous and proactive security, considering the heterogeneity of services, infrastructure, and operational requirements in Telco Cloud environments, is still to be reached. Telco Clouds are telecommunication networks provided by a VNO and complying with the NFV and MEC architectures [10].

As of today, the deficit towards reaching this goal and the key challenging issue to be addressed by this Thesis is the following:

- To date and to the best of the author's knowledge, there is no research work for designing and operating ML-based optimized MTD for large-scale and realistic telecommunication network scenarios, with a forward-looking perspective of driving technologies for B5G networks, such as cloudification and AI-native technologies. While AI/ML-driven MTD promises self-regulated autonomous operations and mitigation of cyber threats, its fundamental advantages in attack prevention, mitigation, and responsiveness have not been identified in various security scenarios.

This comes with subsequent relevant challenges summarized below:

- The integration and utilization of full-stack, full-spatiotemporal action space (e.g., VMs and containers live migration, execution environment diversification, and shuffling) in virtualized 5G infrastructure for MTD is still scarcely explored. Moreover, the cohabitation of proactive modes (*i.e.,* to change the attack surface before the attack) and reactive modes (*i.e.,* to change the attack surface/mitigate during the attack) supported by online network monitoring for situational awareness is still to be developed.

- Most of the work is evaluated only on theoretical game-theory models, and a few also apply it in PoCs, but with relatively simplistic scenarios and limited environments, *i.e.,* based on a static instantiation of a system model, which is used to develop cognitive security schemes and optimization mechanisms. However, more realistic 5G networks are highly dynamic and heterogeneous, resulting in fast variations of the virtual service and network topology over time.

- The strategic placement and movement of network resources aim to improve security, optimize performance, and ensure efficient resource consumption. These objectives do not overlap, and conflicts may arise when performing MTD operations, favoring one goal at the expense of the other. For instance, moving a VNF from a remote Virtual Infrastructure Manager (VIM) to an edge node's VIM for communication optimization may be a poor security choice, as an attacker can easily predict such an action. A purely random placement, on the other hand, improves security by reducing its predictability, but can hinder the network's performance and the QoS of the moved service.

- Within a multi-tenant, multi-cloud communication environment, Telco Cloud services may be provisioned across infrastructure belonging to multiple operators and VNOs. In such scenarios, autonomous decision-making systems managing resources for one tenant could directly influence the resources allocated to other tenants, necessitating either a hierarchical, centralized architecture or a peer-to-peer tenant structure with cooperating autonomous decision-making systems. While leveraging the diverse network experiences and threat intelligence from various tenants offers potential for improved MTD optimization, both approaches present challenges, such as ensuring data confidentiality, establishing trust, ensuring fairness in resource allocation, and securing data aggregation.

1.2 Research Questions

This PhD thesis addresses the challenges outlined in Section 1.1 and proposes a novel approach that enables autonomous, proactive, and reactive cognitive cybersecurity schemes, leveraging MTD for optimized and robust network slice protection in 5G (and prospective B5G) environments. Within this context, four research questions were investigated during the development of this PhD thesis:

RQ1 - MTD Action Selection and Security Impact: Which MTD actions can be taken on a 5G network and against which attack scenarios, considering the security properties they offer?

This RQ1 involves the literature review, evaluation, and comparison of the MTD operations that can be performed on the different layers of a Telco Cloud network, their operational cost, and the substantial improvements for 5G/B5G security management in identified security scenarios.

RQ2 - MTD Efficiency Optimization: How can we minimize the network and resource overhead associated with MTD operations, inducing increased energy consumption, without affecting its security properties?

This RQ2 involves the literature review of emerging AI/ML methods and the adoption/design of an appropriate ML-driven control system that learns and applies proactive and reactive MTD strategies in the right place and at the right moment, improving efficiency (in terms of resource consumption, making MTD also more sustainable), and effectiveness (in terms of security gains).

RQ3 - Formal Modeling for Network Assessment: Which modeling approaches are most suitable for representing communication networks to enable near real-time monitoring and security assessments for MTD systems?

This RQ3 focuses on designing appropriate architecture(s)/models that enable ML-driven MTD as needed in RQ2. This involves developing mathematical models that accurately describe the considered game theory problem for the MTD control system and for virtualized service/resource protection, possibly identifying practical constraints and system requirements.

RQ4 - Multi-tenant distributed MTD solution: What are the ways to distribute the control system of the cognitive MTD solution in a multi-tenant peer-to-peer environment?

This RQ4 focuses on investigating theoretically and experimentally the cognitive and autonomous MTD approach in a distributed peer-to-peer manner among multiple decision systems held by different tenants of a Telco Cloud. This includes using or extending RQ2 results for such problems and investigating theoretically and experimentally distributed ML approaches such as Federated Learning (FL).

1.3 Thesis Contributions

To address these challenges, this Ph.D. thesis designs, implements, and studies the usage of a smart cybersecurity framework focusing on MTD to secure 5G/B5G Telco Clouds. Inherent to 5G, security evolves towards the use of virtualization, and MTD becomes a promising protection scheme that leverages the cloud-native nature of novel telecommunication networks. The Ph.D. thesis proposes a state-of-the-art solution in the area of smart/cognitive cybersecurity systems. It performs threat and security analysis of the relevant 5G subsystems and produces a resource-optimal MTD-based defense technique. AI/ML is used to learn new MTD strategies that optimize the cost/benefit tradeoff of its operations. The contributions of this PhD thesis are summarized as follows:

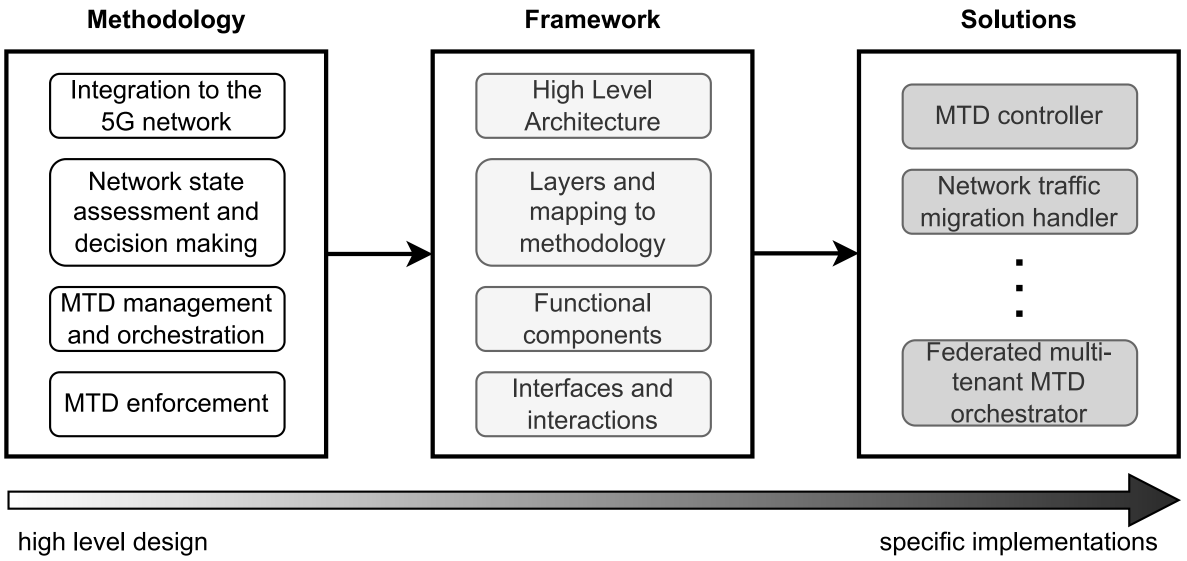

Methodology
Framework
Solutions
Integration to the 5G network
Network state assessment and decision making
MTD management and orchestration
MTD enforcement
High Level Architecture
Layers and mapping to methodology
Functional components
Interfaces and interactions
MTD controller
Network traffic migration handler
Federated multi-tenant MTD orchestrator
high level design
specific implementations

Figure 1.2: Overview of Contributions of This PhD thesis.

- **A novel methodology and framework: *MERLINS*.** The main contribution of this PhD thesis is the methodology designed, resulting in the creation of a novel architectural framework for a cognitive MTD orchestrator that integrates with the management and orchestration (MANO) component of the NFV architecture in 5G/B5G networks. *MERLINS* is designed by adhering to the ETSI ZSM reference architecture, defining a closed-loop control that achieves a self-configuring, self-monitoring, self-healing, and self-optimizing 5G/B5G system. Different solutions are then designed and implemented to cover the tasks defined in the framework, further providing the following contributions.

- **An MTD controller integrated in a 5G network: *MOTDEC*.** This is responsible for enforcing the MTD actions on the 5G network in coordination with the 5G MANO and network slice manager.

- **A live network traffic migration handler: *TopoFuzzer*.** This solution is designed to handle the redirection of session-based communications, preventing disruptions to ongoing sessions during MTD actions such as IP shuffling and VNF migrations.

- **A containerized CNF's live migration for MTD: *ContMTD*.** This is a lightweight proactive MTD solution that leverages SoTa container live migration to dynamically relocate stateful CNFs (*i.e.,* 5G network functions) in cloud-native environments. By periodically migrating containers across hosts, ContMTD disrupts attacker reconnaissance and reduces vulnerability windows without service downtime.

- **A deep-RL optimizer of MTD strategies: *OptSFC*.** This is the cognitive component designed to optimize the MTD strategy in a multi-objective setup that considers different optimization factors in the problem.

- **A federated multi-tenant MTD orchestrator: *MTDFed*.** This enables a multi-tenant orchestration of MTD actions, where multiple operators run their own decision systems while also cooperating to optimize MTD strategies.

Ultimately, these contributions bring MTD proactive and reactive protection into 5G/B5G networks, providing a novel security-aware, cost-aware, and QoS-aware intelligent management and orchestration of Telco Cloud assets.

1.4 Thesis Outline

This PhD thesis is organized into six chapters, depicted in Figure 1.3. **Chapter 2** establishes the theoretical foundations and key concepts of Telco Cloud networks and MTD required for a complete understanding of this thesis. This includes the analysis of current telecommunication networks, *i.e.,* 5G, the architectural standards and technologies behind them, *i.e.,* ETSI NFV and ETSI ZSM, the MTD concept, service live migrations, and the decision optimization using ML, specifically, deep Reinforcement Learning (deep-RL) and Federated Learning (FL).

Next, **Chapter 3** delves into the existing literature regarding MTD. For this, a comprehensive review from a telecommunication networks perspective is conducted to map theoretical models, technologies, and systems that address the challenges of delivering cognitive and optimized MTD policies to reduce operational costs, network overhead, and impact on a VNF's QoS, while maximizing security gains.

Then, **Chapter 4** introduces the core contribution of the thesis: the *MERLINS* approach. It elaborates on each element of *MERLINS*, proposing a closed-loop methodology for continuous network monitoring and ML-based MTD policy enforcement. The chapter further details the architectural framework that operationalizes the defined methodology and proposes a set of solutions to fulfill the

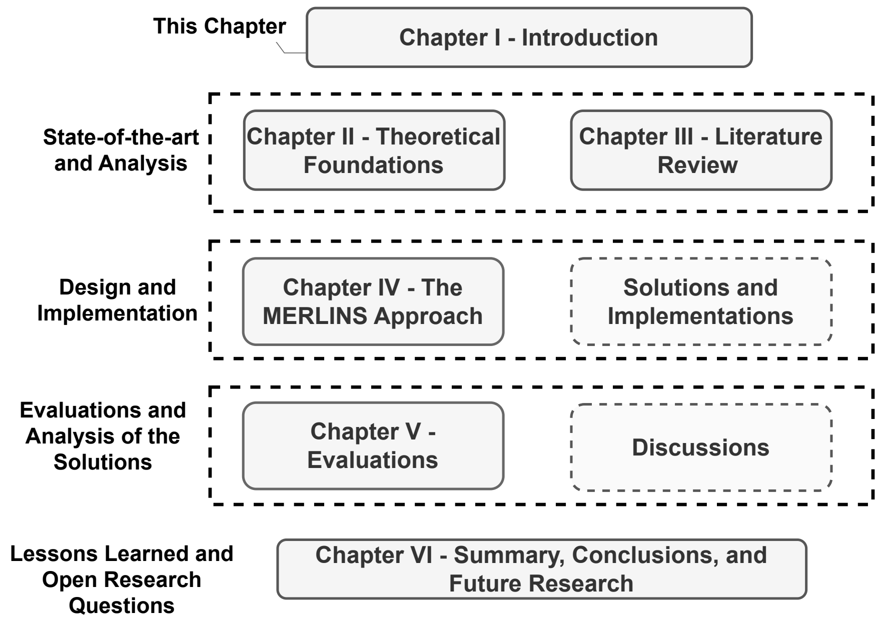

This Chapter
Chapter I - Introduction
State-of-the-art and Analysis
Chapter II - Theoretical Foundations
Chapter III - Literature Review
Design and Implementation
Chapter IV - The MERLINS Approach
Solutions and Implementations
Evaluations and Analysis of the Solutions
Chapter V - Evaluations
Discussions
Lessons Learned and Open Research Questions
Chapter VI - Summary, Conclusions, and Future Research

Figure 1.3: Organization of This PhD Thesis.

framework's requirements. Chapter 4 also details the implementation specifics of the proposed solutions within *MERLINS*. This includes discussions on technology choices, implementation steps, and the technical decisions made during the development process.

After that, **Chapter 5** presents extensive evaluations of the proposed solutions, incorporating both quantitative and qualitative metrics. A dedicated case study using a 5G testbed further demonstrates the feasibility and benefits of the *MERLINS* methodology, framework, and solutions. This chapter concludes with a discussion of all findings, along with any challenges and limitations encountered during the development and evaluation process.

Chapter 6 concludes this PhD thesis by summarizing the key findings and revisiting the research questions outlined in Section 1.2. Finally, it also identifies and discusses some suggestions for future research directions in this area.

2 Theoretical Foundations

This chapter establishes the theoretical foundations, core principles, and contextual knowledge necessary to frame and substantiate the research contributions of this PhD thesis. Section 2.1 introduces the terminologies and concepts related to modern telecommunication networks, including the technologies and architectures used in the current generation (5G) and beyond (B5G), such as Telco-Cloud architectures, Multi-access Edge Computing (MEC), and Fog Computing. These technologies are critical in enabling the MTD operations implemented in this thesis. Following this, Section 2.2 provides a detailed examination of the fundamentals of MTD, its categories, and optimization challenges. Section 2.3 then explores the background of live service migration, employed in *MERLINS* as an MTD operation. Finally, Section 2.4 presents the basics of deep Reinforcement Learning (deep-RL), which is relevant to address the MTD optimization challenges tackled in this thesis.

It is appropriate to highlight that the glossary appendix (*cf.* A.2.2) contains the terminology used in this chapter and throughout the thesis.

2.1 5G/B5G Telecommunication Networks

To assess the environment in which this thesis develops its security orchestration framework, *i.e.* 5G and B5G networks, it is essential to first grasp the structure and components of these environments. This understanding lays the groundwork for identifying the network's attack surface, the associated threat and risk scenarios, and the inherent properties and characteristics that can be leveraged to mitigate these threats. For such reason, this section describes the infrastructure of 5G networks (Section 2.1.1), the relevant modern architectures such as MEC and Fog computing based on the 'cloudification' of telecom networks resulting in the new concept of Telco-Cloud (Section 2.1.4), the standardization perspective in the ETSI standard, including the major NFV standards (Section 2.1.2), and finally the management and orchestration of the virtualized resources following the ZSM framework (Section 2.1.3).

2.1.1 5G Infrastructure

The Fifth Generation of Mobile Telephony, or 5G, is the current state-of-the-art telecommunication system defined by 3GPP from Release 15, in 2018, with ongoing updates up to Release 18 [11]. 3GPP establishes not just the communication method between phones and towers (air interface), but also the protocols that govern how mobile networks function: from managing calls and user connections to delivering services and ensuring seamless switching between networks (*i.e.*, interoperability). This standardized approach allows mobile networks built by different vendors and service providers/operators to work together flawlessly. Nearly all operators offer 3GPP systems today, with Long-Term Evolution (LTE, or 4G) deployed by over 800 operators, and 5G already available from 153 operators in 71 countries, as per GSAcom in March 2024 [12].

Unlike previous generations designed primarily for consumers, 5G extends its reach to various industries and applications demanding real-time responsiveness, such as autonomous driving in the Internet of Vehicles (IoV) [13] and the Tactile Internet (TI) [14]. To deliver these advanced functionalities and enhance user experience, 5G utilizes a toolbox of innovative technologies such as NFV, SDN, MEC, and Satellite Communications. Starting from 5G and going to B5G, telecommunication networks improve the following services:

- *Enhanced Mobile Broadband (eMMB)*: this service aims to provide higher throughput (20Gbps peak data rate) and improved coverage with millimeter-wave (mmWave) frequencies and advanced Multiple Input Multiple Output (MIMO) technology according to 3GPP.

- *Ultra-Reliable Low-Latency Communication (RLLC/URLLC)*: this service focuses on providing higher reliability (expected at 99,9999%) and lower latency (1 to 20 ms).

- *Massive Machine Type Communication (mMTC)*: this service supports a high density of connected devices in massive IoT ecosystems, ensuring efficient and reliable connectivity.

- *Flexible Network Operations*: this provides a set of services offered starting from 5G and continuing with B5G networks, enabling flexible management and orchestration of networking resources. This includes network slicing, NFV, and MEC, further developed in this section.

These services characterize 5G networks as specified by the International Telecommunication Union (ITU) in the International Mobile Telecommunications (IMT) standards, which define 3G in IMT-2000, 4G in IMT-Advanced, 5G in IMT-2020, and is now defining future 6G networks in the IMT-2030 standard, adding to the three 5G services (*i.e.*, eMMB, URLLC, and mMTC) three novel usage scenarios: Integrated Sensing and Communication, AI and Communication, and Ubiquitous Connectivity [15].

The 5G system incorporates similar elements to previous generations, consisting of User Equipment (UE), which includes a Mobile Station and a USIM, a Radio Access Network (RAN), and a Core Network, in this case, a 5G core (5GC). The RAN's central component is the gNB (gNodeB), which is the base station and serves as the radio transmitter. The radio interface is termed "NR-Uu", from New Radio (NR). 5G NR leverages mmWave and MIMO technologies to offer eMMB and operates over multiple frequency ranges, from 400 MHz to 100 GHz, with licensed bands between 600 MHz and 39 GHz, enabled in specific countries and regions. The frequencies enable analog bands for satellite systems, but the terrestrial frequency ranges are identified as *1.* up to 1GHz for rural areas, *2.* 1 to 6 GHz for urban and suburban areas, and *3.* above 6 GHz for dense urban environments.

The 5GC is schematically represented in Figure 2.1, where the User Plane Function (UPF) manages user data and the Access and Mobility management Function (AMF) handles the UE and the RAN. Additional 5GC entities are also included. The interface between the access and core networks is referred to as "NG", composed of several interfaces, primarily N2 and N3.

The 5GC architecture utilizes a Service-Based Architecture (SBA) framework, defining architecture elements as "Network Functions" (NFs) rather than traditional Network Entities. Each NF provides services to other authorized NFs and consumers through a common interface framework, promoting modularity and reusability. NFs are split between the essential NFs of the *signalling/-control plane* and the NFs of the *user plane*. The signalling plane only comprises elements related to managing the 5G connectivity, as the AMF, while the user plane comprises elements involved in the

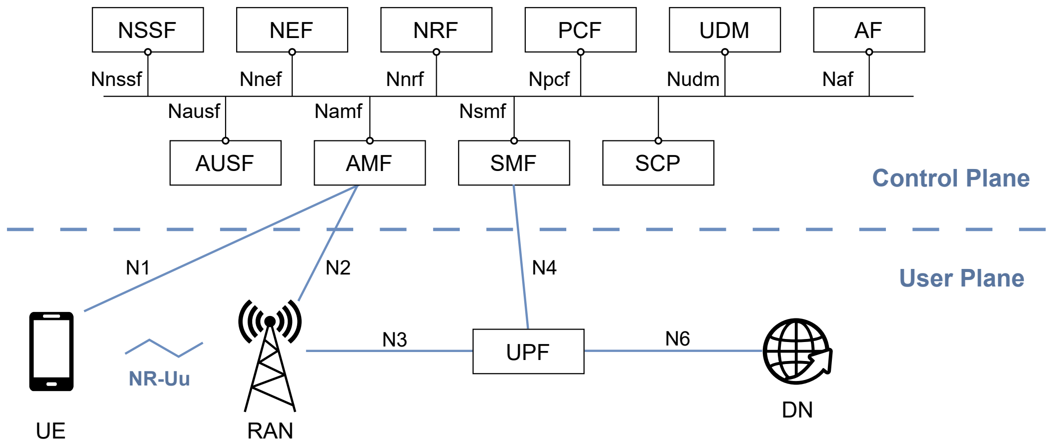
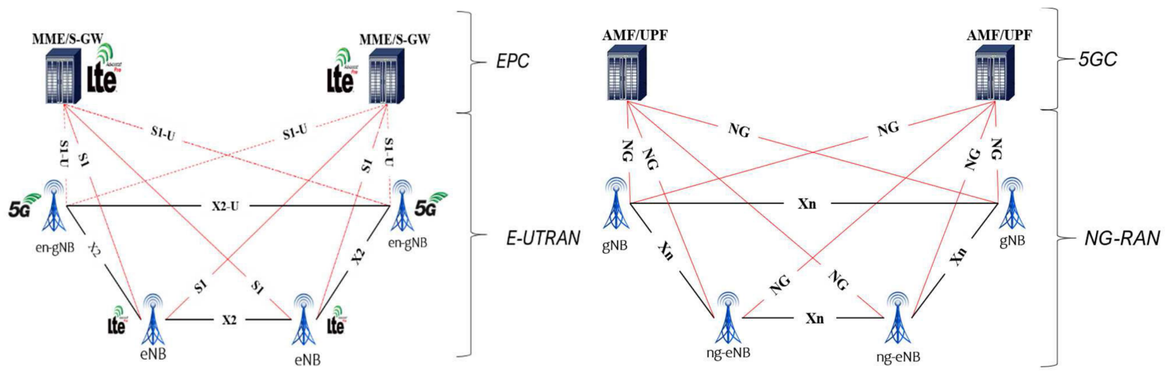


Figure 2.1: 5G core architecture with N*i* interfaces to the UE, RAN, and DN components.

transport of user data. The UPF is an essential NF that manages all NFs in the user plane. The 5G comprises various essential NFs, among which are: The Application Function (AF), which controls the applications (and may also be involved in the user plane); The Service Communication Proxy (SCP), which dynamically scales and manages communication services in the network; The Session Management Function (SMF), which manages calls and sessions and coordinates with the UPF; The Unified Data Management (UDM), functionally analogous to 3G and 4G's Home Subscriber Server (HSS) and 2G's Home Location Register (HLR); The Policy Control Function (PCF), which ensures that user data traffic does not exceed the negotiated bearer capacities; The Network Repository Function (NRF), which manages other NFs by providing support for registering, deregistering, and updating NF services; Security-related NFs: Network Exposure Function (NEF), Authentication Server Function (AUSF), Security Anchor Functionality (SEAF); and The Network Slice Selection Function (NSSF).

5G deployment offers two primary architectural options. The *Non-Stand Alone (NSA)* architecture integrates the 5G RAN and NR interface with the existing 4G LTE and EPC Core Network, allowing for the use of NR technology without replacing the existing network infrastructure. This configuration supports only 4G services but benefits from the advanced capabilities of 5G NR, such as lower latency. With the 4G E-UTRA and 5G NR dual Connectivity (EN-DC), the NSA setup connects the 5G NR base station (en-gNB) to the 4G LTE base station (eNB) via the X2 interface, which was enhanced in Release 15 to facilitate this connection.

The *Stand-Alone (SA)* architecture, on the other hand, connects the NR directly to the 5G Core Network (CN), enabling support for the full suite of 5G services. This configuration represents a complete 5G deployment, independent of 4G infrastructure. The NSA architecture has been viewed



(a) 5G NSA architecture. **(b)** 5G SA architecture.

Figure 2.2: Non-StandAlone (NSA) and StandAlone(SA) architectures for 5G networks, based on [16].

as a transitional step towards the full implementation of 5G, leveraging existing 4G infrastructure to provide enhanced services while paving the way for the more advanced capabilities of the SA architecture. Most of 5G networks today are NSA, such as the Swiss telecommunication infrastructure operated by Swisscom, as of April 2024 [17].

2.1.2 Network Function Virtualization

The ETSI Network Function Virtualization (NFV) architecture [10] is a standard specified by the European Telecommunications Standards Institute (ETSI). It introduces to legacy telecommunication networks the flexibility of could-native infrastructures, as well as the virtualization of network resources. Here, network transportation is software-defined, and traditional NFs are usually expensive hardware middleboxes, plugged and configured manually (*e.g.*, Broadband Remote Access Server (BRAS), Radio Network Controller (RNC), Carrier Grade NAT (CG-NAT), firewalls, and load balancers) become virtualized. They can be deployed in one or many virtual machines or, as in recent efforts, in more lightweight and flexible OS containers, naming them respectively virtual network functions (VNF) and cloud-native network functions (CNF). This significantly reduces the VNO's Capital and Operational Expenses (CAPEX/OPEX).

As depicted in Figure 2.3, the main components of the ETSI NFV architecture are:

- The NFV Infrastructure (NFVI): this groups all the computational resources, storage resources and network resources, at both the hardware and virtual level. This represents the cloud infrastructure of the architecture.

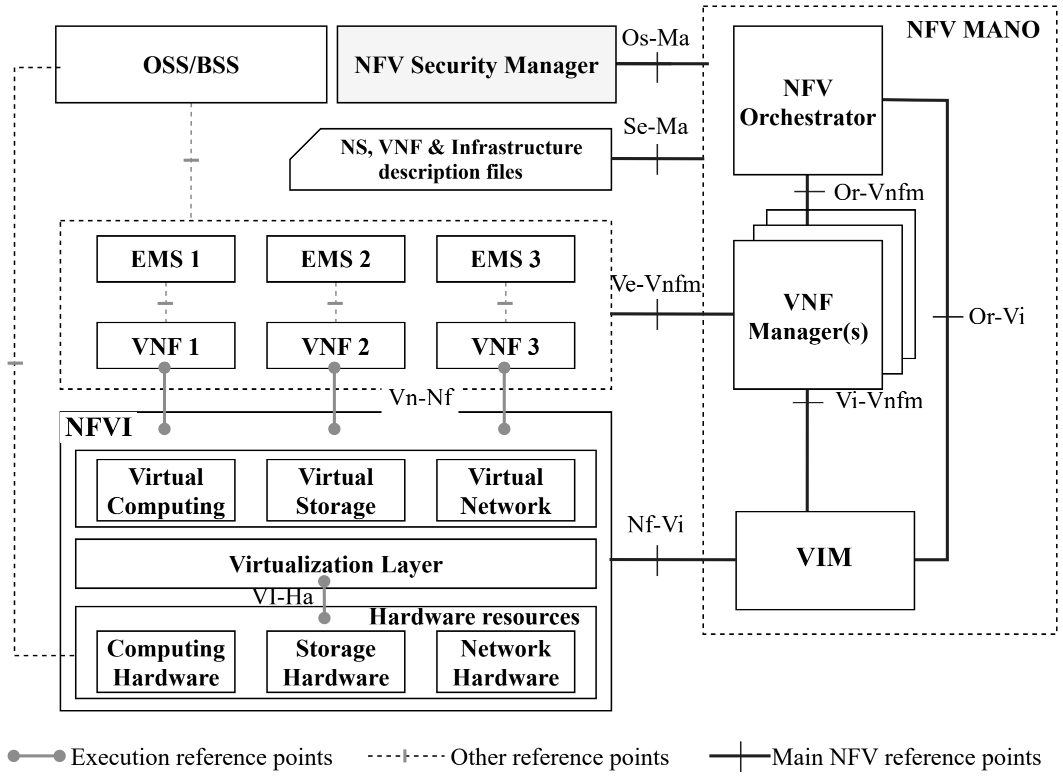


Figure 2.3: ETSI NFV architecture [10].

- The NFV resources: these are the singular network functions and services running in the Telco Cloud network. These resources could be VNFs or CNFs, depending on whether they are hosted in VMs or containers. A group of VNFs, interconnected with virtual links (VL), form what is defined as a network service (NS), where the set of VNFs/CNFs enable the provision of specific services, analogous to the microservices concept in cloud computing. Then NSs can be part of a virtual network/domain, namely a network slice (NSi). All these represent NFV network resources and are defined by a standardized file descriptor (*i.e.*, vnfd for VNFs and CNFs, nsd for NSs, and gst for NSis).

- The Virtual Infrastructure Manager (VIM): it is responsible for the management of the NFV network resources. In practice, this corresponds to the cloud management platforms used today, such as Openstack, VMWare, Microsoft Azure, and Amazon Web Services (AWS).

- The NFV Management and Orchestration (NFV MANO) is the high-level component managing the catalogue and the life-cycle of the NFV network resources. The NFV MANO comprises the NFV Orchestrator (NFVO), the VIM, and the VNF manager (VNFM). The ETSI develops and maintains *Open Source MANO* (OSM)[18], an open-source implementation of the NFV-MANO standard.

Network slices (NSi) provide VNOs' clients a flexible and scalable ICT solution that follows their needs in space (geographic expansion) and time (growth in resource demand). This SotA cloud-native approach also allows the VNOs to meet the requirements of novel 5G and Beyond core networks: responsiveness, dynamicity, and scalability for MEC applications [19].

Research shows significant improvements in performance and efficiency optimization of network services through the NFV MANO leveraging ML and AI techniques to manage and strategically place virtual resources [20]. However, the NFV MANO's autonomous management also must consider the security in its strategic resource orchestration, including the prevention and mitigation of attacks targeting the NFV resources. To this end, MTD is a promising method that extends automated network management to security.

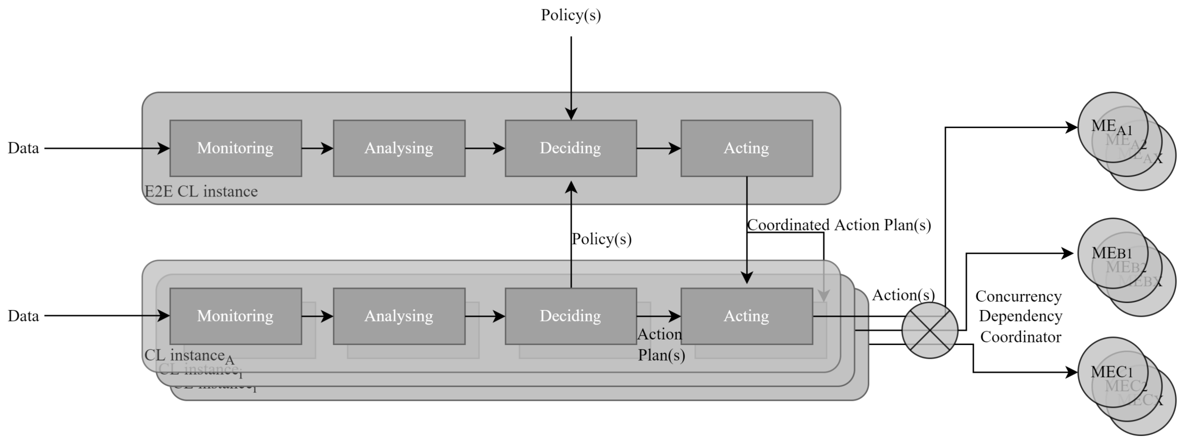


Figure 2.4: ETSI exemplary Closed Loop Coordination timeline [21].

2.1.3 Zero Touch Network & Service Management

ETSI Zero Touch Network & Service Management (ZSM) is an industry specification group (ISG) involved in the standardization of end-to-end network automation for service provisioning and lifecycle management in highly heterogeneous networking environments, such as 5G and B5G systems, employing various technologies to enable its vision, *e.g.*, AI, network slicing, and NFV. The ETSI ZSM standard defines the ZSM Framework that allows the self-optimization improvement of the network, according to specified service level agreements (SLAs). The cellular network is separated into discrete management domains, namely the Radio Access Edge, the Transport, and the Core domains.

Each domain has its closed-loop operation process, subject to an overarching *End-to-End (E2E) Service Management Domain* with a global perspective of the underlying domains and is responsible for the overall service provisioning to the customer. This ensures the smooth operation of the entire E2E system, given the growing trend of resource-sharing between operators (multi-tenancy) and the decentralization of network functions. Hierarchical approaches are predominant in solving the multi-domain management problem in ZSM, originating from the limited visibility of local domain agents in these approaches, given also the privacy and adversarial management implications in a multi-tenant network.

Various models describe closed-loop mechanisms, *e.g.*, the Orient-Observe-Decide-Act (OODA) and Monitor-Analyze-Plan-Execute-Know (MAPE-K) models [22]. Despite differences in step definitions, these models follow a similar high-level workflow: Monitoring, Analysing, Deciding, and Acting, as illustrated in Figure 2.4.

In this thesis, we concretely realize a cognitive closed-loop security management system for MTD operations on NFV resources (*e.g.* VNFs, CNFs, NSs, and NSis). The applied security management system has two possible entry points. The first entry point, the proactive one, deploys MTD operations ensuring compliance with the SLA and security requirements set forth, while the other entry point, the reactive one, enforces MTD operations as countermeasures when a change in the security context (*e.g.*, an attack) occurs.

2.1.4 MEC & FOG Computing

Multi-access Edge Computing (MEC) [19] is an ecosystem connecting telecommunications and IT services, enabling mobile applications to use computational resources at the edge of the RAN, at its cellular base station, bringing cloud computing capabilities closer to the end users' devices, *i.e.*, the UE. This approach offers significant enhancements to various 5G services. For FeMBB, MEC alleviates core network congestion by enabling localized content caching and processing, resulting in improved bandwidth utilization and reduced latency. Furthermore, MEC's distributed nature positions it ideally to support applications requiring enhanced ultra-reliable low-latency communication (eURLLC), such as remote surgery and TI. MEC's efficient resource management capabilities, leveraging NFV and SDN technologies, also prove advantageous for umMTC, enabling it to handle the vast number of connections and data volume generated by recent massive Internet-of-Things (mIoT) and IoV networks.

MEC applies as a standard architecture for telecommunication networks starting from the advent of 5G and spanning to future B5G networks. Standardization efforts are followed by the ETSI, in line with the NFV and ZSM standards, as standardized in ETSI GS MEC 003 [23], making it a valid technology to use for the design and implementation of a cognitive defensive solution for the Telco Cloud security leveraging MTD. MEC itself raises, among other concerns, the issue that cellular base stations are typically installed in public places. Thus, they have fewer physical surveillance measures than traditional core networks, although the criticality of the edge node increases in the MEC disposition. Moreover, the cloud platform at the edge has reduced capabilities compared to conventional data centers; thus, lightweight and computational efficiency are desired security application requirements to maximize the functional usage of edge nodes.

Fog computing, Similar to the concept of MEC and edge computing, fundamentally extends MEC by adding an intermediate layer in the network infrastructure spanning from the edge nodes to the centralized cloud data centers [24]. This layer, also referred to as the "fog layer", incorporates intermediate network nodes like routers and switches to provide computing resources, further pushing the distribution of computational tasks across the network. Centered around small distributed data centers known as cloudlets, fog computing is designed to minimize application delays and processing times. Cloudlets, possessing lower computing capacity and proximity to users, can leverage larger cloud data centers when needed. In scenarios of user mobility, such as in mobile networks, this can further develop into trying to make the service follow the users.

This development has fostered the concept of *Edge-to-Cloud Continuum*, which emphasizes the functional and logical relation of diverse network functions distributed across the infrastructure, from the edge to the cloud, and delivering a unified service, *e.g.* in IoV networks [25] that cater to the real-time communication needs of autonomous vehicles, and the tactile Internet [26] that supports near-instantaneous interactions for applications like remote surgery.

2.2 Moving Target Defense

MTD proactively changes the properties and configurations of an Information and Communication Technologies (ICT) environment. The MTD concept is not only applicable to network security: an example is the Address Space Layout Randomization (ASLR), used in the software security domain to protect code segments against buffer overflow (BoF) and other memory corruption attacks [28].

As depicted in Figure 2.5, MTD operations can be classified into three categories [29]: shuffle, diversity, and redundancy. *Shuffle MTD actions* (such as ASLR) change the layout/topology of a system/network (respectively) and can be applied on IP addresses, port numbers, packet headers, proxies, traffic routes, as well as VMs and container locations. Shuffling aims to reduce the asymmet-

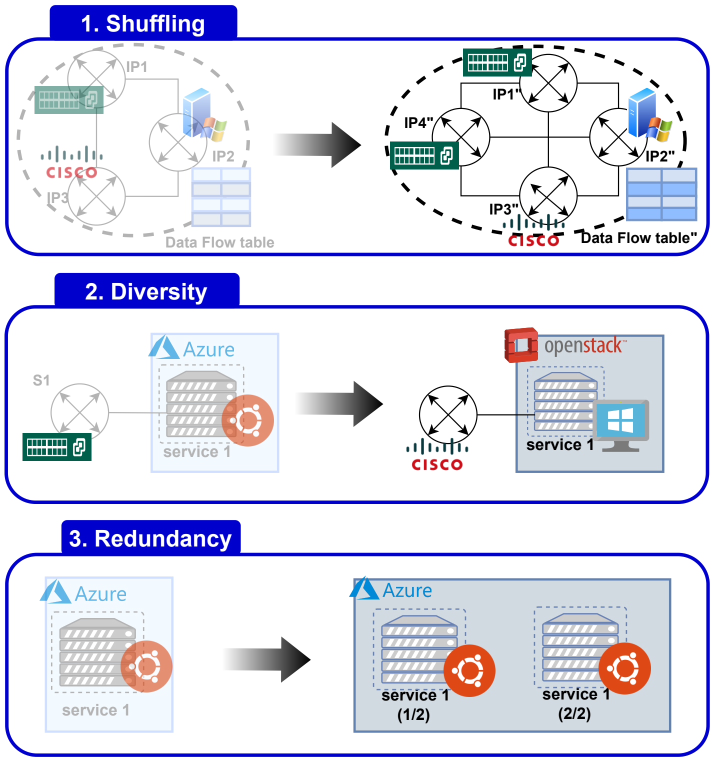


Figure 2.5: MTD operation types in a network [27].

ric advantage of attackers when using reconnaissance attacks to scan the network with tools such as pof, Nmap, and Nessus.

Diversity MTD actions change the execution environment of a running system. They modify the technology stack of the environment like operating systems (OSs), protocols, vendors, or cloud environments underlying virtual components. Diversity MTD actions prevent the exploitation of the environment's vulnerabilities and make an effective defensive measure under the assumption that the substitute technologies possess different exposures, *i.e.*, that the intersection of common vulnerabilities of the substitute technologies is smaller than the vulnerability space of a technology selected singularly. Practical examples of such operations are: changing an OpenVSwitch with a Cisco virtual switch, changing a Linux server with an equivalent Windows server, deploying the same service but implemented in a different programming language, or migrating a VM from an OpenStack to a Microsoft Azure cloud.

Redundancy MTD actions create hardware and software copies, such as backups and load balancers for fault-tolerance and availability improvements. Different techniques in this MTD category have already been extensively investigated in technical fields like fault-tolerant computing and cyber-resilience, and thus are not directly considered in this thesis.

2.2.1 Categories of MTD

Figure 2.6 is a visualization of MTD strategies and how they can be realized in a virtualized and software-defined network. Marker ① shows the MTD *shuffle* and *diversity* schemes that can be performed in a cloud environment to move virtual resources like VNFs and network slice components from an NFVI to another (*e.g.*, from VMware to OpenStack, or Azure), or distribute the resources of a network slice over to different cloud NFVIs, rather than grouping them in a single cloud infrastructure. Marker ② describes the mutation of the infrastructure layer linking different network elements, allowing to change the topology of the network and the packets' path (*e.g.*, with data flow tables of SDN controllers). Diversity can be added by using different switch vendors (*e.g.*, shuffling OpenVSwitch, Cisco, and Windows switches) or moving components from a local cloud at the edge of the network to a remote cloud via the Internet.

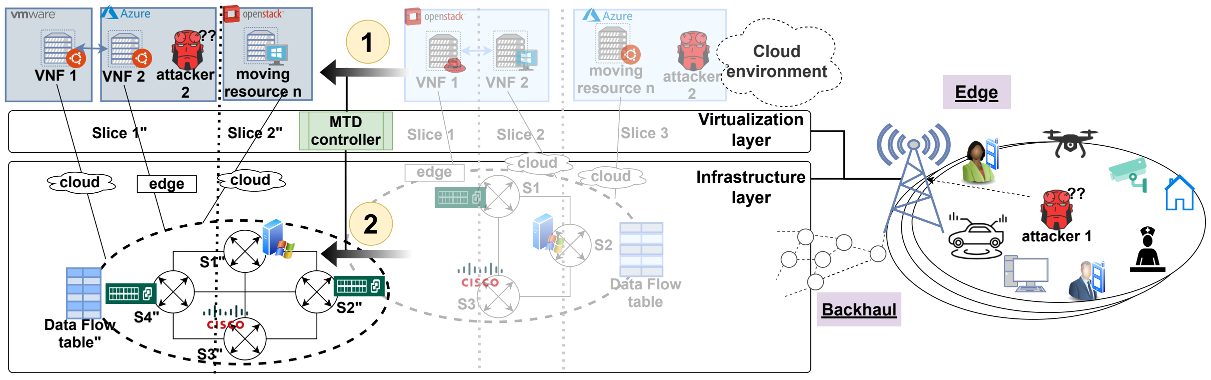


Figure 2.6: Combination of MTD operations at different levels of an NFV-SDN network [30].

In the context of 5G/B5G networks, MTD operations can be performed at three levels of an SDN-NFV infrastructure. Specifically, at the:

1. networking level: this concerns network traffic and protocols, and includes attacks such as reconnaissance, Man-in-the-Middle (MitM), or Denial-of-Service (DoS);

2. virtualization level: this concerns the hypervisor's vulnerabilities and the isolation of VM-based or container-based network functions;

3. application level: this concerns software vulnerabilities and exploits.

For NFV infrastructure management and orchestration, MTD operations are mainly executed at the first two levels of the infrastructure: the networking and virtualization levels. The security of the application level is under the responsibility of VNF developers. Therefore, application vulnerabilities are checked and patched by developers and out of the scope of VNO's network management systems. At the networking level, MTD can change traffic routes using SDN control, change virtual switches for diversity shuffles (*e.g.*, change a Cisco virtual switch with an Open vSwitch), and modify the network topology using NFV and proxy nodes. At the virtualization level, MTD can migrate virtual resources to different cloud platforms (*e.g.*, move a VNF from an Openstack Virtual Infrastructure Manager (VIM) to a VMWare or an Azure one), changing the virtualization stack on which network functions are running.

2.2.2 MTD Optimization Challenges

There are various pertinent challenges that must be addressed to facilitate and enhance the cognitive MTD security approach within envisioned 5G/B5G networks. This is due to the combination of multiple factors involved in changing configurations and positions of services while keeping the network manageable and coherent, especially in a complex multi-domain and multi-tenant environment such as B5G. A concise depiction of these factors and challenges is provided in Figure 2.7, followed by more detailed discussions in the next subsections.

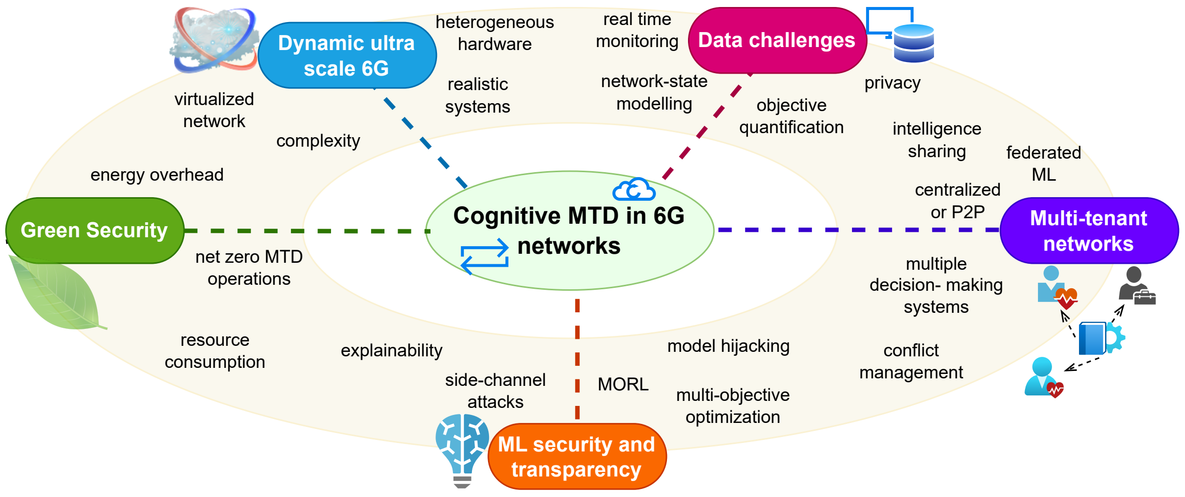


Figure 2.7: Research challenges and directions for MTD in B5G/6G [27].

Dynamic Ultra-Large-Scale Networks

An inherent problem is the complexity since B5G networks are ultra-large-scale systems incorporating various systems, technologies, and devices, from IoE to space networks [3]. VNOs will also be faced to a spectrum of service requirements and hardware/software performance capabilities. For instance, B5G-enabled remote surgery demands ultra-low latency (< 1 ms) and high edge computational power (from 100 Giga Operations Per Second (GOPS) to 1 TOPS) for real-time robot control, image processing, and haptic feedback utilizing Extended Reality (XR) interfaces. This translates to a need for high bandwidth (from 100 Mbps to 1 Gbps) for seamless graphical and sensory data transfer. In contrast, smart agriculture applications may leverage B5G MIMO technology operating in the 1-10 GHz frequency range to achieve higher spatial resolution for radio access points (RANs). However, it does not necessitate the stringent latency or bandwidth requirements of remote surgery and exhibits lower sensitivity to network instability. This inherent heterogeneity in service demands significantly influences the cost models associated with MTD actions, ultimately complicating the optimal MTD policy. Notably, the negative effects of applying a uniform network performance approach could be significantly more detrimental to a service like remote surgery compared to smart agriculture.

Moreover, in the context of ML-based optimization, there is the constraint to initially train the ML model on a representative yet limited testbed. Thus, the optimization model's performance must be subject to re-evaluation within the context of the practical network's settings and requirements.

Data Challenges

The significance of data engineering during the MTD optimization process cannot be overstated. Given the diverse range of objectives under consideration for the optimization of the MTD strategy, a comprehensive aggregation of real-time data sourced from the network's monitoring infrastructure is necessary. Next, to harness the value of such data, it becomes imperative to establish a well-designed reward system. This system translates the collected data into quantifiable metrics that reflect the network's status toward the objectives outlined in the optimization problem. Furthermore, it is crucial to exercise caution when training ML models on monitoring data of public networks. This practice carries the inherent risk of revealing sensitive information, such as end-user personal data, and impairing privacy. Consequently, vigilant safeguards must be implemented to avert any inadvertent data leakage from the model. Additionally, privacy-preserving data analytics techniques could be employed.

MTD in a Multi-tenant B5G Network

An envisioned advanced MTD design for B5G networks involves the integration of a multi-tenant federated MTD approach, aimed at capitalizing on the diverse range of network experiences and threat intelligence of various VNOs. Different architectural patterns exist for such a federated solution. One approach entails a centralized MTD model operating within a hierarchical relationship among tenants/operators. An alternative perspective relies on a distributed MTD model characterized by a peer-to-peer (P2P) tenant structure, where independent decision-making systems function autonomously. While the former exhibits simplicity in implementation, it requires trust and data transparency – frequently impossible to facilitate. In contrast, the latter P2P configuration offers tenants greater independence but amplifies the difficulties associated with intelligence sharing and interplay between disparate models. Another prospect is hybrid systems containing both federated and locally oriented decision-making systems. However, such approaches necessitate incorporating a conflict resolution system. For instance, a hierarchy could be established among these decision-making systems, prioritizing the local model over the federated one, or vice versa.

Sustainable Security

The exploration of energy-efficient MTD orchestration as another optimization objective constitutes a relevant research direction in light of environmental considerations pertaining to sustainable energy consumption and the overarching goal of achieving a global net-zero footprint and UN Sustainable Development Goals (SDG) [31]. In this context, the integration of MTD techniques can consider the carbon emissions associated with network slices' activities, *e.g.* by strategically placing VNFs within cloud nodes powered by green energy sources rather than fossil-fuel-powered nodes. Furthermore, the energy cost for MTD actions can be integrated into the optimization model.

Security and Explainability of ML-based MTD Policies in Critical Infrastructures

While an MTD framework provides an additional layer of security to future networks, it can also be a new point of attack that can be exploited. The ML model, for instance, is a sensitive attack surface, as it can be vulnerable to hijacking, poisoning attacks during training, evasion techniques such as adversarial example attacks, or triggering meaningless MTD operations as a DoS attack and resource waste [3]. Measures can be implemented to safeguard data integrity and authenticate the origin of data, particularly from network endpoints providing monitoring data.

For technical, economic, and legal reasons, the decisions of the ML agents should also be humanly explainable (*i.e.*, with statistics, heuristics, and formal logic). For instance, the European Union Agency for Cybersecurity (ENISA)[32] emphasizes the explainability of AI and ML algorithms, especially Deep Neural Networks (DNN), which are not explainable by nature. However, Integration of explainable AI into MTD is not a trivial task and remains an open research question.

2.3 Live Migration of Services

As introduced in the previous section (Section 2.1), Telco Cloud networks, such as 5G/B5G, expand telecommunication networks with cloud-centric data centers and edge nodes, inheriting the properties of cloud infrastructures. Such properties range from the virtualization of resources to programmable networks, both used to facilitate the orchestration of network functions managing the telecommunication infrastructure, and the final applications provided to end-users. In this context, Live Migration (LiMi) plays an increasingly critical role in these environments by enabling seamless application mobility covering the edge-to-cloud continuum, thereby maintaining these requirements and enabling agile services. LiMi, or hot migration, of virtualized services, whether hosted as VMs or containers, consists of migrating the service's instance from the host machine to a new one while the service is being available to the end-users. This opposes cold migration, where a migrated service needs to be stopped first, hindering the continuity of its functions. This disruption, even in the order of tens of seconds, can be undesirable when dealing with live or critical applications such as live video streams, autonomous vehicles (*e.g.*, drones and cars), healthcare systems, and industrial control systems.

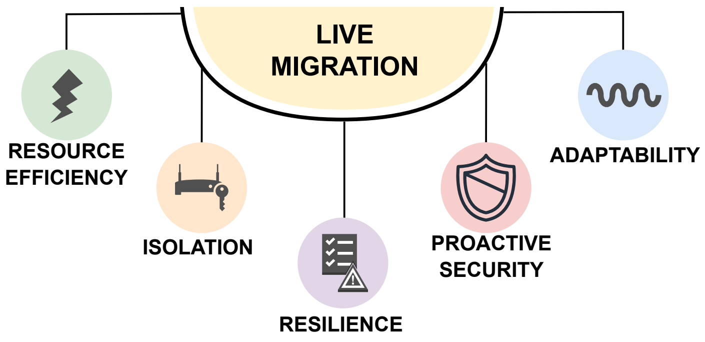


Figure 2.8: Essential features in ICT systems enhanced by LiMi [33].

LiMi can be used to perform MTD operations, which is one of the main focuses and contributions of this thesis. In fact, LiMi could be used as a service shuffling method to improve the isolation of services in multi-tenant hosting infrastructures, to enhance networks' resilience against cyber threats, and to maintain fault tolerance and availability of networked services. Lime also improves the adaptability to new regulations and security policies (whether for technical or geopolitical considerations), Finally, LiMi could also be used in other relevant topics, such as managing and optimizing the carbon footprint associated with networked applications. This aligns with the imperative of developing networking practices that align with the Net Zero emissions and climate-neutrality targets set or considered by over 100 countries in international UN organizations and various agreements, including the Paris Agreement[34].

2.3.1 Security Properties of Live migration

LiMi plays a crucial role in today's network evolution to ensure and improve various requirements (depicted in Figure 2.8) of applications in the edge-to-cloud continuum, such as isolation, adaptability, resilience, availability, security, and resource efficiency.

Improved isolation in multi-tenant infrastructures (non-static hosting)

Most cloud-hosted services share their infrastructure with other tenants and owners of various services. The hypervisors managing the service instances are often vulnerable. While vulnerabilities that allow direct unauthorized access to third-party resources are rare and quickly disclosed and patched, other, more passive attacks, such as side-channel attacks, are more common. Side channel attacks allow attackers to get sensitive data on neighboring VMs or containers hosted within the same node [35–37]. By dynamically migrating such VMs and containers, the time exploitation window of such attacks is substantially reduced as services move to different nodes and locations of the cloud infrastructure. Recent studies and solutions also deal with the isolation problem using the TEE technology [38]. In this scenario, containers and VMs run in encrypted enclaves, guaranteeing confidentiality even in a multi-tenant system. Enabling LiMi for such TEE-protected systems becomes relevant to provide an additional layer of isolation while also removing static-hosting constraints on services.

In fact, another feature enabled by LiMi against multi-tenant threats is the on-demand allocation of a private physical node within the cluster, which would only be used by a single applicant. This entails the LiMi of the services of the applicant to the new node and the displacement of unrelated resources to other nodes. LiMi to a dedicated physical node is also critical in scenarios of scaling out the service and in High-Performance Computing (HPC), to avoid performance interference and ensure consistency throughout the service execution.

Adaptability to changing policies and regulations

An important property needed for cloud-based infrastructures is the possibility to dynamically adapt to changing policies and regulations. For instance, with the advent of the European General Data Protection Regulation (GDPR), the personal data that is generated in Europe cannot be stored or processed outside of the European territory unless an 'Adequacy Decision' is offered by the European

Commission or a user explicitly consents (and can revoke) the transfer of their data [39]. At the occurrence of this kind of new regulations, especially when frequently happening (*i.e.* due to shifting geopolitical conditions), LiMi would allow swift adaptation with minimal efforts by migrating the resources, data, and processing applications to the required location (assuming the infrastructure at such a location is available).

Improved proactive and reactive security

LiMi of services dynamically alters the attack surface of cloud-based infrastructures, making it a versatile MTD mechanism. Reactively, LiMi can mitigate detected attacks by relocating services away from compromised nodes, effectively evading ongoing threats. Proactively, LiMi reduces the ASP by periodically shuffling workloads, thereby disrupting attacker reconnaissance and exploitation attempts, even for undetected threats. This approach is particularly effective against advanced cloud-based attacks, including side-channel attacks, data exfiltration, isolation evasion techniques (such as vertical movement in multi-tenant environments), and malware propagation [33]. The efficacy of LiMi as an MTD strategy against these threats is rigorously analyzed and evaluated in this thesis.

Increased availability and resilience

These fundamental and classic requirements have always been targeted at the different levels of ICT systems: software engineering, hardware, infrastructure, and networking. The different levels have possibly met and culminated in the virtualization technology with the microservices paradigm[40], which is changing the way software is developed, served, and managed in new cloud-based networking infrastructures. It is in such an environment that LiMi sees its best prospect and usage, as it uses the characteristics of microservices-based ICT systems: modularity, portability, replicability, distribution, and autonomous orchestration. In microservices, availability and fault tolerance are mainly improved with the replication of the applications and the load balancing of requests to the various replicas. LiMi can keep the same level of availability across different distributed nodes while containing the number of replicas, which consume more resources. Moreover, traditional microservice replication only applies to stateless microservices, *i.e.*, applications that only need the VM/container image to start serving the service, as that does not make use of dynamic contextual data or state. However, with LiMi, both stateless and stateful running applications can be migrated from one node to another, making it possible to also migrate the clients' connections. In MEC and fog computing scenarios, this also enables bringing the service closer to mobile users, thereby improving the availability and quality of service (QoS) in use cases such as smart connected vehicles and mobile devices [41] [42–44].

Energy and carbon footprint control

The power consumption of datacenters and cloud infrastructures in 2022 was 240-340 TWh, about 1-1.3% of global electricity use [45]. Despite notable improvements in efficiency, the substantial increase in workloads managed by data centers has led to a significant rise in energy consumption, experiencing an annual growth rate ranging between 20-40% [45].

LiMi can serve as a mechanism for regulating and overseeing energy utilization. Restricting application replicas while maintaining high service availability via migration offers a means to limit resource and energy consumption, as previously discussed. An alternative approach, referred to as "consolidation", involves live migrating all active applications within a cluster to a minimal number of server nodes, as demonstrated in Hermenier et al.'s work on Xen VM migrations [46]. This practice enables the hibernation of idle server nodes during periods of decreased resource demand, effectively mitigating energy consumption. In fact, even at a low 10% CPU utilization, the power consumption remains high, exceeding 50% of the host's maximal power consumption [47]. Conversely, in certain higher workloads, distributing applications across more nodes can result in reduced waste heat dissipation, thereby lowering the energy consumed by cooling systems, along with improving the user's QoS.

2.3.2 Live Migration Methods and Taxonomy

LiMi and most of its optimization techniques originated with VM technology [48–55].

The same principles and optimizations are transferable to container runtimes as well. LiMis can be classified as *inter-cluster* or *multi-cluster*. In *inter-cluster* migrations the application is moved between nodes of the same cluster, and this can be performed in two different ways:

- *Over shared storage*, where the volume and data of the application are immovable and placed in a storage network, which will be mounted on the application instance in the new node (see Marker ① in Figure 2.9)[51].
- *Over the network*, where the volume and data of the application are attached to the running instance and are moved altogether through the network (see Marker ② in Figure 2.9)[48].

In *multi-cluster* migrations, the application is transferred from one cluster to the other, and this can be extended to multi-cloud and/or multi-vendor migrations [56]. As the destination in these sce-

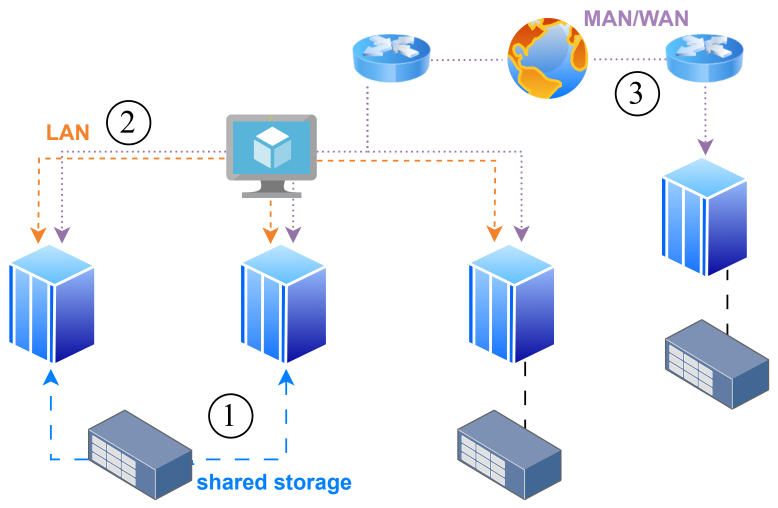


Figure 2.9: Types of LiMi in a multi-cloud environment [33].

narios is far from the source, the storage network cannot be the same, as it will create a big communication overhead in latency and traffic, hindering both application QoS and network resource consumption. The full application transfer then happens over the WAN/Internet, as shown by Marker ③ in Figure 2.9, making it a slower process compared to inter-cluster migrations, depending on the geographical distance between source and destination nodes, the quality of the physical connection, and the size of the application to migrate [53].

VM vs container LiMi

In this subsection, we point out some differences between VM LiMi and container LiMi, summarized in Table 2.1.

With respect to literature coverage, scientific contributions, and technological maturity, VM LiMi is at an advantage as it came first and has been extensively used (and is still being heavily used) in critical infrastructures by industries and enterprises. Thus, VM LiMi has a larger knowledge base and finds itself more reliable and used compared to its container-based counterpart, which is a relatively newer technology that has been less tested in enterprise environments.

In theory, VMs are also more portable compared to containers, and thus, they have fundamentally fewer constraints for the destination node to enable the migration. On the other hand, containers are dependent on libraries of the host's OS, thus requiring the same libraries (or OS) on the destination host. However, in real-life LiMi scenarios, this difference between VMs and containers in terms of

Table 2.1: VM vs container LiMi.

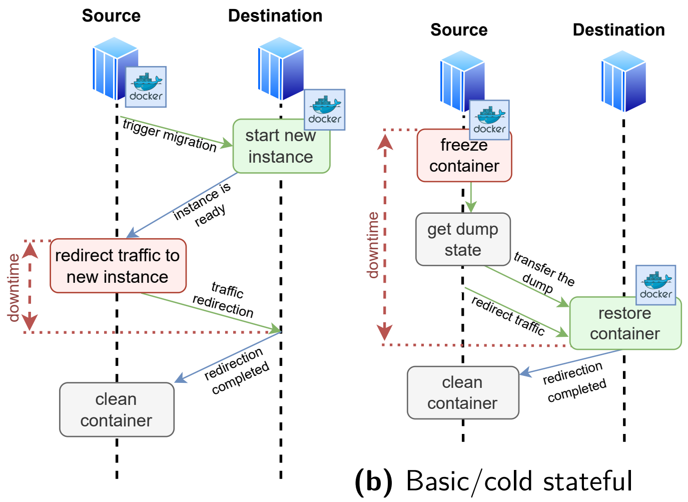
Factors	VM migration	Container migration
Maturity	Mature enterprise use of migration technology.	Newer and less documented in enterprise use cases.
Flexibility	VMs are more portable as they are independent of the host OS, enabling, in theory, LiMi with fewer/no constraints on the host machine.	Containers are dependent on libraries of the host's OS, thus requiring the same libraries or OS on the destination host.
	VM LiMi's optimizations mostly depend on specific implementations, thus requiring the same hypervisor at the destination.	Almost all container LiMis are currently done with CRIU, which runs on any Linux OS, independent of the container orchestrator.
Efficiency	VMs are generally large implementations of an entire ecosystem with multiple software/applications $\Rightarrow$ reduced control of migration granularity.	Containers are small single applications running as modular components in a microservices environment $\Rightarrow$ high control of migration granularity.
	A VM image size is in the order of gigabytes, requiring greater bandwidth consumption during migration.	Container images are measured in megabytes, making the migration less taxing on the network.
Performance	The significant data transfer causes a longer migration downtime (during which service is unavailable).	Lightweight container images cause shorter migration downtime.
	VM warm startup (which excludes the first boot) is in the order of seconds. The startup time contributes to increasing the migration downtime.	Container warm startup is in the order of milliseconds. However, the first boot (cold startup) is similar to or longer than VM's boot due to *filesystem remap*.

portability/cross-migration could be reversed. This is because VM LiMi implementations, especially with live optimizations, are mostly closed-source and licensed, such that both source and destination nodes are required to run the same hypervisor (or one from the same vendor). However, almost all containers' LiMis, especially with live optimizations, currently use the open-source library CRIU (further described in Chapter 4, Section 4.3), which can run on any Linux OS, making it more usable across a variety of infrastructures and vendors.

Because VMs are much larger in size, they take longer to migrate, and for the time-constrained requirements of live/hot migration, this is a significant drawback. On the other hand, container images are much smaller and lightweight, leading to faster migration with reduced data transfer and network overhead. Kotikalapudi et al. [57] demonstrated that migrating the identical service with a 66% workload, encapsulated within a system container, exhibits a 45% acceleration and a 75% reduction in downtime compared to its encapsulation in a VM. For a meaningful comparison, the authors used Linux containers (LXC), which are full system containers providing an entire Linux system and its kernel, as opposed to application containers such as Docker, which only package the application and its dependencies (which would further reduce the container's size).

Because VMs have to initiate a full operating system's (OS) kernel, VM-based applications have significantly longer spin-up times [58], meaning it takes longer for a service to be available and served at the instantiation phase (it generally takes minutes with VMs and a few seconds, often milliseconds, with containers). These times can be added to the downtime of the service during the migration (*i.e.*, the time where the service is not available to the end-users). This is relevant to stateful applications, as opposed to stateless applications where the instantiation time can be neglected, as further clarified in the next Section (2.3.2).

Another important difference is the way applications are implemented when using VMs compared to containers. For instance, when vendors implement their virtual network functions (VNFs) for Telco Cloud networks following the NFV standard[10], they tend to implement all the components needed in one single VM, making it one big component. On the other hand, for CNFs, they take the microservices approach, implementing multiple smaller services in portable and more or less independent containers that are easier to transfer. Hence, the relevance and importance of LiMi research in microservice architecture for future networks is evident as described in the previous section.



(a) Stateless LiMi. **(b)** Basic/cold stateful LiMi.

Figure 2.10: Stateless vs stateful service LiMi [33].

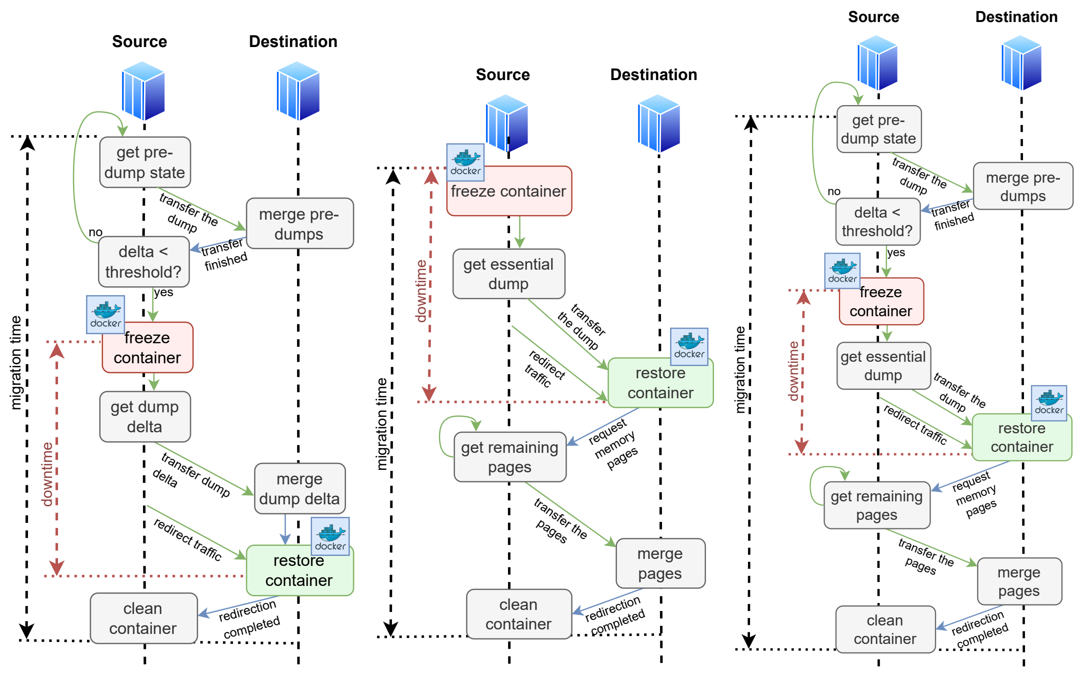
Stateless LiMi

LiMi methods and algorithms also differ based on whether the service to migrate is stateless or stateful. Stateless services are, for instance, the majority of firewalls, DNS servers, or network gateways that do not have a dynamic context to keep and only have rare changes in their configuration that could be added preemptively to the system image used to start the service. In this case, the LiMi becomes much more accessible and efficient because there is no need to transfer the running instance and its volume. A new instance of the stateless service can be created at the destination node while the old instance is still running. Once this new instance is ready, the clients' traffic is redirected from the old instance to the new one. The relevant property of this method is the minimal service disruption that happens only during the traffic redirection, in the order of milliseconds, as depicted in Figure 2.10a. In contrast, the downtime of stateful services includes the transfer time of the application state (comprising the VM or container volume) and the spin-up time for the application to start at the destination node, as depicted in Figure 2.10b.

Another relevant property of stateless migrations, in contrast to stateful ones, is that the image used to start the new instance at the destination node is an original image of the service that can be guaranteed not to contain any malware or backdoor that was installed during a previous service lifetime cycle. That is to say, even when the previous instance of the service is unknowingly infected with an advanced persistent threat (APT)[59], after a stateless LiMi, such infection is effectively removed from your service. This property alone makes stateless LiMi a highly required security feature for future networks, while stateful LiMi does not have this property and would transfer the infection or APT along with the service's instance. Unfortunately, stateful services are the most common types of services and therefore a relevant challenge is how to make this security property applicable in such circumstances. This challenge is further discussed and tackled in Section 4.3.

Stateful LiMi

Stateful services have a dynamic context and use data that frequently changes during its life cycle. Therefore, the standard method to live migrate them is to stop the instance (*i.e.*, the VM or container), make a checkpoint of the instance's state, transfer this checkpoint to the destination node, and restore the instance from the checkpoint (as depicted in the Figure 2.10b). The main issue with this method is that the checkpointing of the service is performed during the *state dump* process. The state dump requires the instance to stop, implying that during the whole migration process, the service is not available to the end-users. This leads to a service downtime that depends on the service's instance size, checkpoint transfer speed, and instance restoration time. Thus, three optimization methods are traditionally used to solve this problem and reduce the service disruption:



(a) Pre-copy LiMi. (b) Post-copy LiMi. (c) Hybrid LiMi.

Figure 2.11: Different stateful LiMi optimizations [33].

Pre-copy optimization

The pre-copy or iterative optimization method is depicted in Figure 2.11a and consists of performing preliminary service state checkpointings, called *pre-dumps*, that do not require the service to be stopped [48]. As the service is still in use, the service state can change during the pre-dump, and the difference, named delta, is the only data that requires the service to freeze in order to be checkpointed and transferred with a *dump* call. The dumped delta is then merged with the pre-dumped images, allowing to restore the full instance at the destination node. To reduce the delta to its minimum, the pre-dump can be iterated several times before the final blocking dump. The iterative pre-dumps only consist of those memory pages that have been modified, also named dirty pages, that are added to the pages of previous pre-dumps [54]. The pre-dump iteration is configured by setting the maximum number of iterations wanted and the number of new dirty pages below which the blocking dump can be called to resume the migration. This configuration can be set and optimized with heuristics based on the service nature, as too many pre-dump iterations increase the overall migration time and network overhead, while a short pre-dump phase would instead increase the service downtime based on the frequency of memory changes in the service. Elsaid et al. provided a survey on the literature focusing on the pre-dump parameters' optimization using mathematical models and heuristics [60].

In a separate study, Ma et al. [54] enhanced the pre-copy method to limit the iteration process to five iterations. This involves the exclusion of the most frequently updated memory pages from immediate copying, marking them to be copied only at the last round of the iteration process. Their method allows for a reduction of up to 34% of the total transferred data and up to 32.5% of the total migration time.

Post-copy optimization

The post-copy or lazy migration method is another optimization method designed to reduce service downtime and is depicted in Figure 2.11b. It first transfers with a blocking dump only the essential state that allows the instance to start at the destination node, such as the CPU state and the registers. Once the service runs and is available to users, the remaining memory pages are then injected into the running task address space in a phase called *pre-paging*, to restore the full-service state[61]. While the post-copy allows the downtime to be consistently reduced independently of the size or the update frequency of memory in the application, the quality of service (QoS) gets degraded when users' tasks need data that is still to be injected, moving from latency levels of RAM to considerably worse latency levels of network i/o operations.

Hybrid optimization and comparisons

Other hybrid optimization methods exist using both pre-copy and post-copy methods [55], as illustrated in Figure 2.11c. The objective of a hybrid method is to use the pre-copy to reduce the time of the pre-pages migration phase, in order to reduce the exposure to a communication failure. At the same time, the post-copy would reduce the number of pre-dumps, making the total migration time less dependent on the frequency of memory changes in the application. However, as summarized in Table 2.2, the hybrid method also inherits both the drawbacks of pre-copy and post-copy, making it the method with the longest total migration time, as it has both a pre-dump phase and a pre-paging phase. The next subsubsection details the performance evaluations in the literature, pointing out which methods to use based on the characteristics of the service to migrate, the network and infrastructure in place, and the requirements of the service.

2.3.3 Dynamic Networking Support for Live Migration

During LiMi, successful traffic redirection is crucial to maintain service availability for users after an instance is restored at the destination node. This redirection can be achieved through various

Table 2.2: Evaluation and comparison of different LiMi methods [33].

Factors	basic	Pre-copy	Post-copy	Hybrid
Total migration time	lowest	low	low (except with throughput $\approx$ PDR)	highest
Service corruption	no	no	possible	possible
Downtime	high	low (except with throughput $\approx$ PDR)	low	low
QoS degradation	no	no	yes	yes
Total transferred data	low	high	low	middle

methods, all with the common goal of minimizing latency and QoS overhead. The choice of method depends on the ecosystem's properties and the migration scenario, as illustrated in Figure 2.12.

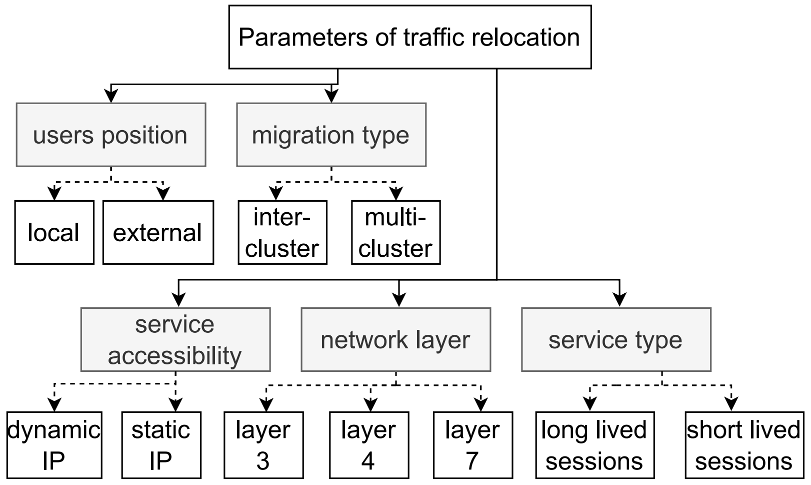


Figure 2.12: Taxonomy of traffic relocation for LiMi [33].

These methods encompass:

1. *Service users' position*: this varies based on whether the users are connected externally (*i.e.*, via the internet) or whether they are in the same local network or infrastructure of the application;

2. *Migration type*: this depends on whether the service is migrated within the same cluster or on a different one;

3. *Accessibility*: this depends on whether the IP of the service can be different after the migration or if it should keep the same IP for external accessibility;

4. *Network layer*: this factor depends on whether the connection handover of communication happens at the IP layer (*i.e.*, layer 3 of the TCP/IP stack), transport layer (*i.e.*, layer 4), or at the application layer (*i.e.*, layer 7);

5. *Service type*: this depends on whether the service is using TCP short-lived sessions (*e.g.*, HTTP requests or REST API requests) or TCP long-lived sessions (*e.g.*, Internet-of-Things (IoT) devices, real-time applications, or network file transfers);

For LiMi within the same cluster or LAN, a service can keep the same IP address performing the network migration at the link layer (*i.e.*, layer 2) using ARP packets to update the IP and MAC address mapping in the order of tens of milliseconds [48]. For wide area network (WAN) migrations, the redirection has to happen at the transport layer or application layer. Most of the proxies implemented for load balancing at the transport layer make use of dynamic DNS, *iptables*, or Linux kernel's IP Virtual Server (IPVS) to perform network address translation (NAT). Replies will also be redirected with a reverse DNAT rule, keeping the private IP hidden from the end user. As UDP is connectionless at the transport layer, packets are accepted and directly pushed to the application layer. However, when dealing with TCP-based traffic, each packet is bound to a connection defined by the tuple (`src_ip, src_port, dst_ip, dst_port`). Using simple DNAT will work only for newly established connections, while old running connections continue to use the old private IP as a destination until their end. This results in a service disconnection for such instances when the service at the source is terminated, leading to an ungraceful attempt to reconnect with the new IP address. Further details on the state-of-the-art methods for traffic redirection in LiMi are given in the literature review chapter of this thesis, Chapter 3, Section 3.3.

2.4 ML Fundamentals and ML Algorithms

Machine Learning (ML) is a branch of AI focused on developing systems that can automatically learn from data and improve their performance on specific tasks without being explicitly programmed with a human-designed algorithm. ML is generally split into three main categories: supervised learning, unsupervised learning, and RL. While supervised and unsupervised learning primarily address pattern recognition problems (*e.g.*, classification and clustering), RL focuses on optimizing sequential decision-making processes through trial-and-error interactions with an environment.

In supervised learning, the most common approach to classification, models are trained on labeled datasets, where each object is already assigned to a specific class. The ML model then learns to predict the class of unlabeled objects. Provided the dataset is comprehensive and free from issues like class imbalance, bias, or errors—often the most challenging aspect of implementing an ML classifier—performance evaluation becomes straightforward. Standard metrics such as false positive rate (FPR), true negative rate (TNR), recall, accuracy, and precision are used to measure the model's performance by comparing its predictions to the ground truth.

In contrast, RL faces a different challenge in optimizing decision-making processes, as there is no pre-defined ground truth for comparison. The performance of an RL model can only be assessed through practical comparisons with other RL models or against predefined key performance indicators (KPIs).

The following section describes the principles of RL algorithms, how the environment it interacts with (in our case, the 5G/B5G telecommunication network) is modeled into a Markov Decision Process (MDP), and how this formalization is used in different RL algorithms to learn the optimal decision policy.

2.4.1 RL and MDP Modeling

As depicted in Figure 2.13, RL involves training a decision agent to optimize sequential decisions based on experience, resulting in learning overall complex behaviors. Here, the agent learns by interacting with an environment, observed and modeled as a Markov Decision Process (MDP), a discrete-time stochastic control process. Formally, an MDP is defined by a tuple (S, A, P, R, γ) where S is the set of states of the environment, A is the set of actions that the agent can take, P is the transition probability matrix defining the probability that an action a_i changes a state s_i to a new specific state s_j, R is a set of reward values for all (a_i, s_i) pairs and γ is the discount factor defining the importance of the immediate rewards over the future rewards.

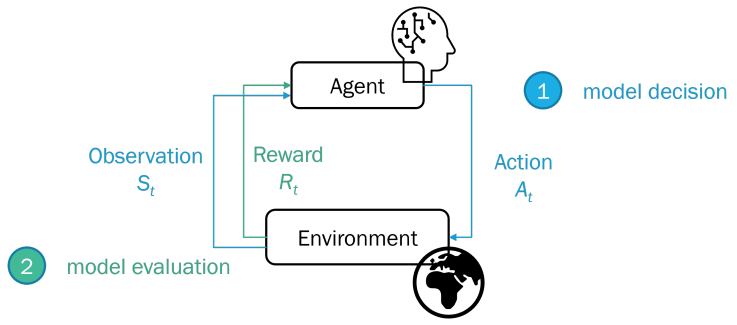


Figure 2.13: Interaction of the RL agent with the modeled environment.

The agent receives rewards or penalties (*i.e.* negative reward values) based on the observed effects of its actions on the environment, aiming to maximize its cumulative return, denoted as G_t, which is the sum of rewards accumulated throughout the RL task.

RL tasks can be finite (*i.e.*, episodic tasks), where an episode consists of a defined starting and ending state within the MDP, or infinite (*i.e.*, continuous tasks), where the MDP has no terminal state. An example of an episodic task is a game, which concludes with a win or loss, while continuous tasks typically involve management and orchestration tasks, such as the MTD strategy optimization formulated in this thesis. In continuous tasks, where there is no defined endpoint, the return is computed using the discount factor γ as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The RL agent's goal is to update its policy π, which represents the probability distribution of all actions a in the action space A for each state s in the state space S, until the policy converges to the optimal policy π^* that maximizes the return, *i.e.*,

$$\pi^* = argmax_{\pi} E_{\pi}[G_t | S_t = s], \forall s \in S$$

This optimization can be approached in two ways: *1.* by evaluating states and actions of the MDP using **value-based methods**, such as the Bellman equation [62], Q-learning [63], and state-action-reward-state-action (SARSA) [64]; or *2.* by directly evaluating and fitting the policy function using **policy-based methods**, which include algorithms like policy gradient [65] and REINFORCE [66].

A third hybrid approach, known as **actor-critic methods**, integrates the advantages of both value-based and policy-based approaches. This architecture features two complementary components working in tandem: the actor implements the policy function and determines which actions to take, while the critic assesses these actions by calculating their estimated value (through either state-value or action-value functions). The critic's evaluations provide essential feedback that guides the actor in refining its policy to optimize reward outcomes. This synergistic design allows actor-critic methods to capitalize on value-based methods' stability and computational efficiency while retaining policy-based methods' adaptability and ability to directly optimize policy performance.

2.4.2 DNN Integration and deep-RL Taxonomy

Value-based and policy-based methods are examples of what are known as model-free RL algorithms. These algorithms operate without requiring a predefined model of the MDP, instead estimating the policy value solely through interaction with the environment. RL faces the problem known as the "curse of dimensionality", occurring when the environment's exploration space is extremely large, involving a vast action or state space. Agents may use a predictive model that constrains the exploration space. These predictive models serve as guides or frameworks for the learning process, leading to faster convergence (*i.e.,* quicker training). However, these models may introduce limitations in asymptotic performance due to inherent modeling restrictions or biases.

Model-free RL algorithms impose fewer constraints on the model and employ exploration mechanisms to prevent the agent from becoming trapped in local optima [67]. However, it suffers from the curse of dimensionality problem requiring extensive interaction with the environment to learn effectively. Deep-RL helps address this inefficiency by approximating the optimal policy using DNNs rather than greedy tabular-based algorithms. Deep-RL enabled the agent to handle complex high-dimensional environments [68], contributing to major AI/ML breakthroughs in various fields such as robotics, autonomous driving vehicles, and more recently, generative conversational applications like ChatGPT [69]. ChatGPT employs a semi-supervised training method, incorporating deep-RL to optimize Large Language Models (LLMs) based on human feedback.

Deep-RL algorithms have an extensive taxonomy as DNNs have been applied on value-based methods, such as with Deep Q-Network (DQN) and its variations [70–72], on policy-based methods, such as Trust Region Policy Optimization (TRPO) [73], and actor-critic methods, which sees the currently most used state-of-the-art deep-RL algorithms such as Asynchronous Advantage Actor Critic (A3C) [74], and Proximal Policy Optimization (PPO) [75]. Actor-critic methods are a hybrid approach combining both value-based and policy-based methods, such that the RL agent uses an actor component to directly optimize the policy and a critic component that evaluates the policy through the value function. Other more advanced deep-RL methods have also been proposed, such as multi-objective deep-RL [72, 76, 77], partially-observable MDP (POMDP) based deep-RL [78, 79], and multi-agents deep-RL [80].

2.4.3 Federated Learning and Federated deep-RL

Federated Learning (FL) is an area of ML focused on methods that enable multiple, distributed parties to collaboratively train a shared ML model using each party's locally held data. This collaborative approach results in a model that is more robust and accurate than those trained on any single party's data alone. The training of the ML model is then extended to two steps: ***1.*** the first standard step is the local model training, in which the local participant *i* obtains optimal model weights w_i^t at timestep

t minimizing the value of the loss function $L(w_i^t)$, *i.e.*,

$$w_i^t = \arg\min_{w_i^t} L(w_i^t)$$

2. The second step is the global model aggregation, where the different participants aggregate their weights w_i^t using algorithms such as FedAvg [81] to a global model weight w_G^t minimizing the global loss function $L(w_G^t)$, *i.e.*,

$$w_G^t = \arg\min_{w_G^t} L(w_G^t) = \arg\min_{w_i^t} \frac{1}{N} \sum_{n=1}^{N} L(w_i^t)$$

FL addresses the challenge of low sample efficiency, particularly in cases requiring substantial data diversity to improve model generalization. By enabling parallel model training across distributed data sources, FL accelerates learning and convergence rates through effective information exchange among participants. Additionally, FL is particularly valuable for privacy-sensitive applications where direct data sharing is not feasible due to confidentiality constraints, as it provides a privacy-preserving mechanism for model aggregation that merges model weights without exposing local data. The primary goal of FL is to derive an aggregated model M_{FED} that would have approximately the same performance V_{FED} as a single ML model M_{SUM} that would be trained on the union of all parties' data in a centralized setting. Formally, let ε be the real number defining the FL's performance loss, the objective is then to minimize ε, ideally achieving:

$$|V_{SUM} - V_{FED}| < \varepsilon \approx 0$$

FL can be categorized into horizontal FL (HFL) and vertical FL (VFL). HFL applies when each party has datasets with the same feature and label spaces but different sample sets. In contrast, VFL is suitable when parties possess data on the same samples but with different features or label spaces, allowing them to combine distinct but complementary information about each sample to train a unified model effectively. As shown in Figure 2.14, FL architectures can also be classified as either centralized or decentralized. In centralized FL, a designated aggregator coordinates the updates from all participants and consolidates them into the global model. By contrast, decentralized FL (peer-to-peer FL) allows each party to share model updates directly with peers, following consensus protocols that define model exchange rules and manage redundancies.

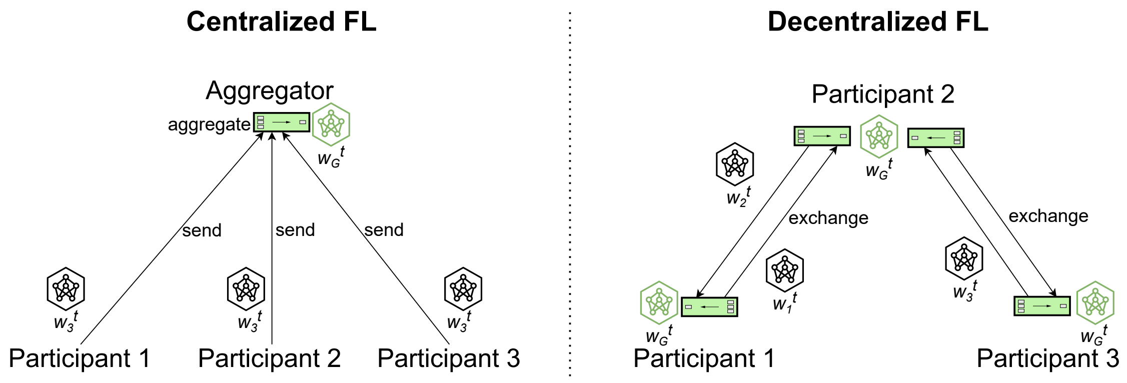


Figure 2.14: Centralized and decentralized FL architectures.

Federated Reinforcement Learning (FRL) applies FL concepts to RL, with Horizontal FRL (HF-RL) and Vertical FRL (VFRL) as analogous categories to traditional FL. FRL methods enable agents to learn from distributed environments with similar properties, such as edge domains in a telecommunication network, thereby improving policy performance and robustness by aggregating insights from diverse state-action-reward dynamics across different edge environments, while respecting the edge constraints inherent in distributed data settings. For this reason, and given the shared challenges between FL and multi-agent deep-RL, namely efficient multi-party training and secure model-sharing, the integration of FL into deep-RL has become a research field of interest in this thesis and an area of active research.

2.5 Summary of the Thesis Fundamentals

This chapter established the foundational knowledge necessary for understanding the key contributions of this thesis. In Section 2.1, we reviewed the architecture of 5G networks, focusing on their distributed MEC layout within edge and core domains. We also examined ETSI NFV's virtualization paradigm, detailing VNFs and CNFs and their flexible management through VMs and container runtimes, which enables the application of MTD strategies as an added security layer in telecommunication networks. Section 2.2 provided an overview of MTD, which this thesis integrates into 5G/B5G networks to enhance network defense and resilience against both proactive and reactive threat scenarios. The thesis also focuses on optimizing the deployment of implemented MTD operations, addressing most of the optimization challenges stated in Section **??**. This includes the data challenge (*i.e.*, gathering data for near real-time monitoring, modeling the state of the network, and quantifying the optimization objectives) and the challenge on multi-tenant networks (keeping data private in optimization's collaborations and maintaining self-managed orchestration).

In Section 2.3, we introduced the LiMi concept, underscoring its value in virtualized environments and the increasing adoption of container-based LiMi. The thesis leverages LiMi as an MTD mechanism specifically to secure VNFs/CNFs, neutralizing intrusions and malware infections for stateless network functions, and isolating infected stateful instances upon detection. Finally, Section 2.2.2 explored machine learning, with a focus on deep reinforcement learning, as a framework for optimizing decision-making challenges relevant to MTD strategies in dynamic network contexts. Specifically, in Section 2.4.3, we introduce FL, *proposed in this thesis as a means to distribute MTD strategy optimization across operators and throughout the multiple edge domains within a shared 5G/B5G infrastructure.*

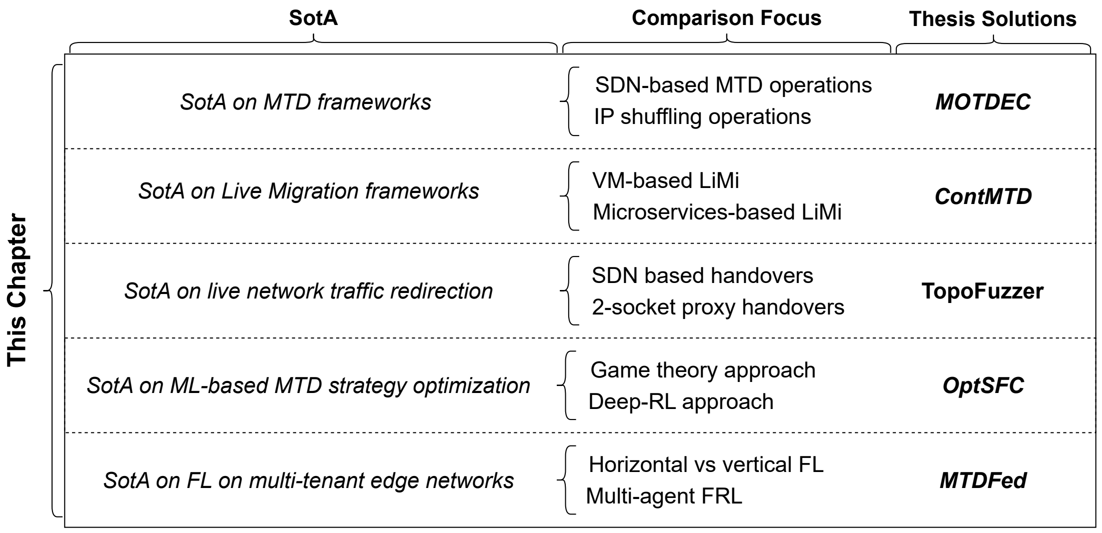
3
Literature Review

This chapter reviews the literature and the SotA for the solutions proposed in this thesis. The primary contribution of this thesis is a comprehensive framework for integrating cognitive MTD within 5G/B5G networks, providing a procedural methodology and a high-level architecture we name *MERLINS* (see Section 4). This is followed by the design and implementation of five distinct solutions that constitute an implemented version of the *MERLINS* architecture. Accordingly, this chapter is structured into five sections, each dedicated to examining related works and the SoTa approaches pertinent to each solution developed in this thesis, as shown in Figure 3.1.

The SoTa review was conducted based on the established search of the predefined themes, following a search process conducted in two phases:

- ***Phase 1***: The collection of recent and relevant LiMi work was performed based on initial survey papers collected. This phase resulted in the theme-based aggregation of the formerly surveyed papers as the baseline set of approaches.

- ***Phase 2***: This had been extended by conducting our own literature search exploring various digital libraries, such as ACM and IEEE, using their respective search engines as well as Google



SotA	Comparison Focus	Thesis Solutions
SotA on MTD frameworks	SDN-based MTD operations IP shuffling operations	***MOTDEC***
SotA on Live Migration frameworks	VM-based LiMi Microservices-based LiMi	***ContMTD***
SotA on live network traffic redirection	SDN based handovers 2-socket proxy handovers	**TopoFuzzer**
SotA on ML-based MTD strategy optimization	Game theory approach Deep-RL approach	***OptSFC***
SotA on FL on multi-tenant edge networks	Horizontal vs vertical FL Multi-agent FRL	***MTDFed***

Figure 3.1: SoTa Division.

Scholar. Finally, the augmented body of work in this review is presented in a structural manner, *i.e.*, with appropriate groupings and corresponding sections.

Tables 3.1, 3.2, 3.3, 3.4, and 3.5 enlist the literature reviewed in the corresponding sections for their respective themes. Section 3.1 examines traditional SDN/vNIC-based approaches (e.g., IP/-port shuffling), grounding the MOTDEC solution. Section 3.2 describes the works made to advance service LiMi, covering both VM LiMi and container LiMi, relevant to the ContMTD solution. Section 3.3 identifies the various methods used for network traffic redirection, particularly in the context of seamless session handover, which forms the foundation for the TopoFuzzer solution. Section 3.4 covers works on game theory and ML-based MTD strategy optimization, including Multi-Objective RL (MORL), which is used in the OptSFC solution for the MTD optimization method developed in this thesis. Finally, Section 3.5 presents FL-based optimizations in edge and telecommunication networks, relevant to the MTDFed solution.

3.1 SotA on MTD Frameworks

Several works have been conducted on MTD mechanisms for traditional networking infrastructures based on the OSI networking layers, particularly focusing on IP shuffling (both in the IPv4 and IPv6 ranges) and port shuffling at the network layer, as summarized in Table **??**.

IP Randomization: Jafarian et al. [82] proposed OpenFlow Random Host Mutation (OF-RHM) based on IPv4 addresses, which implements a shuffling technique that keeps the real IP addresses of hosts unchanged but assigns a random virtual IP address for each host inside the network. OF-RHM is implemented on Mininet as an OpenFlow-based SDN solution that selects IP addresses randomly based on their weight. Every time a currently unused IP address is scanned, its weight increases by one, since an already scanned IP is less likely to be scanned again by an attacker. The interval in which virtual IP addresses are changed depends on the total size of the allocated address range inside the specific subnet. This work is especially security-oriented, focusing on worm and external scanner threats. For the external scan, an IPv4 class B network with 2^{10} possible hosts was used. Not more than 1% of the IP addresses to protect have been discovered by the attacker. Overhead is briefly addressed based on the flow table size, but performance-relevant measurements, like the effective bandwidth, have not been reported.

IP Randomization and Port Hopping: Sharma et al. [83] proposed Random Host and Service Multiplexing (RHSM) using IPv4, which uses an IP Shuffling and Port Shuffling technique to hide the legitimate hosts inside the network. They use a multiplexing method, where multiple virtual IP addresses and virtual ports are used for the host's communication with each other. The implementation was built on an SDN network. Their attack model assumes that the attacker exists outside the SDN. Their work aligns with the focus of the work in [82]. Overhead is based on the size of the flow table. The focus is on the domain of IP address and port-based scanners, mirroring a thematic parallel with our work. Luo et al. [84] proposed Random Port and Address Hopping (RPAH) based on IPv4, which deploys an IP Shuffling and Port Shuffling technique to create obfuscation so that an attacker cannot easily locate a target system. However, it had been deployed for traditional networks, and, therefore, SDN was not used. The host's IP address and port dynamically change through a pseudo-random function that incorporates a secret key, the source's identity, the type of service, and a timestamp as input variables. The attack model contains internal and external adversaries as well as internal and external network scanners, SYN flooding attacks, and worm propagation. However, the resulting scan success rate of nearly 0% is overly optimistic. In addition, the focus is on performance based on throughput and latency measurements. Luo et al. [85] proposed the Test Access Point (TAP) based Port and Address Hopping (TPAH) based on IPv4, which is similar to RPAH. TPAH is carried out within a universal virtual-network kernel driver TAP for a general and multiplatform deployable port and address hopping mechanism. It implements the IP address and port selection as an output of a pseudo-random function and is based on time synchronization, while its input consists of the shared secret key, time, port, and address variance range. This work focuses on the detection and mitigation of port scanning and Denial of Service (DoS) attacks. Furthermore, it conducts latency measurements to assess the performance.

More related to the Telco Cloud infrastructure of 5G/B5G networks, Aydeger et al. [86] explored the use of an MTD framework exploiting SDN and NFV to prevent Crossfire DDoS (Distributed Denial of Service) attacks and redirect traffic to virtual shadow networks for deception. They demonstrate the efficacy of SDN-based MTD mechanisms to disrupt reconnaissance attacks with negligible costs in resource consumption and acceptable network overhead in use cases of videophone applications (with delay requirements of 150 ms). Similarly, Rawski et al. [87] explored the use of SDN and NFV to implement MTD techniques, allowing to change the logical network topology and dynamically attach security VNFs (such as monitoring probes or firewalls) to respond to detected threats. In both studies, MTD actions primarily modify the network's topology and traffic flow using SDN rules. The mitigation of the attack is planned with the deployment of a new security VNF, such as a firewall. However, the protection of targeted VNFs by directly moving them or re-instantiating them is still not fully explored. These recent research directions with MTD Proof-of-Concepts (PoC) available apply to emulated environments, leaving research questions unanswered, especially on a practical impact for 5G Key Performance Indicators (KPI).

Table 3.1: Comparison of related work (I-S: IP Shuffling, P-S: Port Shuffling, PoC: Proof of Concept, S: Security, P: Performance).

Papers	**I-S**	**P-S**	**Deployed PoC**	**IP version**	**Network type**	**Analysis**
[82]	✓	✗	✓	IPv4	SDN	S
[83]	✓	✓	✗	IPv4	SDN	S
[84]	✓	✓	✓	IPv4	trad.	S + P
[85]	✓	✓	✓	IPv4	trad.	S + P
MOTDEC Soft MTD Actions (this thesis)	✓	✓	✓	IPv6	SDN	S + P

3.2 SoTa on Live Migration Frameworks

This section covers the literature works on LiMi, dividing them into VM-based LiMi and microservices-based LiMi, which are detailed in Sections 3.2.1 and 3.2.2, respectively.

3.2.1 Live Migration of VM-Based Network Functions

Works like [37] and [36] leverage MTD against vertical movements and data leakage threats (via side-channel attacks) that exploit isolation weaknesses in hypervisors managing VMs and containers. A similar approach, proposed by Moon et al. [35] and named NOMAD, employs VM migration based on a formal model of information leakage. This model considers factors that increase leakage rates, including the number of service replicas and potential collaboration between attackers (or maliciously controlled devices). Wang et al. [88] tackle both side-channel attacks and DDoS attacks on VMs and propose an MTD framework made cost-effective by using a renewal reward theory-based algorithm (RRT) to evaluate the LiMi costs and determine when to perform the LiMi to minimize such costs.

Various works, in fact, tackle the VM LiMi optimization problem by considering different factors in the decision strategies. Torquato et al. [89] present MTD operations based on VM live migration, offering a decision model that considers the security gains in terms of ASP against the QoS overhead in the downtime and service unavailability of the migrated service. They present a stochastic reward net (SRN) model for evaluating the probability of insider attack success in an Infrastructure-as-a-Service (IaaS) cloud. Nonetheless, in this work as well, the SRN model consists of a conceptual attack graph that is not reflective of the actual network's condition and the practical ASP that each service faces based on the system it runs.

This thesis takes the research question of how to quantify the security benefit of proactive MTD operations a step further in a more empirical and practical manner. Its contribution is also the optimization of the MTD decision strategy by considering *1)* the types of MTD operations, and *2)* by modeling the decision problem into a multi-objective optimization problem that not only considers improving security against another factor, but takes multiple factors into consideration (specifically, the improvement of the QoS/QoE and the reduction of operational costs of the MTD operations.

3.2.2 Microservices-based Live Migration

Checkpoint/Restore In User-space (CRIU) is an open-source Linux-based library developed by Virtuozzo, which also operates the VM hypervisor OpenVZ [90] and is a de facto standard for containers' Live Migration. CRIU is exclusively implemented in the user-space, ensuring its applicability across various Linux-based operating systems. It has evolved to become the de facto standard for container LiMi, seamlessly integrating with major container management and orchestration systems. At the low level of a container runtime, such as runC and LXC (Linux Container), CRIU enables pre-copy, post-copy, and hybrid optimizations.

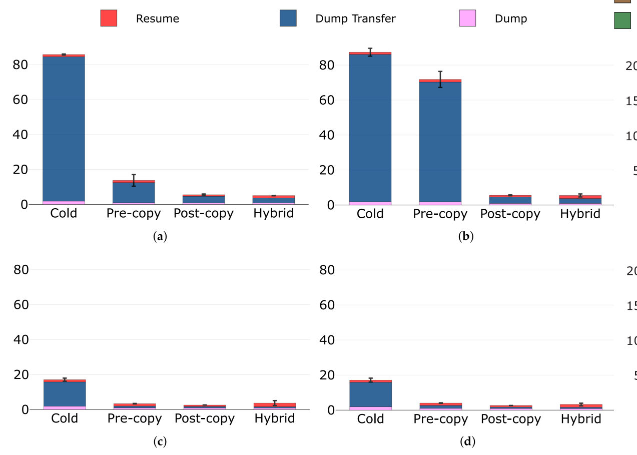
At a higher container orchestration level, CRIU has been incorporated into Docker as a beta feature, but it is worth noting that, as of the time of writing this survey, this integration does not yet support the utilization of optimization methods such as pre-copy and post-copy. In contrast, LXD and Podman integrate CRIU with the pre-copy optimization, while a more recent integration of CRIU as a beta feature has been implemented on Kubernetes [91]. Nonetheless, similar to Docker, only the standard/cold LiMi method without optimization features has been integrated. Kubernetes, arguably the most popular and used container orchestrator, has also presented a new method of live migrating containers running as StatefulSets in a Kubernetes environment [92].

With CRIU established as the leading container runtime LiMi library, recent research has shifted towards expanding its accessibility. This focus on democratizing container LiMi aims to make it a viable option to deploy in critical applications and corner cases where container LiMi was not possible: *e.g.*, containers that deploy Remote Direct Memory Access (RDMA) protocol [93], containers using the Stream Control Transmission Protocol (SCTP) [94], containers using GPU CUDA cores in AI application [95], or again containers running in Trusted Execution Environments (TEEs) [96].

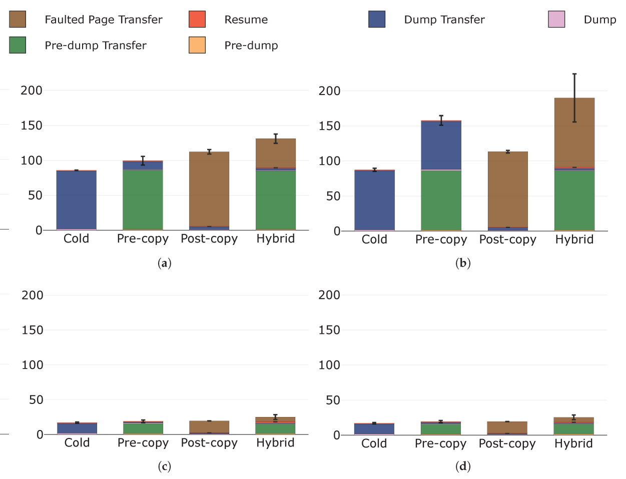
In the telecommunication context, Schiller et al. [97] introduce a CRIU-based LiMi framework to migrate a running CNF. Specifically, the authors migrated the Baseband Unit (BBU) in two types of mobile networks, namely Long Range Wide Area Network (LoRaWAN) and Long Term Evolution (LTE) mobile networks. They observed that while migration is possible within LoRaWAN networks, it is not possible in LTE networks due to the BBU usage of the SCTP protocol in such networks. This issue is later addressed by Ramanathan et al. [94], which also enables the LiMi of the 5G core central unit (CU), containerized as a CNF, that uses SCTP to exchange signaling messages with other 5G NFs. The authors have experimentally validated their implementation in a federated testbed environment, employing the three LiMi optimization methods (*i.e.* pre-copy, post-copy, and hybrid) and a VM-based CU counterpart for a comparative study. Their performance analysis concludes that the hybrid migration method decreases the downtime of the CU by up to 36% compared to the basic migration. Moreover, compared to state-of-the-art pre-copy VM migration, the container-based pre-copy migration reduces migration time by 96.7% and downtime by 75.2%, highlighting the advantages of containers over VMs.

Kaur et al. present the LiMi of microservices between two remote Kubernetes clusters in the context of migrating CNFs of a Magma 5G core in an NFV-based TelcoCloud network [98]. This work employs the stateless migration method, where a new instance is started on the destination node while the source node continues to run the older instance.

Machen et al. [99] present a proximity framework to live migrate different services, both encapsulated in VMs (KVM) or system containers (LXC), from one edge node to another following users' proximity, to reduce communication latency and improve QoS/QoE. The authors use a layering approach similar to the pre-copy optimization to reduce the service downtime and present the advantages of live migrating containers over VMs for different types of services: *i.e.*, face detection, video streaming, game server, RAM-intensive app, and empty shell (*i.e.*, container or VM). In the best-case



(a) LiMi downtime measurements.



(b) LiMi total time measurements.

Figure 3.2: Measurements of LiMi without optimization (*cold*) and with three optimization methods (pre-copy, post-copy, and hybrid) from Puliafito et al. [100].

scenario (the face detection service) and worst-case scenario (the RAM-intensive service), containers have 29-fold and 4.7-fold shorter downtimes, respectively, compared to VMs.

Puliafito et al. [100] evaluate the performance of the four different LiMi methods on containerized services in two different fog network scenarios: a mobility scenario with very low throughput and high round-trip time (RTT) for node-to-node 4G/LTE LiMi, and a management/orchestration scenario using a bridged LAN with higher throughput and lower RTT. Each scenario is tested with high memory workload (*i.e.*, high page dirtying rate) and low memory workload (*i.e.*, low page dirtying rate), making for four different use cases: presented in Figure 3.2: subfigures *(a)* and *(b)* present results in a 4G network with 11 Mbps of Throughput and 122 *ms* RTT. Configurations *(c)* and *(d)* present an average network with around 72 Mbps of throughput and 7 *ms* RTT. *(a)* and *(c)* have a low memory workload while *(b)* and *(d)* have a high memory workload. Their results show that the optimization methods have similar improvements in downtime compared to the basic/cold migration. With the mobility scenario having a throughput as low as the page dirtying rate (around 11 Mbps), the pre-copy method loses its advantage over the basic migration, as the dump phase transfers almost all the memory. This indicates that the post-copy and hybrid methods are more suitable in low-performance networks with memory-intensive applications. Nonetheless, this challenge is gradually diminishing with the continuous enhancement of current 5G networks and the anticipated advancements in future mobile networks. To illustrate this, a simple practical test of the Swiss 5G connectivity is conducted using the Swisscom operator's network. The test involves establishing a 5G connection

Table 3.2: Works on LiMi as MTD.

Papers	**VM/ Containers**	**Method**	**Tech**	**Use Case**
Azab et al. [101]	Containers	MTD	LXC	Evasive LiMi as a proactive MTD mechanism
MIGRATE & RELOCATE [36, 37]	Containers	MTD	Docker	MTD against vertical movements and border information leakage in public cloud
NOMAD [35]	VM	MTD	KVM	MTD against public cloud side channel attacks
Wang et al. [88]	VM	MTD	KVM/ vSphere	LiMi as MTD against DDoS and Covert Channel Attacks
MOTDEC Hard MTD Actions (this thesis)	VM	MTD with MORL optimization	runc/CRIU	Stateless VNF LiMi and reinstantiation
ContMTD (this thesis)	Containers	MTD with MORL optimization	ETSI OSM / Openstack	stateful CNF LiMi and parallel microservices LiMi

from a mobile device located in Winterthur to a Swisscom endpoint in Lausanne, covering a distance of 240 km. The test reveals a throughput of 660 Mbps with a latency of 21 ms. Compared to the mobile network performance used in [100], a substantial 60-fold increase in throughput is observed, coupled with a five-fold decrease in latency, all while the distance between the two endpoints is a hundred kilometers longer. In this thesis, we re-evaluate the different optimization methods based on the new 5G network performances and consider different types of services, developing a novel ML-based LiMi framework that selects the most efficient LiMi method based on the nature of the service to migrate (further details in Chapter 4, Section 4.3).

The adoption of MTD with container LiMi is being recently explored in various security threat scenarios more pertinent to generic cloud-centric networks rather than specific telecommunication networks. Azab et al. [101] quantify the probabilistic advantage of LiMi for MTD by considering scenarios where services are migrated across cloud hosts, some of which may be compromised. Their simulations compare the effectiveness of MTD using LiMi when the number of compromised hosts is either static or dynamic, and they factor in IDS response time. The results demonstrate that container LiMi significantly reduces the probability of service compromise compared to a non-moving container.

3.3 SoTa on Live Network Traffic Redirection

The redirection of client traffic when migrating a service can be achieved through various methods, which vary depending on multiple factors as defined in the taxonomy of traffic relocation for LiMi introduced in Chapter 2, Section 2.3.1. A set of relevant works presenting such methods is presented in Table 3.3 and further detailed in the following paragraphs.

Bradford et al. [102] implement a Xen VM migration mechanism that allows such live connections to reach the new instance using IP tunneling, while new connections find the new instance's IP with dynamic DNS. This redirection method incurs a service disruption of approximately 1.04 seconds, excluding the time for stateful migration. A potential drawback of this approach lies in the exposure of the new IP address change to end users. This visibility can be detrimental as it allows attackers to know when an instance is migrating and to discern the IP range used for the service shuffling. By exploiting this knowledge, targeted attacks could be launched that render defensive migrations ineffective. Moreover, IP tunneling requires the original source host to remain operational until all existing connections are terminated. While this poses minimal impact for web services with short-lived connections, the scenario is significantly different for long-lived connections commonly found in IoT and live streaming applications. In these cases, the extended persistence of prior connections can lead to an unwanted increase in resource consumption, as the source node cannot be hibernated, as well as a decrease in QoS due to the additional step in the data path to reach the new instance.

Fahs et al. [103] present *Proxy-mity*, a proxy that replaces Kubernetes's standard kube-proxy for proximity-aware load balancing. For this, authors use *iptables* NAT rules, which are limited to redirecting only new connection requests; established connections continue to use the old IPs and ports until they are disrupted by the service instance termination in the source node.

Teka et al. [104] use multipath TCP (MPTCP) for seamless live VM migration between neighbor cloudlets. MPTCP allows the simultaneous use of multiple IP addresses for a single regular TCP session, providing an alternative solution to avoid the re-establishment of the TCP connection. This approach hinges on two critical preconditions: *1)* prior knowledge of the private IP address allocated to the new instance, and *2)* enabling the services to use MPTCP. The latter requirement poses a significant challenge for cloud and telecommunication operators, as it necessitates control over the service

Table 3.3: Traffic relocation work.

Papers	VM/ Containers	Method	Tech	Use Case	Objective
[48]	VM	Layer-2 overlay	Xen	LAN LiMi	LAN availability
[53]	VM	lightpath	Xen	Multi-cluster (MAN/WAN) adaptation of [48] on optical networks	MAN/WAN availability
[102]	VM	IP tunneling & DynDNS	Xen	Multi-cluster LiMi using IP tunneling and DynDNS	WAN availability
[103]	Container	NAT	k8s	Proximity-aware load-balancing for microservices	Load-balancing
[104]	VM	MPTCP	KVM	VM WAN LiMi between cloudlets	MAN LiMi
[105]	Container	SDN/P4/ SRv6	CRIU	VM WAN live IPv6 edge workload migration	WAN IPv6 LiMi
[106]	Container	Layer 7 (app)	CRIU	Live stream fog service	Edge availability
[107]	VM/ container	SDN	*n.n.*	Handover of long-lived TCP sessions	Seamless TCP migration
[108]	VM/ container	TCP-repair	*n.n.*	Handover of long-lived TCP sessions	Seamless TCP handover
TopoFuzzer (this thesis)	VM/ container	vNICs 2-socket proxy	*n.n.*	Handover of long-lived TCP/QUIC-based sessions	Seamless long-lived session handover

binaries running on the machines. In these scenarios, the services might be owned by third-party clients, hindering the operator's ability to enforce MPTCP adoption.

Harney et al. [109] utilize Mobile IPv6 to maintain the same IP address, thereby avoiding changes to the DNS, but with a service disruption of 8 seconds (excluding downtime due to the service migration itself). A more recent work in the same direction, from Lemmi et al. [105], addresses IPv6 workload migration of an edge service that maintains the same IP address by redirecting IPv6 traffic using SDN and state-of-the-art P4 network programmability [110] and the SRv6 protocol [111].

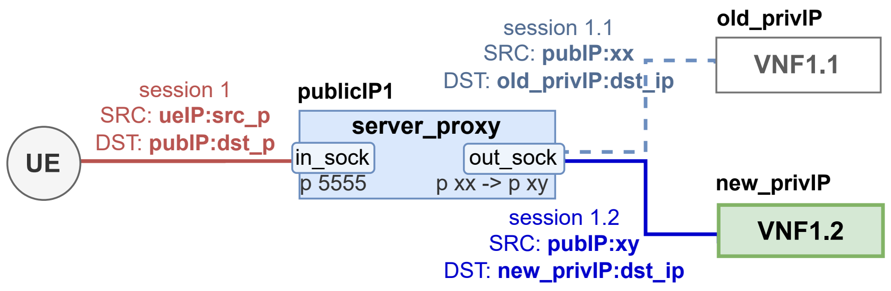


Figure 3.3: 2-socket proxy redirection.

SRv6 follows the segment routing paradigm and adds a segment routing header (SRH) to the IPv6 header that allows to identify transit routers and dynamically performs IPv6 forwarding to migrate the network. Lemmi et al. additionally implement a buffer mechanism to keep packets sent during the CRIU container migration. Their results show the best position to initiate the buffer is at the destination node, allowing a lossless traffic redirection, while the service downtime due to CRIU's checkpoint/restore of their specific service is 3.28 seconds, which could be reduced by using an optimization method as described in Section 2.3.2.

Puliafito et al. [106] investigate stateless container-based service migration and its impact on the QoS perceived by mobile devices, achieving network downtimes of five seconds (in the experiment, 10 frames are lost in a two FPS live stream fog service). The framework employs a white-box implementation, where the connection handover occurs at the application level in the UE/end-device. This is not feasible in scenarios of cloud service providers and telco operators who would prefer to migrate resources as black boxes, independently of the application's implementation and connected clients.

All NAT-based techniques, such as Hole Punching [112], share the same issue of DNAT with TCP and may expose the private IP address of the VNF when attempting to establish a direct connection. By using SDN traffic redirection, packets are forwarded to the new instance, but they are dropped because they are not recognized as belonging to it. Binder et al. [107] demonstrated that the connection can be transferred to a new server by performing NFQUEUE packet manipulation: synchronizing the TCP sequence number (TCP SEQ), congestion window, SYN/ACK numbers, and checksum values of IP packet headers in the SDN switch at the time of packet forwarding. This "TCP state machine" synchronization is performed by changing all packets directed to the service from the moment the service is migrated until the connection of the client ends, which translates into an important SDN load and the QoS degradation that persists all along the connection life-cycle and is not limited to the first traffic redirection.

Cunha et al. [108] introduce instead a 2-socket proxy to redirect live attack connections to a Honeypot service. The proxy is introduced after the connection with the attacker is established using the Linux kernel's TCP_REPAIR feature, which is also employed by CRIU when checkpointing a container runtime to save the container's network state. This work demonstrates that the 2-socket redirection process has a much lower impact on the traffic's QoS and outperforms SDN TCP handover and NFQUEUE packet mangling, with a continuous redirection latency overhead of only 0.03 ms. As depicted in Figure 3.3, the UE, as the client node, can only see the connection established with the public IP of the VNF. The proxy receives packets to a specific binded port where the `in_socket` is listening. The payload is then extracted and sent to the `out_socket` pipe. When the VNF moves, a new connection is established using the `out_socket` and the payload is sent to the new server. Nonetheless, their solution is proxying only one specific connection to a specific port (*e.g.*, a web server on port 80), which makes it inadequate for VNFs that have multiple ports open. Hence, the proxy should be able to redirect traffic with different destination ports simultaneously. Moreover, a proxy node can only redirect the traffic to one VNF, as it configures its vNIC to use the VNF's public IP to receive client traffic. The solution proposed in this thesis, TopoFuzzer, employs the two-socket proxy method, inspired by [108], and addresses the aforementioned issues of redirecting traffic to a single port and a single VNF. Additionally, it introduces a traffic isolation concept using vNICs to maintain network slice isolation.

3.4 SoTa on ML-based MTD Strategy Optimization

This section covers the works on MTD strategy optimization, with game theory-based (*cf.*, Section 3.4.1) and ML-based (*cf.*, Section 3.4.2) approaches. The latter also reviews MORL as it is used in the MTD optimization method developed in this thesis.

3.4.1 Game Theory Approaches

Optimization of MTD operations in network security scenarios is already attracting significant attention from research and industrial communities. In the past, game theory-based MTD optimization strategies have been developed to counter various attack models in various environments, ranging from Partially Observable Markov Decision Processes (POMDP) in large-scale networks [113] to general-sum Markov Games in cloud networks [114]. In the latter, Sengupta et al. use the attack graph of a cloud network to formulate a general-sum Markov Game and to solve the Stackelberg equilibrium problem, shown to provide an optimal strategy for the placement of security resources to protect cloud systems.

3.4.2 Deep-RL and MORL Optimization

Table 3.4: Recent works and their scope on MTD optimization.

	Environment	Approach	Deep-RL	5G/ B5G	Edge-to-Cloud Security	Game theory	Multi-objective
Online Defense Algorithm [113]	Theoretical model	POMDP				✓	
Markov Game Model [114]	Cloud testbed	General-sum Markov Games			✓	✓	
DQ-MOTAG [115]	Cloud-hosted web app.	DQN	✓			✓	
DESOLATER [116]	In-vehicle network with SDN	Multi-agent deep-RL	✓		✓	✓	✓
CyberBattleSim [117]	Network simulation	MDP & deep-RL	✓			✓	
OptSFC (this thesis)	5G testbed	MORL, FL	✓	✓	✓	✓	✓

Research on AI/ML-optimized MTD is relatively recent, aligned with game-theoretical methodologies that have a more established historical foundation and remain a necessary component to tackle the optimization problem with deep-RL. DQ-MOTAG [115] employs MTD IP shuffling to mitigate ongoing Distributed DoS (DDoS) attacks on web servers, against DDoS attacks proposed in [118], extending another MTD solution from the literature [118] and optimizing it with Deep Reinforcement Learning (DRL). It finds the original hidden malicious user giving up the server's IP to other active attacking hosts with a minimal number of shuffles using deep-RL. DESOLATER [116] presents a multi-agent deep-RL method to train an MTD shuffling strategy of the IP address layout in in-vehicle networks with SDN capabilities. Microsoft has also contributed to advancing this research direction for local and enterprise networks by releasing the open-source research toolkit "CyberBattleSim" [117]. However, this toolkit does not allow the modeling of complex environments like Edge-to-Cloud systems.

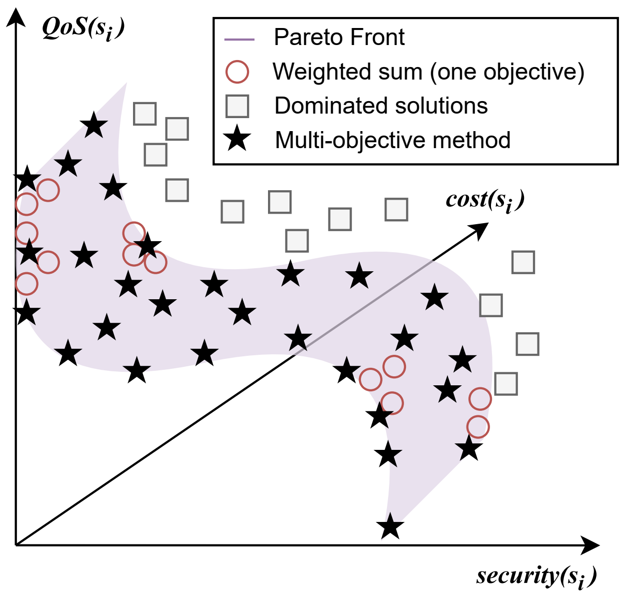


Figure 3.4: Pareto front for three objectives showing the benefits of MO methods over weighted sum optimization methods (Adapted from [119]).

Conventional deep-RL algorithms, however, are designed for single-objective optimization and used with the scalarization of the different rewards corresponding to various objectives into a single reward value. In that case, this scalarization can be part of the missing information we want to learn, *i.e.,* the best trade-off among objectives to maximize the overall return. If the optimization occurs for only one fixed weighted sum, the result produced would be suboptimal, as other weight sums are not explored. The scalarization can also depend on a user's preference, which makes the weights a variable that should be controlled. With legacy deep-RL, this is not possible unless a different model is trained with new weights. Moreover, the single scalarized value can be semantically meaningless as the multiple objectives are fundamentally different in nature. For instance, one objective can be to reduce an economic cost metric, measured in a monetary unit, while another objective is to improve proactive security, which may be measured in terms of the ASP reduction of a threat. Merging both measures into a value that is difficult to interpret leads to decisions that are equally challenging to explain/interpret, whereas the explainability of the deep-RL MTD policy is a critical requirement for important technical and societal aspects, such as liability and resilience.

MORL is a new category of RL algorithms that keeps different interpretable reward functions, one for each objective, and iterates the optimization process on different weighted sums, avoiding sub-optimal solutions and approximating the set of optimal policies for all scalarizations (see Figure 3.4). This solution set is defined as the *coverage set* (CS), which, for monotonically increasing reward functions, is reduced to the *Pareto Front* (PF) [120]. PF is the set of undominated solutions, where each solution is optimal with respect to a specific scalarization. MORL's interactions to retrieve the PF occur with a multi-objective MDP (MOMDP), where the main difference with respect to the MDP's definition is that R is now a vector $\overline{R}$ comprising the reward values of the multiple objectives defined in the model. These objectives do not always align with one another, as optimizing one objective may be done at the expense of others. In the scenario of MTD operations in the edge-to-cloud continuum, moving a resource to a closer node may improve latency, but it may also weaken security since its position becomes predictable to attackers. Conversely, a completely random move aimed at enhancing security could negatively affect the network and service performance. Therefore, in contrast to legacy deep-RL models, MORL allows one to learn the optimal trade-offs and enables the user to set his own preferences in terms of objective prioritization.

3.5 SoTa on Federated MTD Orchestration

While MTD has been used to secure FL training against poisoning attacks and model inversion attacks, especially in distributed FL setups [121–123], the use of FL to optimize MTD operations made to shuffle cloud-native environments has not been studied yet. Nonetheless, FL has commonly been used in telecommunication networks and edge computing, particularly for applications in anomaly detection, resource allocation, and edge computation offloading. In the area of anomaly detection, Sahu et al. [124] employ a CNN-LSTM combined model for identifying cyberattacks. A CNN extracts complex data features, while an LSTM classifies them by identifying temporal patterns vital for intrusion detection, improving the detection of cyber-attacks. As NN-based models rely heavily on large datasets, often containing sensitive and private information, FL models have been introduced to improve confidentiality, enabling decentralized cyber-attack detection across edge-computing (EC) networks without requiring direct data sharing. This FL-based approach preserves data confidentiality while maintaining robust accuracy in distributed cyber-attack detection. Abeshu et al. [125] introduced a distributed deep learning approach for cyber-attack detection in fog-computing environments. This scheme was evaluated based on key performance metrics, including accuracy, detec-

Table 3.5: Works on FL and 5G/B5G management optimization.

Papers	Environment	Method	Use Case
Beltrán et al. [101]	MEC/IoT	DFL, MTD	Mitigating communication-based security threats in DFL: eavesdropping, Man-in-the-middle, network mapping, and eclipse attacks.
Feng et al.[122]	generic networks	DFL, MTD	Mitigate poisoning attacks in DFL.
Zhou et al. [123]	IoRT	Cross-silo FL, MTD	Preserve privacy & integrity in cross-silo IoRT (resist poisoning).
Wang et al. [126]	MEC/Edge	FL + Deep-RL	Joint optimization of MEC: caching, offloading, and communication.
Abeshu et al. [125]	Fog/IoT	CNN	Distributed attack detection in fog-to-things / IoT.
Qi et al.[127]	Wireless edge	FL	Local popularity prediction for proactive caching at wireless edge.
Han et al.[128]	Edge/IoT	FL + Deep-RL	Offloading optimization for IoT-edge.

tion rate, and scalability, to assess the model's effectiveness in identifying attacks across decentralized networks.

In MEC environments, Deep-RL has become a preferred centralized ML approach for optimizing resource allocation and computation offloading policies, as seen in various studies [129–132]. FL has been leveraged to further decentralize the decision-making and increase confidential data sharing among parties. For instance, to optimize content caching and task offloading strategies, FL aids in determining whether to cache specific content, as well as identifying the optimal timing and methods for offloading computations. These decisions play a crucial role in maximizing the efficiency of both communication and edge computation, influencing services and system performance in terms of QoS and QoE. Agents in constrained edge nodes assess whether to replace local content based on the learned popularity trends derived from the distribution of content requests. This popularity reflects shared user interests across different edge nodes and operators involved in the FL process, enabling more informed and synchronized caching decisions [126, 127, 133].

Deep-RL has been used in MEC-based telecommunication networks for context-aware computation offloading to find the optimal trade-off between energy consumption, execution cost, network usage, and fairness, and FL is applied on distributed deep-RL agents to reduce the network load and transmission costs between agents [126, 128, 134]. Wang et al. provide one of the first studies coupling Deep-RL with FL in MEC networks to optimize both edge caching and edge computation [126]. These methods see a great potential in optimizing MTD proactive strategies in 5G/B5G networks with similar MEC setups, and to the best the author's knowledge, this thesis is the first to tackle such multi-objective optimization problem using both MORL and FL.

3.6 Key Findings and Takeaways

This chapter reviews the related works and literature underpinning the various solutions designed and proposed in this thesis. The first section (*i.e.*, Section 3.1) covers MTD techniques focused on traffic redirection and dynamic endpoint reconfiguration, including IP and port shuffling. These methods are later termed "soft-MTD" operations, as they affect only network interfaces, socket configurations, and packet redirection, without impacting service instances or the underlying infrastructure. In contrast, the second section (Section 3.2) discusses more pervasive techniques, which will be referred to as "hard-MTD" operations, specifically LiMi of VMs and containers. This section highlights the growing significance of container LiMi and introduces optimization algorithms designed for various container types and service requirements. It is observed that there is no one-fits-all optimal algorithm, making it necessary to have a solution that dynamically selects the suitable algorithm according to the container type, especially in microservices applications where interdependent containers belonging to one service migrate in parallel. The ContMTD framework developed in this thesis presents such a solution in the subsequent chapters. The discussion on LiMi continues in the following section (Section 3.3), focusing on the challenge of maintaining traffic continuity during migrations, especially with session-based protocols like TCP. Various solutions have been explored, yet difficulties remain when a service's IP address or port changes mid-session. The TopoFuzzer solution developed in this thesis addresses this technical challenge directly.

Section 3.4 addresses the optimization of decision-making problems, covering studies using game theory approaches and later derivatives using RL and deep-RL, having as a common denominator the modeling of the network into an MDP. The challenge of balancing multiple objectives is identified, leading to the literature study of recent MOMDP and MORL algorithms, which, to the best of the author's knowledge, have never been applied in decision-making optimization problems regarding 5G/B5G orchestration, in general, and security and MTD-based strategies, specifically. Finally, Section 3.5 gives a brief SoTA literature review of the application of FL in 5G/B5G networks, especially in MEC settings where multi-party edge distributed computing requires it for a multitude of use cases, including, in this case, the optimization of MTD strategies.

4 The *MERLINS* Approach

This chapter presents *MERLINS*, the MTD-centered approach proposed in this thesis. The *MERLINS* methodology, HLA, and the solutions implementing its various components are the core contributions of this thesis, presenting novel preliminary results that advance research towards the practical mainstream adoption and improvement of MTD as an additional security layer in 5G/B5G networks. *MERLINS* HLA is designed following a systems design approach centered on iterative development principles to simplify the definition of a continuous and autonomous MTD security system. This methodology defines the end-to-end process for monitoring and executing MTD operations within 5G/B5G networks, adhering to the ETSI ZSM standard [135] by leveraging a closed-loop approach to automate security management and orchestration.

This enables continuous improvement and dynamic adaptation of *MERLINS* MTD strategies in response to evolving network conditions and threats.

Depicted in Figure 4.1, the *MERLINS* methodology is composed of four cyclic phases, listed as follows:

- Ⓐ Integration to the 5G/B5G network: This phase consists of an initial setup that occurs only in the first loop of the cycle to define the way the communication with the 5G/B5G network

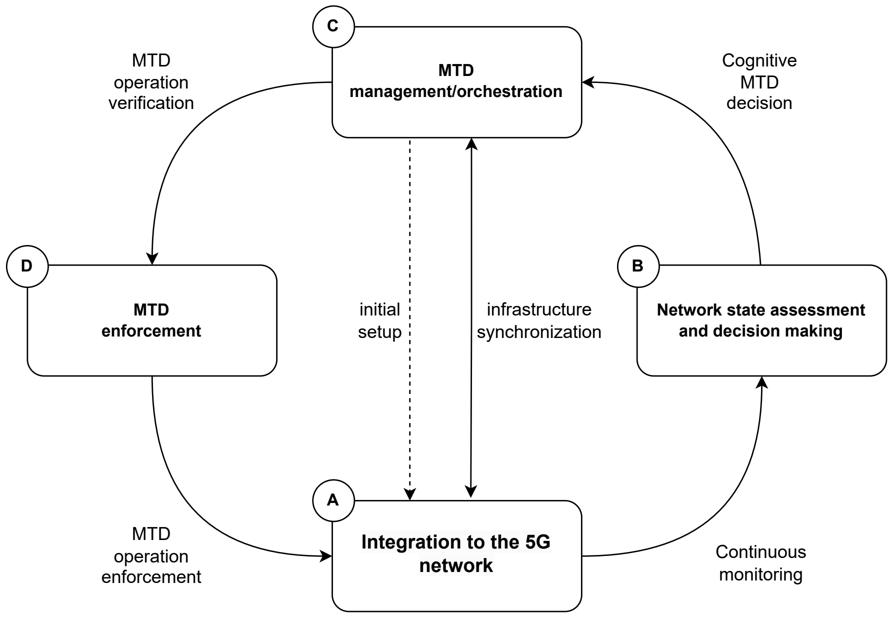

C
MTD management/orchestration
MTD operation verification
Cognitive MTD decision
D
MTD enforcement
initial setup
infrastructure synchronization
B
Network state assessment and decision making
A
Integration to the 5G network
MTD operation enforcement
Continuous monitoring

Figure 4.1: The 4-phases methodology for cognitive MTD orchestration in 5G/B5G applied in the *MERLINS* HLA.

occurs. Then, an infrastructure synchronization is permanently running, involving passive and active interactions with the network. The passive interactions are the consistent and real-time observation and monitoring of the network, while the active interactions consist of the ability to operate on the network components, *i.e.*, the VNFs and VIMs of the different domains in the edge-to-cloud continuum, spanning from the multiple edge clusters or nodes to the core network.

- Ⓑ Network assessment and decision making: Using the data obtained from the passive interactions and monitoring in the previous phase, this phase focuses on analyzing data, such as performing performance evaluations, resource consumption analysis, and security evaluations, to assess risks or detect attacks. This analysis then results in a decision on whether to enforce an MTD operation or not.

- Ⓒ MTD management/orchestration: In the advent of the decision to perform an MTD operation, this phase goes through the validation process, analyzing whether the operation can be performed, with respect to technical feasibility, *i.e.*, if the operation can be implemented on the specified target, and policy-based feasibility, *i.e.*, if there is no other orchestrator with a conflicting policy and a higher hierarchical priority.

- Ⓓ MTD enforcement: At the validation of an MTD operation, this phase enforces and implements the MTD operation on the 5G network, also using the active interactions available in Phase Ⓐ, transitioning to this phase for the next iteration of the cyclic methodology.

The remainder of this chapter is organized as follows: Section 4.1 introduces the *MERLINS* HLA, its layers, and its components. Subsequent sections present the solutions designed and implemented to represent such components of the *MERLINS* HLA, in the following order: the MTD Controller (MOTDEC), the container MTD framework (ContMTD), the network topology fuzzer (TopoFuzzer), the security function optimizer (OptSFC), and the MTD federated framework (MTDFed).

4.1 *MERLINS* High-Level Architecture

MERLINS HLA covers the four phases defined in the methodology depicted in Figure 4.1, using a three-layer structure as presented in Figure 4.2. Each layer of *MERLINS* has components tasked with different operations defined as follows:

1. *The Management and Orchestration Layer (MOL)*: This layer is responsible for the operations of phases A and C of the *MERLINS* methodology (*i.e.*, the integration to the 5G network and the MTD management and orchestration). As *MERLINS* has to evaluate and enforce proactive and reactive MTD actions in a telecommunication network, a proper interface has to be defined with the latter to obtain the configuration of the telecommunication infrastructure, in our case, a 5G infrastructure, and to subsequently operate in its domains, *i.e.*, both in the core and the edge domains. To do so, a northbound interface (NBI) with the NFV Orchestrator and network slice manager is necessary. From the NBI, *MERLINS* MOL also collects various data for the real-time monitoring of the network, necessary for the next *MERLINS* layer to evaluate the attack surface and ASP on NFs to be protected (whether VNFs or CNFs), for proactive MTD operations. Moreover, the monitoring component should also be able to collect anomaly detection and attack detection alerts from other security sources, enabling reactive protection. The MOL also acts as the central system to orchestrate *MERLINS* MTD operations through an *MTD controller*, interfacing the enforcement layer with the 5G network's VIMs and NFVIs. Finally, the MOL is designed to enable federated cooperation among peer controllers distributed in various edge nodes, in the scenario of different *MERLINS* instances owned by network slices or virtual operators who want to improve the effectiveness of their MTD framework collectively.

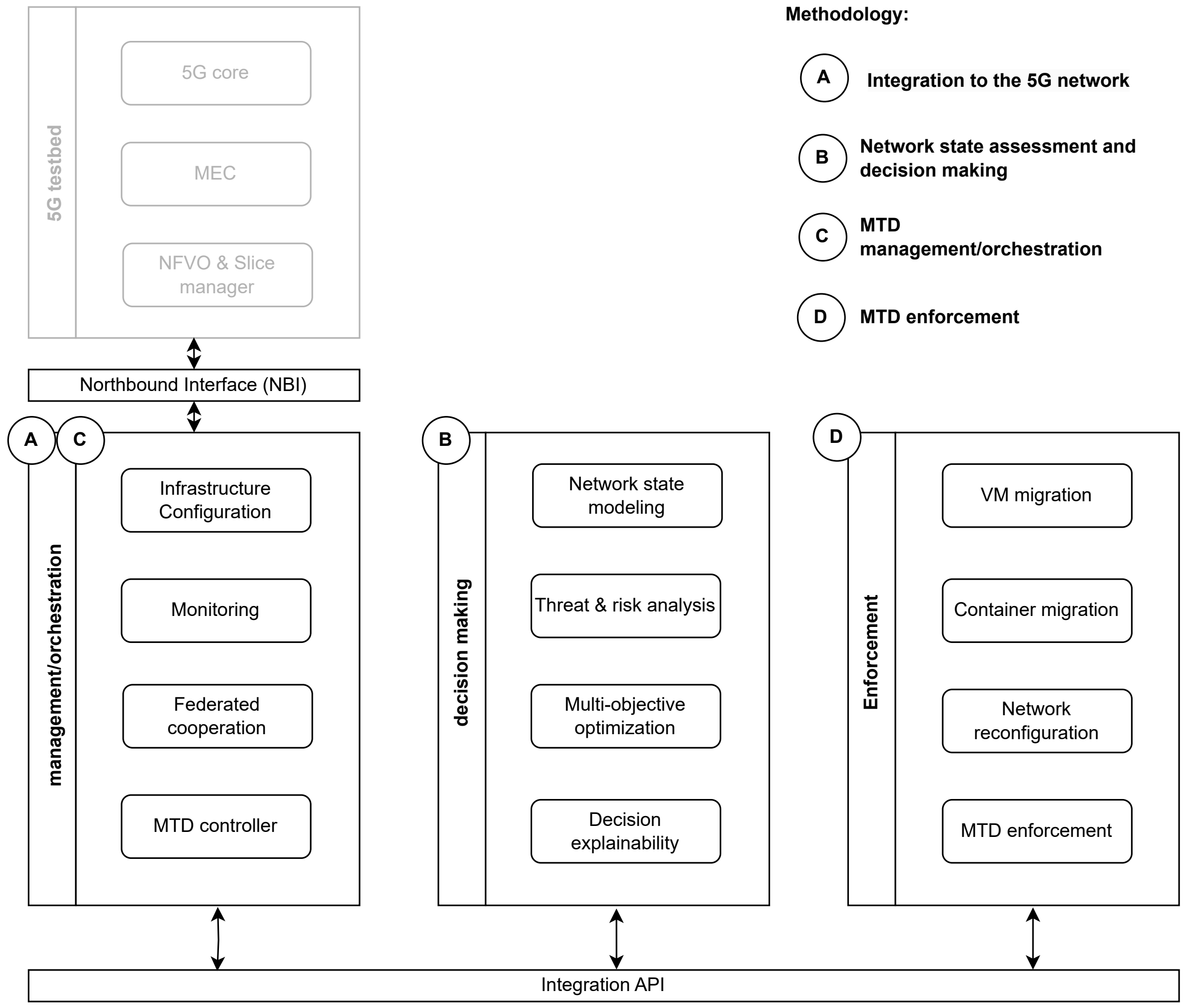


Figure 4.2: *MERLINS* HLA and the mapping of its layers with the phases of the methodology.

2. *The Decision-Making Layer (DML)*: The DML is responsible for the operations of Phase B of the *MERLINS* methodology (*i.e.*, the network state assessment and decision making). As the name indicates, this layer performs all operations required to make informative and automated decisions on which MTD actions to perform and, in the mid-long term, what MTD strategy to follow. These operations, identified by the DML's components, include: network state modeling, threat and risk analysis, multi-objective optimization, and decision explainability. The network state modeling is necessary to make a correct interpretation of the 5G network state using the raw data obtained by the monitoring operation performed in the MOL.

   The threat and risk analysis follows, augmenting and deducing additional metadata based on the raw data. This is equivalent to threat intelligence extracted from vulnerability scanner reports, as well as anomaly and attack detection alerts obtained from detection systems. Such data is then adapted to be integrated into the network state model or used to decide on direct mitigation actions.

The multi-objective optimization operation is required to balance and find the trade-off between various objectives when deciding on the MTD operations to perform, such as optimizing latency and QoS, increasing the entropy of MTD actions for security, or reducing the resource consumption and operational costs of the protected NFs.

The decision explainability is the last component identified in the DML, as MTD decisions affect critical infrastructures and multi-tenant services running in them, thus requiring some levels of robustness, which can be gained when decisions are humanly explainable, and liability, which necessitates motives of why certain operations were made in the network.

3. *The Enforcement Layer (EL)*: The EL is responsible for Phase D of the *MERLINS* methodology, *i.e.*, the enforcement of MTD actions. After the decision on which MTD action to perform is made by the DML, the EL implements the action's mechanisms and executes them on the targeted 5G resources. Such MTD actions are identified as three main operations: VM migration (targeting VNFs), container migration (targeting CNFs), and network reconfiguration (targeting network interfaces, network routing, and network topologies). The MTD enforcement occurs with the coordination of the MTD controller, from the MOL, which coordinates operations imparted to the involved NFs, network slices, NFVIs, and VIMs for the successful execution of the MTD action.

MERLINS HLA intends to consider any feature necessary to the execution of MTD operations during the network's life-cycle, and it could be extended to accommodate specific features: *e.g.*, based on the requirements of new MTD operations in the EL, or changes in the modeling of the network state in the DML. Nonetheless, being a high-level architecture, *MERLINS* is agnostic of the technologies and methods used to fulfill each of the operations defined in its layers and components. This thesis delivers the design and implementation of a suite of solutions that represent an implemented instance of the *MERLINS* HLA to demonstrate its feasibility and its contributions to securing current and future Telco Cloud networks, such as 5G and B5G systems.

MERLINS solutions are used to tackle various threat scenarios, proactively and reactively, based on the MTD operations implemented. For instance, stateless LiMi and reinstantiation actions implemented in the MTD Controller (MOTDEC) are used for proactive defense against undetected intrusion and malware infection, IP and port shuffling implemented in MOTDEC tackle reconnaissance and fingerprinting attacks, while stateful LiMi implemented in ContMTD is used to reduce

Table 4.1: Solutions developed as part of the implementation of *MERLINS* HLA.

Solution	**Description**	**Phase Addressed of the Methodology**	**Layers of the HLA Implemented**
MOTDEC [30, 136]	An MTD controller and stateless VNF migration orchestrator	Phase A, C, and D	Layer 1, Layer 3
ContMTD [33, 137]	An orchestrator for stateful container LiMi	Phase C and D	Layer 1, Layer 3
TopoFuzzer [138]	A networking solution for live TCP and QUIC session handover	Phase D	Layer 3
OptSFC [27, 136]	A MOMDP and deep-RL based decision-making optimizer for MTD strategies	Phase B	Layer 2
MTDFed	A secure and privacy-aware FL approach for peer-to-peer OptSFC agents	Phase A and B	Layer 1, Layer 2

and disrupt data exfiltration rate and lateral movement to sensitive CNFs. These mechanisms do not operate in isolation; instead, they can be sequenced or combined: for instance, a reactive shuffling of ports after an intrusion detection can immediately cut off an attacker's reverse shell, while a subsequent stateful migration of the compromised service to a sanitized node ensures operational continuity and isolates the threat. The framework is designed for extensibility, allowing new MTD techniques such as instruction set randomization for binaries or dynamic application topology mutation to be integrated as new components in the EL and added to the MTD catalog for the MOL and DML. This flexibility enables *MERLINS* to extend to novel attack vectors, but it also introduces challenges to consider, such as the potential for conflicting actions (*e.g.*, a migration triggered during active shuffling). Effectively keeping new MTD strategies operationally stable is one of the central tasks of the *MERLINS* HLA and the designed closed-loop methodology.

Table 4.1 presents a comprehensive overview of the developed solutions, highlighting the phases of the methodology they address and the components of the *MERLINS* HLA they implement. As shown, each phase of the methodology (Figure 4.1) and every component of the HLA (Figure 4.2) is addressed by at least one solution, except for the decision's explainability component, which has been added to the HLA for sake of completeness, but has not been addressed in this thesis as it is out of scope and not considered in the research questions defined in Section 1. It is important to note that these solutions are not intended to serve as a definitive approach to achieving cognitive MTD-based security for Telco Cloud networks. Rather, they offer a foundational framework and valuable insights

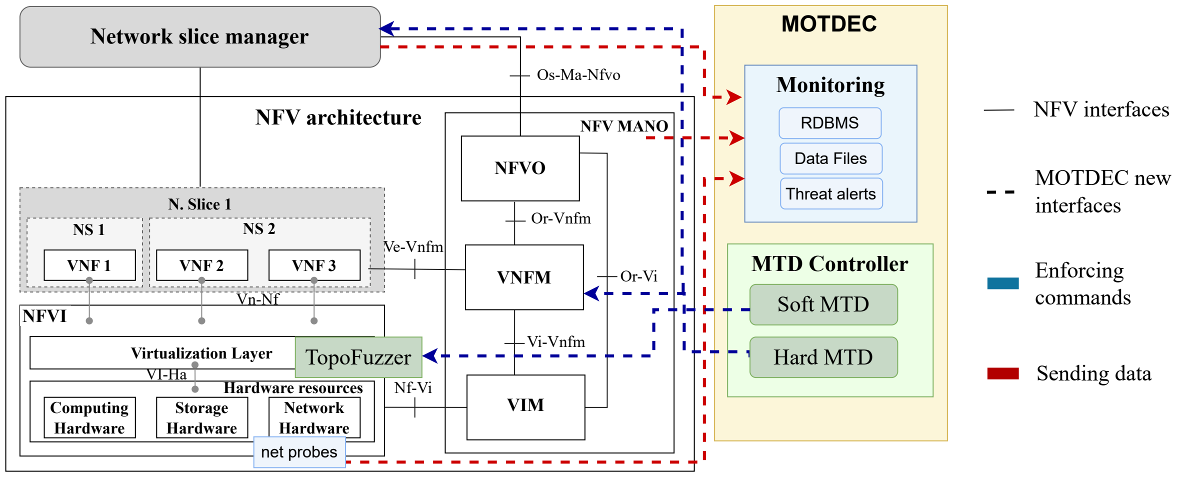


Figure 4.3: MOTDEC architecture.

for enhancing Telco Cloud security through MTD. The following sections will describe each solution individually, providing details of their design, architecture, functionalities, and implementation.

4.2 MOTDEC - an MTD Controller

The MOTDEC framework provides an implementation of the components in the HLA's MOL, namely the NBI with the telecommunication network, the infrastructure configuration, the monitoring, and the MTD controller. MOTDEC targets implementing the fundamental operations of the MOL and tackles the research question ***RQ1***: *'Which MTD actions can be taken on a 5G network and against which attack scenarios, considering the security properties they offer?'*.

4.2.1 MOTDEC's Architecture

As depicted in Figure 4.3, MOTDEC's architecture is designed and implemented to be interfaced with the standardized NFV architecture used in 5G and B5G networks and is mainly formed of two modules: a monitoring module, which synchronizes its information from the network in order to keep a history and a near-real-time observation of the 5G network, and an MTD controller, which enforces the various MTD actions implemented in the framework. The following sections describe MOTDEC's formal positioning with respect to the NVF standard architecture (Section 4.2.2), MOTDEC's interface and monitoring module (Section 4.2.3), and MOTDEC's MTD controller and its implemented MTD actions (Section 4.2.4).

4.2.2 MOTDEC Proposed Integration to the NFV Standard

MOTDEC uses the NFV standard architecture as a reference point to interface itself with the 5G/B5G telecommunication networks. Thus, a detailed study of the standards is necessary, as presented in the earlier foundational chapter of this thesis (Chapter 2). When considering telecommunications standards, the ETSI NFV standards group defines a candidate architecture that could integrate MTD in 5G and Beyond 5G networks. The ISG is currently working on improving the security of such architecture, with additional components like the *OSS/BSS security managers (OSSM)*, *NFV Security Managers (SM)*, and *Security Agents (SA)*. These elements are defined by the ETSI GS NFV-SEC 024, which is still an early draft and has yet to be a Harmonized Standard. MTD could be effectively integrated into this architecture, as depicted in Figure 4.4. The additional elements being defined by the NFV-SEC 024 draft are represented in blue in that figure.

In such integration, MOTDEC would be placed as an OSS/BSS security manager, having a higher-level view and control of the NFV infrastructure. The soft MTD actions modifying the virtual network in the cloud environment, or the data plane flow in the Network Hardware would be performed at the NFVI layer, with the help of its SDN controllers and the VIM. Hard MTD actions, such as moving single VNFs or whole network services from one NFV infrastructure to another, are performed at the EMS and VNF layers. These MTD "playgrounds" are highlighted in the figure with pink clouds. A network service provider can improve MTD effects by using VIMs that control a diverse set of NFVIs, combining both *shuffle* and *diversity* schemes at the cloud level. The cognitive decision-making component, such as the OptSFC solution, is placed as an NFV security manager which uses, among other data, the monitoring data collected by MOTDEC to make its informative and optimal decisions (*e.g.* using ML/AI) on which MTD action to take. Finally, the decisions of OptSFC are communicated to MOTDEC, which enforces them on the 5G/B5G network by coordinating its operations with the NFV MANO, the network slice manager, and monitoring probes used to verify the network's state throughout the execution of the MTD action.

4.2.3 MOTDEC's Monitoring Module

As depicted in Figure 4.5, the monitoring module of MOTDEC mainly collects data regarding the infrastructure's configuration from the network slice manager and the NFV MANO component. The module periodically retrieves the catalog of network resources, core and edge domains, and updates the entities in a relational database management system (RDBMS). The NFV MANO and the network slice manager also deliver other information about the life cycle of the network resources, like

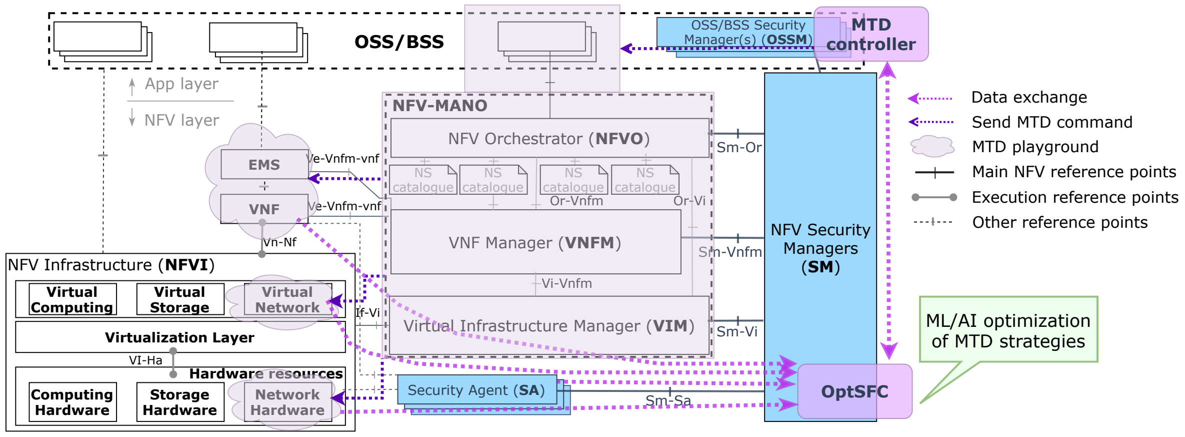


Figure 4.4: Monitoring process for the *MERLINS* MDP modeling.

the virtualization deployment units (VDUs) (*i.e.*, the individual VMs or containers that are running the NFs), their minimum resource requirements, their location and hosting VIM, their layout and links to VNFs, their belonging to different NSs, and their split among network slices. MOTDEC continuously collects such information periodically and permanently in another database other than the RDBMS, specifically, a file-based database using JSON-formatted files. The two databases are complementary: a relational database for querying currently available network resources, and a JSON-based archive that logs all historical changes, thus decoupling real-time operation from historical record-keeping.

The monitoring module also receives additional statistical data from network probes installed in the infrastructure to track the influx of packets in and out of the various network interfaces. This information is used to keep the MOTDEC framework capable of performing the various MTD actions, while it can also deliver its data to external security agents such as detection systems and the other *MERLINS* solutions like OptSFC, which requires such information for decision-making and MTD strategy optimization. In the case of anomaly detection systems, MOTDEC's monitoring could also receive alerts back and integrate them into its observation of the 5G network state.

4.2.4 MOTDEC's MTD Controller

The MTD Controller module enforces MTD actions grouped into two distinct categories: *soft MTD actions*, and *hard MTD actions*. The first category is named "soft" as its operations are software-defined network configurations that necessitate minimal resources and are very fast to execute. Conversely, "hard" actions are characterized by higher resource consumption, extended completion time, and

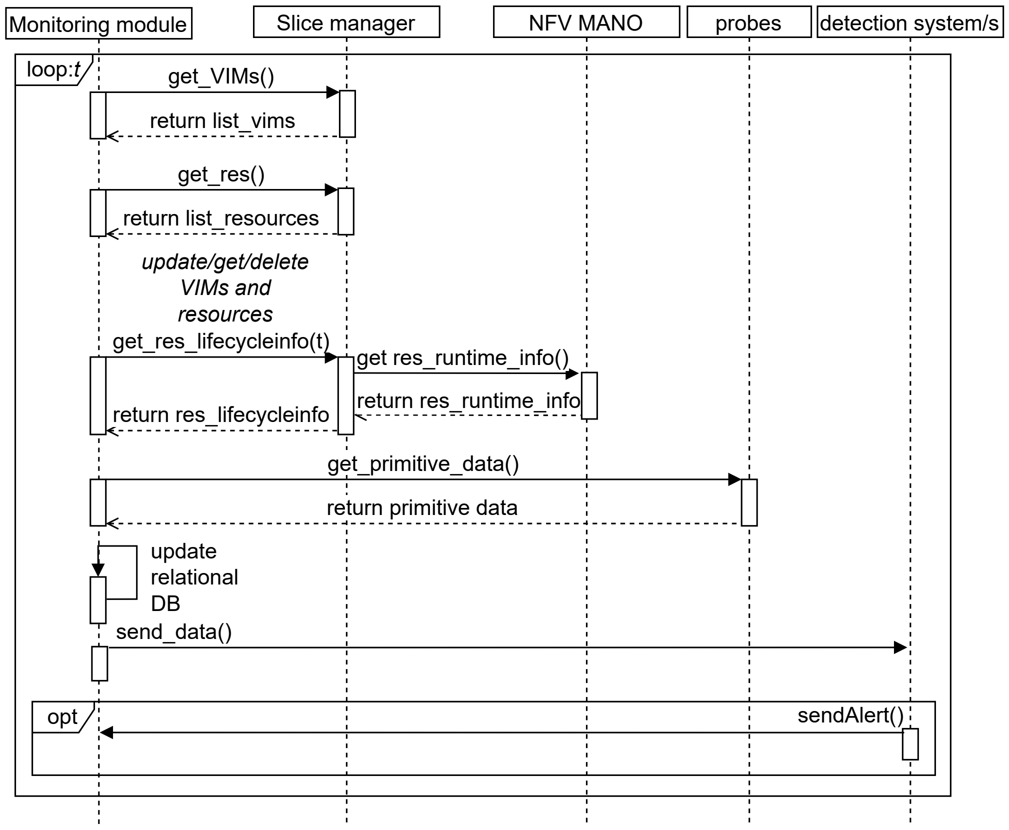


Figure 4.5: Monitoring process for the *MERLINS* MDP modeling.

ultimately a more significant overhead on service availability. The two categories are further defined as follows:

- **Soft MTD actions**: These are SDN-based shuffle operations performed on network interfaces, traffic flow, and network topology on both the internal and the external/public views of the network. In the internal view, the MTD controller could prevent an attacker inside the network slice from easily exploring and further penetrating it. In the external/public view, the resource is meant to be always accessible by external devices with a public interface. The MTD controller provides a different public IP address to suspicious end-users or UEs, allowing further targeted traffic analysis and adding a second layer of security through proxy VNFs. To this scope, the MTD controller integrates an SDN controller (*i.e.,* ONOS in NFV MANO case) to shuffle IP addresses, as detailed later (*c.f.,* Section 4.2.5), in the IPv6-based soft MTD action implementation. Another solution proposed in this thesis, TopoFuzzer, is also used to enhance soft MTD actions by changing node links and network data flows, further increasing the difficulty for attackers to identify the network's topology.

- **Hard MTD actions**: These are operations directly performed on the NFV assets used in the network infrastructure, for both assets allocated by the operator's clients and assets deployed by the operator to provide and manage their services. There are two hard MTD actions implemented in MOTDEC: 1) *MTD restart action:* the MTD controller restarts the NSs or NFs by re-instantiating the resource starting from verified images. This mitigates security scenarios of attackers introducing themselves in the virtual units to eavesdrop and acquire sensitive data, block the application running on the unit (resulting in a DoS attack), encrypt the unit with ransomware, or create a C&C bot and exploit it as a vector for other chained attacks. The new instance of the service replaces the old one, expelling the intruder from the logic (and physical) resource. 2) *MTD cloud diversity action:* the MTD controller moves the protected resource from one VIM to another with a different cloud execution environment. This action changes the environment of the running resource and reduces the threats due to different peculiar system's attack surface. In practice, this action is similar to the MTD restart action, but performs a LiMi of the VNF by creating a new resource instance in a different VIM than the previous one. Thus, in addition to the MTD diversity effect, it also mitigates the same threats the MTD restart action addresses.

4.2.5 MOTDEC Implementation

This section further delves into the specific implementation of MOTDEC and the MTD controller's soft and hard MTD actions, specifically, the IPv6-based IP shuffling mechanisms and the implementation of stateless LiMi and instantiation. It is noteworthy that other implemented MTD actions, such as those in ContMTD and TopoFuzzer solutions within the context of this thesis, also contribute to expanding the portfolio of MTD actions that the *MERLINS* framework can execute, encompassing both *soft* and *hard MTD action* categories. Moreover, other MTD actions can be implemented and integrated in the future to extend *MERLINS* capabilities.

Soft MTD action: SDN-based IPv6 shuffling

MOTDEC's soft MTD action implements the shuffling of ports and IP addresses of VNFs in the IPv6 address range, proactively mitigating network reconnaissance and fingerprinting attacks [139]. This implementation divides IP shuffling and port shuffling into two independent components: *IP Shuffling* masks the real IPv6 address of protected VNFs with a virtual IPv6 address that changes in a set interval (*i.e.*, all communication addressing the actual address is dropped to force the use of the

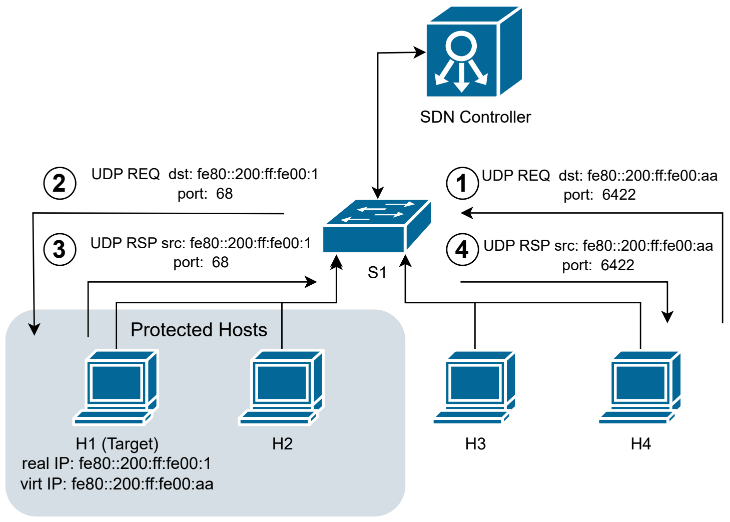

SDN Controller
2 UDP REQ dst: fe80::200:ff:fe00:1 port: 68
1 UDP REQ dst: fe80::200:ff:fe00:aa port: 6422
3 UDP RSP src: fe80::200:ff:fe00:1 port: 68
4 UDP RSP src: fe80::200:ff:fe00:aa port: 6422
S1
Protected Hosts
H1 (Target)
real IP: fe80::200:ff:fe00:1
virt IP: fe80::200:ff:fe00:aa
H2
H3
H4

Figure 4.6: MOTDEC's soft MTD action workflow.

virtual address); *Port Shuffling* switches the protocol header of packets in a fixed interval to mask the protocol used by an application.

Figure 4.6 shows a typical UDP packet flow with Host 4 (H4) addressing the protected Host 1 (H1):

1. H4 sends a UDP request with the destination IP as the virtual IP of H1 and with the destination port of the virtual port of H1.
2. The destination address and port are switched from the virtual IP and port of H1 to the real IP and port of H1 by the switch S1.
3. H1 sends a UDP response with its real IP and port as the source IP and port to H4.
4. The source IP and port are switched from the real IP and port of H1 to the virtual IP and port of H1 by the switch S1.

Algorithm 1 shows how the MTD controller handles unknown packets. MOTDEC's soft MTD action is implemented in Python and uses the Python-based Ryu as the SDN controller.

Hard MTD action: stateless VNF migration and reinstantiation

MOTDEC implements hard MTD actions for stateless VNFs, in other words, for those services that do not require a checkpoint and transfer of the last occurring state when migrated. This approach differs from stateful VNFs, which are explored in the context of container-based services in the ContMTD solution further proposed in this thesis. The LiMi and live reinstantiation of stateless VNF services implemented by MOTDEC are conducted in four steps:

Algorithm 1 Packet Handling for Shuffling

```
Require: $packet.type = TCP|UDP|ICMP$
 1: $hdr = pkt.hdr$
 2: if $hdr.dst.ip$ in $getRealIPs()$ or $hdr.dst.port$ in $getRealPorts()$ then
 3:     drop packet
 4: end if
 5: if $hdr.dst.ip$ in $getVirtIPs()$ then
 6:     $hdr.dst.ip = getRealIP(hdr.dst.ip)$
 7: end if
 8: if $hdr.src.ip$ in $getRealIPs()$ then
 9:     $hdr.src.ip = getVirtIP(hdr.src.ip)$
10: end if
11: if $hdr.dst.port$ in $getVirtPorts()$ then
12:     $hdr.dst.port = getRealPort(hdr.dst.port)$
13: end if
14: if $hdr.src.port$ in $getRealPorts()$ then
15:     $hdr.src.port = getVirtPort(hdr.src.port)$
16: end if
17: $addFlow(pkt)$
```

1. A new instance of the VNF is re-initiated with an authenticated image in the chosen location (either in the same location or in a different one if it is a migration.)

2. The MTD Controller waits for the availability of the new instance. While this transition is still undergoing, the service is available through the old instance. In order to keep both instances in the same network, a new IP is assigned to the new instance.

3. When the new instance is ready, the traffic should be redirected from the old instance to the new one in a transparent way, i.e., connected clients should not notice that they moved to a new instance.

4. Once all traffic is redirected, the old VNF instance is terminated and the related resources are freed.

The advantages of hard MTD actions for stateless VNFs rely mainly on the usage of an authenticated image in Step 1, removing from the service any post-instantiation infection and APTs originating from malware (*e.g.*, installed backdoors, spyware, botnet command and control (C&C), ransomware, or other hijack services). Moreover, a service can increase the fault tolerance to its host by migrating the VNF to a different edge node in case the current node is under attack or simply under maintenance. Another worth-noting aspect of hard MTD actions for stateless VNFs is that old instances stay employed until the new ones are operational. This allows MTD restart actions to have as little QoS/QoE overhead over the communication performances as the soft MTD actions, since the only moment where communications get interrupted is the reconfiguration of the network links and traffic routing.

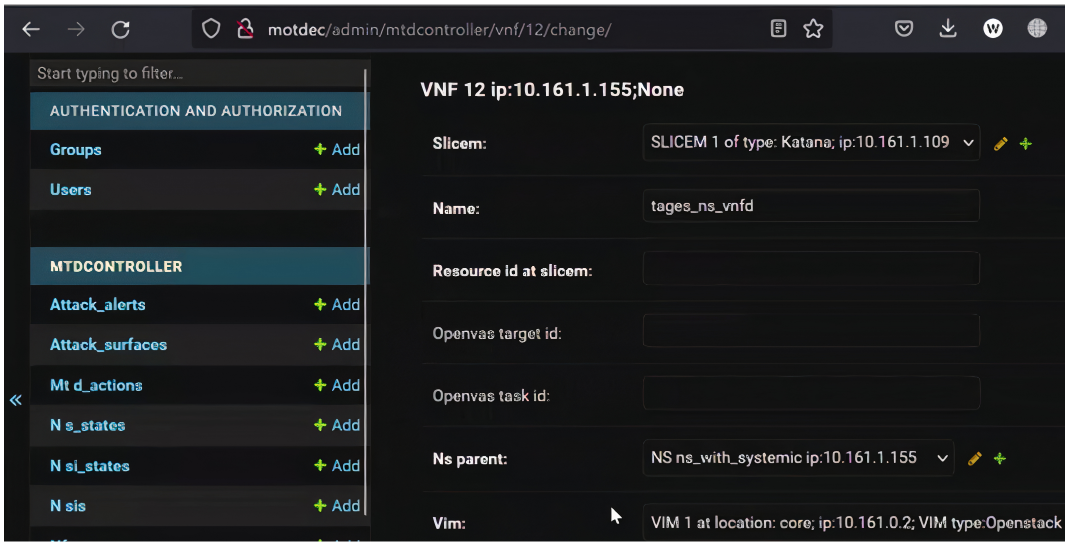
For the MTD cloud diversity action, the same argument is valid. However, there is an additional QoS/QoE overhead if the new cloud infrastructure is more distant or has worse network performance. For inverse reasons, MTD cloud diversity action could instead improve the communication performance of the protected service by decreasing the physical distance of the service to the UEs, or if the new destination VIM has better network performance. This is a motivation to explore efficient cognitive systems that would strike a trade-off between performance optimization and MTD security efficacy. Finally, there are other concerns with hard MTD actions for stateless VNFs in terms of resource consumption and operational costs. They may take a considerable time to complete the instantiation of the new instance. This may take several minutes depending on: i) infrastructure capabilities, ii) whether the NF is a VM or a container, and iii) the size of the VNF/CNF image. During this time, the resources allocated for the VNF are doubled, generating a financial operational cost for hard MTD actions.

MOTDEC is implemented in Python using the Django framework. The monitoring module is implemented with interfaces to Katana and OSM, which are, respectively, the network slice manager and the NFV MANO used in the 5G testbed implemented in this thesis (Section 5.1). The monitoring data collected on VNFs is accessible both from a REST API interface, implemented with the Django REST API framework, and a web graphic user interface (GUI) as illustrated in the screenshot in Figure 4.7.

Finally, the Katana slice manager [140] is also directly interfaced with the underneath OpenStack NFVI, enabling the implementation of hard MTD actions (*i.e.*, stateless LiMi and reinstantiation) at the NFVI level and its enforcement from the slice manager's interface.

4.3 ContMTD - Live Migration of Microservices-based NFs

The ContMTD solution extends *MERLINS* portfolio of hard MTD actions, addresses the LiMi of containerized stateful services, further answering ***RQ1*** and partly ***RQ2*** *'How can we minimize the network and resource overhead associated with MTD operations to ensure system performance and scalability?'* as it partially handles the efficiency issue of stateful service live migration (LiMi) with the usage of



VNF 12 ip:10.161.1.155;None

Slicem:	SLICEM 1 of type: Katana; ip:10.161.1.109
Name:	tages_ns_vnfd
Resource id at slicem:	
Openvas target id:	
Openvas task id:	
Ns parent:	NS ns_with_systemic ip:10.161.1.155
Vim:	VIM 1 at location: core; ip:10.161.0.2; VIM type:Openstack

Figure 4.7: MOTDEC Web GUI.

containers over VMs. Existing research and solutions address the challenge of migrating multiple containers or VMs composing a single application simultaneously. This is especially relevant in today's microservices-based software architectures, as future B5G and 6G networks also move towards the usage of CNFs and cloud-native orchestration technologies. In fact, the work in this thesis takes into account this evolution towards Telco Clouds, since MOTDEC's stateless VNF migration and reinstantiation can be easily applied to CNFs as well, *e.g.*, via pod replica management in Kubernetes. On the other hand, ContMTD focuses on CNF LiMi, as VNF LiMi alone has comparatively more limitations for stateful services, some of which are:

- *Image size*: CNFs are lightweight, thus, they have faster checkpoint and migration phases with a lower network bandwidth and latency footprint compared to VNFs.

- *VM-centric migration*: Some approaches often assume containers of the same service reside on a single VM, enabling the LiMi of the entire host VM [141]. But this is not applicable when a service is scattered across different clusters or VM cluster nodes.

- *Heterogeneous microservices load*: Other studies do explore parallel individual container migration, but they employ the same LiMi method to all containers [56]. This approach fails to consider the diverse nature and resource demands of each container. In fact, those can be memory-intensive, CPU-intensive, network-intensive (*i.e.*, high I/O), or disk-intensive (*i.e.*, high R/W).

4.3.1 ContMTD's Architecture

As there is a lack of a one-size-fits-all optimal LiMi method for all types of load, a dynamic approach that tailors the migration algorithm to the specific container type is necessary. For instance, microservices commonly utilize standalone containers for caches and dictionaries in in-memory data structures (*e.g.*, Redis and Memcached). Such a container with a high dirty memory page rate can benefit from a simpler cold migration approach without pre-copy or post-copy optimization methods. Another container could have a relatively small state, for which a post-copy migration strategy might incur lower QoS overhead compared to pre-copy alternatives.

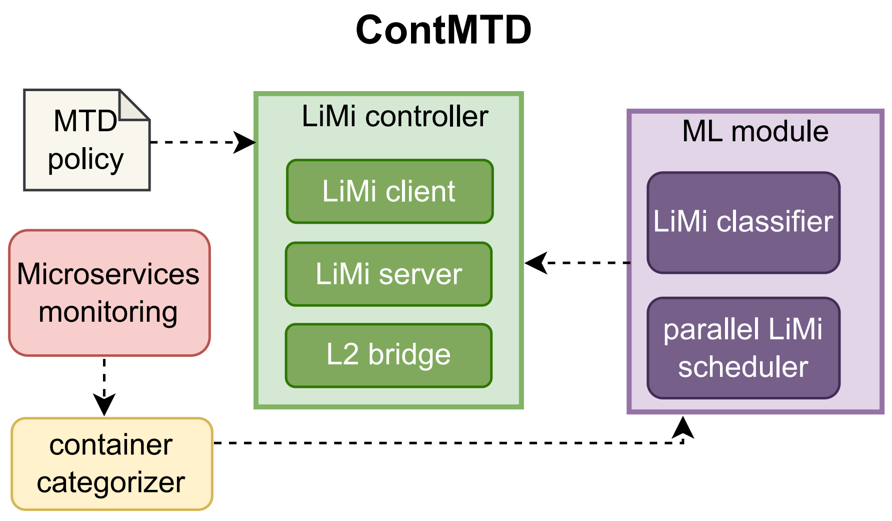


Figure 4.8: ContMTD architecture.

The ContMTD framework evolves around the nodes of a cloud infrastructure, whether such nodes are edge nodes or core datacenter nodes. The modules depicted in Figure 4.8 compose the framework's architecture and consist of the following: the microservices monitoring module, the container load categorizer, the LiMi controller, the ML module, and the MTD policy. The first two modules (colored in red and yellow in the figure) form the initial data-gathering on the running microservices and the classification of containers based on their resource consumption. The ML module (in violet) deals with two tasks: *1)* deciding the optimal LiMi migration method for individual containers, performed by the LiMi classifier, and *2)* estimating the migration time of the various containers, necessary to improve the scheduling of multi-container microservices migration. The MTD policy is the decision engine selecting which microservices application or which individual container to migrate, while the LiMi controller enforces the migration, using the LiMi clients and servers running in the cloud nodes (in inter-cluster and multi-cluster migrations). The LiMi controller also uses a layer 2 network bridge mechanism to transfer traffic without disrupting existing sessions running on TCP. The remainder of this section further details the novel solutions proposed in ContMTD.

4.3.2 Parallel Migration of Microservices

As the literature reveals the lack of a one-size-fits-all optimal LiMi method for all types of container loads, ContMTD employs a dynamic approach that tailors the migration algorithm to the specific container type during parallel container migration, enabling the efficient mixed use of optimization techniques to transfer a full set of app-related microservices.

ContMTD categorizes each container based on four metrics: CPU load, memory load (*i.e.*, high memory dirty-page rate), storage load (*i.e.*, high disk r/w operations), and network load (*i.e.*, high packet throughput). For each metric, to simplify the categorization, the resource consumption is set to three intensity levels: low, medium, and high. These levels cannot be objectively determined as they depend on individual use case requirements, but are estimated based on typical resource demands observed, such as cloud instance offerings from major providers like AWS [142], and considering a median container-based organization hosts approximately eight containers per host, [143]. Thus, the following thresholds were used: 40%, 65%, and 100% for CPU load; 150MB, 240MB, and 1.25GB for memory load; 10MB/s and 200 IOPS, 50MB/s and 500 IOPS, and 150MB/s and 1500 IOPS for storage load; and 100 req/s, 1000 req/s, and 5000 req/s for network load. For the network load metric, an additional intensity level is added, which is the "very low" and aimed at including private computational container applications or idle applications with very low to absent network traffic. Given this categorization, the categorization space of a container is $3(CPU) \cdot 3(RAM) \cdot 3(disk) \cdot 4(network) = 108$. However, some combinations are impossible to have in practice, reducing the possible combinations to 34 (detailed in Section 4.3.5). This categorization is then used to experiment with the performance of all LiMi methods on all the possible categories of containers, which is used to form the dataset used to train an ML classifier. ContMTD then leverages the LiMi classifier (of the ML module) to employ the predicted best LiMi method for the container.

The dataset is also used to learn a regression model to predict the total migration time of a specific container category. This prediction is used to align the parallel LiMi of containers so that all containers related to the same microservices application are restored at the destination node simultaneously. This is to prevent application-level errors that would occur if one restored container attempts to communicate with a container that is still being migrated. The details on the models trained from the dataset and the LiMi optimization workflow of ContMTD are presented in Section 4.3.3.

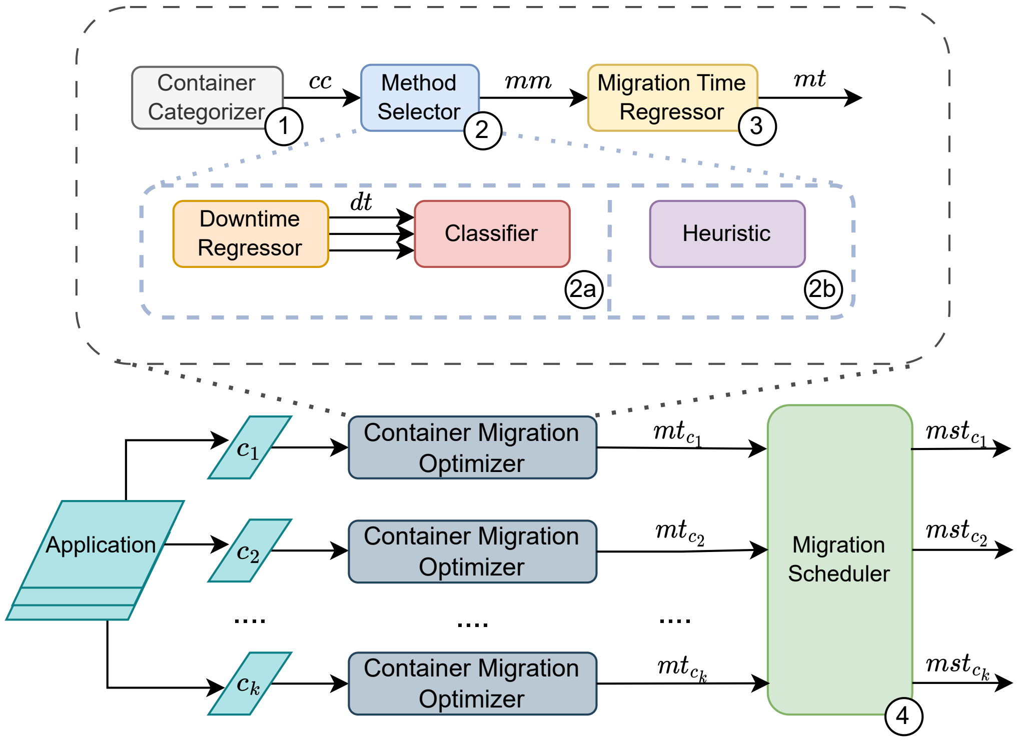


Figure 4.9: The migration optimization workflow of ContMTD.

4.3.3 ML Module and Optimization Workflow

The ML module is designed to take as input the category of the containers to be migrated, and send to the controller the schedule that defines, for each container, the migration method and the time to which the migration should occur. To this end, ContMTD follows a workflow of four steps, visualized in Figure 4.9 and defined as follows: (1) The containers to be migrated are monitored and categorized based on the resource metrics, fed to the method selector logic; (2) The migration method selector subsystem can be implemented in two different ways: (2a) An ML regressor model is trained to estimate the downtime of the migration given the container's properties for each of the four LiMi methods, and feed them to a min-max classifier, which selects the method with the lowest downtime; (2b) A classifier directly determines the best method given the category of the container, such as the heuristic model proposed in ContMTD. This approach does not require a regression model and provides faster decisions, as explained in Section 4.3.5; After the method is decided, (3) A second regression model predicts the total migration time of the given container and the migration method; Finally, (4) A scheduling algorithm calculates the time at which each container should be migrated in order for them to be restored simultaneously, thus providing the order of the migrations.

4.3.4 Mitigating Malware Migration in Stateful LiMi

To restore the important security properties of stateless LiMi when migrating stateful services, ContMTD uses the microservices architecture and best practices in cloud-native software development. One such practice is decomposing services into multiple microservices, maximizing the number of stateless components while minimizing the stateful ones, which can be deployed in more isolated environments, such as bare-metal or VM-based servers. The term "stateless-ification" is introduced to describe this process.

Figure 4.11 illustrates a use-case scenario of a generic web service following this microservices process. The service includes a front-end GUI, a REST API gateway, an authentication system, a user manager, and a messaging service, all provided as stateless containers. Meanwhile, the databases, in-memory caches, and other service functionalities are provided as stateful applications. The red containers are particularly exposed to potential security threats, given their reception of requests from external users/devices. Orange containers signify a diminished risk profile, whereas those in green exhibit the minimal risk to security threats given the required attack chain to reach them.

ContMTD proposes an MTD security policy that restricts access to stateful microservices through stateless ones. This configuration ensures that to compromise a stateful container, attackers must first breach the stateless container. This setup enables more frequent and seamless stateless LiMis for containers at the service's forefront, which are less critical but more vulnerable to attacks and undetected infections, akin to a buffer security zone. By frequently migrating these potentially exposed stateless containers, the system effectively "cleans" the service's attack surface from malware and APTs, significantly reducing the likelihood of attackers penetrating deeper into the more critical stateful containers. This strategic design lowers the need for frequent stateful LiMis, which are not primarily used for infection cleansing but are employed as a proactive MTD measure to mitigate the risk of side-channel attacks, as previously addressed in SotA LiMi-based MTD approaches discussed in the related work section. The effectiveness of this policy is demonstrated in the performance evaluation in Section 5.3.1.

4.3.5 ContMTD Implementation

This section describes the implementation of the ContMTD framework and further technical details of its modules, focusing on the container LiMi, the creation of a LiMi dataset, and the ML regression and classification models trained.

Container LiMi

The LiMi controller module manages container migrations and comprises three components: a LiMi client, a LiMi server, and an L2 bridge. During migration, both the LiMi server and client establish a secure SSH tunnel to ensure end-to-end encryption of the container's checkpoint during transfer. The transfer process utilizes *rsync*, which is optimized to transmit only the incremental differences between the new checkpoint and other possibly existing images of the container at the destination. The LiMi controller supports various migration methods (basic, pre-copy, post-copy, and hybrid) via the LiMi client, which directly interfaces with runC and its CRIU integration to initiate migrations. The LiMi server also operates through the runC interface to receive the eventual pre-dumps, dumps, and post-copy memory pages from the LiMi client and to restore the container at the destination node. For stateful migrations, CRIU is configured to preserve TCP-established sockets, ensuring that existing connections between the container and its clients remain intact. Since TCP sessions are defined by IP address and port pairs, the L2 bridge module ensures the migration of the container's IP address at the ARP (Address Resolution Protocol) table level. This is critical for maintaining session continuity and minimizing availability disruptions for services that rely on persistent connections, such as databases, SSH sessions, and voice-over-IP (VoIP) applications. Finally, the LiMi client reduces LiMi downtime by transferring in parallel both the dump/checkpoint delta and the volume delta for container applications with writable root file systems.

Benchmarking containers

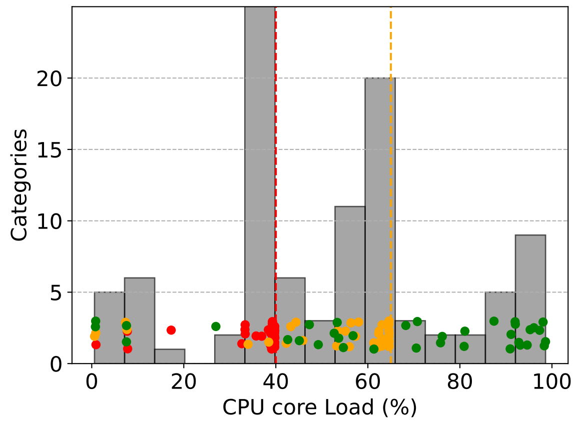
To test all the 108 categories that containers can fall into, and for subsequent benchmarking purposes, a containerized application is implemented to cover the different loads as follows:

1. CPU: perform a series of exponential calculations and use a random number generator. Multiple instances are executed for multi-core loads.

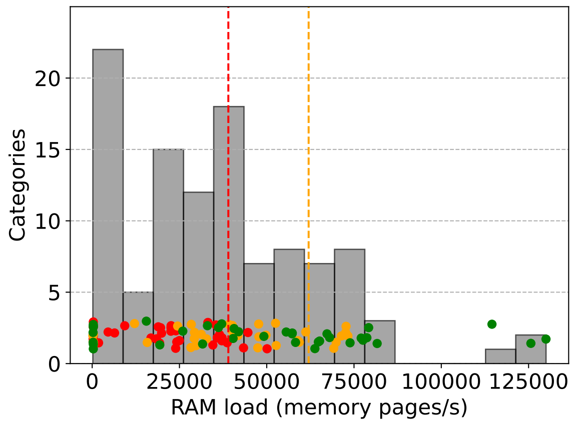
2. RAM: saves the randomly generated data in the application memory segment. To reduce RAM operations' dependency on CPU usage, only 1 MB of data is randomly generated at every writing cycle, which is then written iteratively to 50 MBs of newly allocated memory pages.

3. Disk: writes the data in a file and flushes the file descriptor to ensure it has been written in the disk.

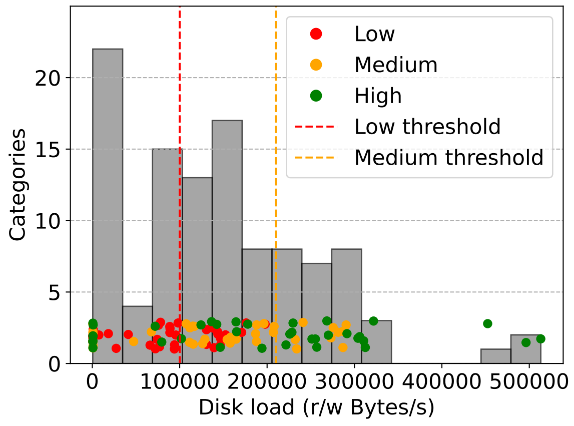
4. Network: runs a FastAPI server that receives HTTP requests and replies with the total number of requests received, allowing the verification of a successful stateful migration.



(a) CPU load distribution.



(b) RAM load distribution.



(c) Disk load distribution.

Figure 4.10: Measurements of actual resource consumption vs. intensity levels[144].

The different intensity levels are set within a container by limiting the resources the container can use when running. This is done using the Linux `cgroups` kernel functions set at the container runtime level (`i.e.`, runC), except for the network intensity limits that are set by configuring the Locust client with a specific number of users and HTTP requests per second sent to the container, following the thresholds previously defined in Section 4.3.2.

When containers are operational, the monitoring module of ContMTD uses the runC interface to get the status of each container and the Linux `pidstat` kernel function to monitor the CPU, RAM, and disk in real-time. During the evaluation, it became clear that some combinations are practically not feasible. For instance, when the CPU limit is low, the RAM and disk usage cannot be high due to the computational bottleneck of the CPU, effectively reducing the performance of read/write operations per second and IOPS, even when limits are set to high. Similarly, if the limits on RAM are low, disk performance will also be reduced. Hence, a series of measurements is run for all the 108 possible combinations, running the container application for 180 seconds (*i.e.*, three minutes) and measuring the actual resource consumption of the application. The measured resource consumption is evaluated to find which combinations are actually possible to run, and the thresholds reflecting the distinction of intensity levels on the 4 resources are statistically set. Figure 4.10 shows the distribution of resource consumption for each type of resource, using `pidstat` for CPU, RAM, and disk measurements. New intensity thresholds are then fixed based on the distribution to maximize the number of technically valid combinations. In a second step, some combination intensities were relabeled based on these thresholds rather than the original *cgroups* limit, defined as the "de facto consumption" of the containers that were expected to use resources differently. For instance, the container "llmo," standing for low CPU, low RAM, medium disk, and "o" networking, was renamed "lllo" as de facto; the disk consumption was low. Using the new "de facto" consumption labels, 34 valid combinations are obtained, which inherently include the 21 combinations validated in the first step, as their new labels are the same as their original ones and they get equally revalidated.

Dataset creation

The LiMi measurements used to form the dataset for the ML module are performed on the running container application with the 34 resource combinations. Each container is migrated in 4 different ways: cold/standard LiMi, pre-copy migration, post-copy migration, and hybrid migration. Each migration is performed 55 times per container, for a total of 7480 LiMis performed. Out of these 7480 migrations, 5890 could be validated due to technical issues.

On each LiMi performed, along with the LiMi optimization method used and the container category, additional features are extracted, including total migration time, pre-dump time and pre-dump size, pre-dump transfer time, dump time and dump size, dump transfer time, volume size and volume transfer time (the size considers delta optimization of sync transfer), time for parallel transfer of the dump and volume deltas, layer 2 IP transfer (via ARP table update), restore time, approximate container downtime (calculated by summing dump time, dump transfer time and restore times), and a precise downtime measured by the locust traffic generator (including the additional downtime from packets queuing).

ML module's regressors and classifiers

In this work, two classification techniques are proposed to predict the best LiMi method for a container category. The first is a statistical model built based on the migration downtime data in the dataset. It takes the average downtime per container category and per LiMi method, and selects for each category the method with the smallest average downtime. The second classification method uses a ML regressor model trained on the dataset to estimate the service downtime during migration. A simple classifier would then define the optimal LiMi method to select as the one with the lowest estimated downtime, as mentioned in the ContMTD workflow in Section 4.3.3. For the evaluation of the regressor model's performance, we test four different ML algorithms to train the model: **Random Forest Regressor** (RFR) [145], **Support Vector Regression** (SVR) [146], **NN** [147], and **Bayesian Ridge** (BR) [148]. The hyperparameters used are described in Table 4.2, selected with grid search and k-fold cross-validation during training. The same algorithms have been used to train the second regression model on the dataset for predicting the total migration time, used in the scheduling algorithm.

Table 4.2: Optimal hyperparameters for the regression models.

Model	Parameter Name	Optimal Value		Parameter Space
		Downtime	Mig. Time	
RFR	Max Depth	8	8	2, 3, 4, ..., 10
	Estimators	650	250	50, 100, ..., 1000
NN	Hid. Layers	160	200	100, 120, 140, ..., 200
	Act. Func.	ReLU	ReLU	ReLU, tanh, logistic
	Solver	LBFGS	LBFGS	Adam, SGD, LBFGS
BR	Max Iter.	700	800	100, 200, 300, ..., 1000
	Tol	0.0006	0.0004	1e-4, 2e-4, ..., 1e-3
SVR	Degree	6	7	1, 2, 3, ..., 10
	Kernel	poly	rbf	linear, poly, rbf, sigmoid

ContMTD scheduling algorithm

Algorithm 2 defines the ContMTD scheduling algorithm designed to optimize the start and end times of container migrations in such a way that all containers complete their migration process simultaneously. This minimizes the potential disruption to services caused by migration by coordinating the start times across all containers.

Algorithm 2 Container Migration Scheduling

```
1: procedure SCHEDULE_MIGRATION(cnt_list, reg_model)
2:     $T_{end} \leftarrow 0$
3:     for each cnt in cnt_list do
4:         $cnt.migration\_time \leftarrow reg\_model(cnt)$
5:         if $cnt.migration\_time > T_{end}$ then
6:             $T_{end} \leftarrow cnt.migration\_time$
7:         end if
8:     end for
9:     for each cnt in cnt_list do
10:        $cnt.start\_time \leftarrow T_{end} - cnt.migration\_time$
11:    end for
12: end procedure
```

CloudHopper, a recent work implementing parallel container LiMi [56], proposes a method that sorts containers based on their image size, transferring first the biggest container. The next biggest container is then transferred when its image size becomes equal to the remaining image size of the previous container still being transferred. An issue identified by this method is that it does not consider the possible difference in the pre-dump and dump time, as well as the restore time, which can drastically change depending on the container's load. ContMTD proposes a new approach that leverages the regression model of the ML module for the scheduling of the migration process. First, the migration time of each container in the application is estimated by using the trained regression model. Then, the maximum estimated migration time among the containers in the application is obtained, which will be used to schedule the migration of all relevant containers. For each container, the algorithm computes the ideal start time of migration by subtracting its predicted migration time from the overall maximum migration time. Depending on the accuracy of the regression model, this ensures that each container finishes migrating approximately at the same time.

4.3.6 ContMTD Security Strategy

A notable issue that has not been comprehensively addressed in the existing literature is related to the limitation in neutralizing malware infections and backdoors using stateful LiMi compared to the stateless LiMi method. In the case of stateless services, the image utilized for instantiating a new instance at the destination node does not have to be a direct checkpoint of the running instance at the source node; instead, it can and should be derived from an authenticated, original container image free of any infection or malware introduced post-instantiation. Conversely, in the context of stateful services, if an instance is contaminated and subsequently migrated, the new instance inherits the infection, perpetuating the compromised state of the service, i.e., stateful infection persistence. This distinction underscores a critical security difference and a challenge in maintaining the integrity and security of stateful services.

To utilize the security property of stateless LiMi when performing LiMi of stateful services, a solution is to leverage the microservices architecture and established decomposition best practices. One such best practice in decomposing the service into multiple microservices is to separate the stateless components of the service into a maximized set while maintaining a minimal set for stateful components, which may reside in separate microservices (or more traditional server VMs based on specific technical requirements), i.e., a "stateless-ification".

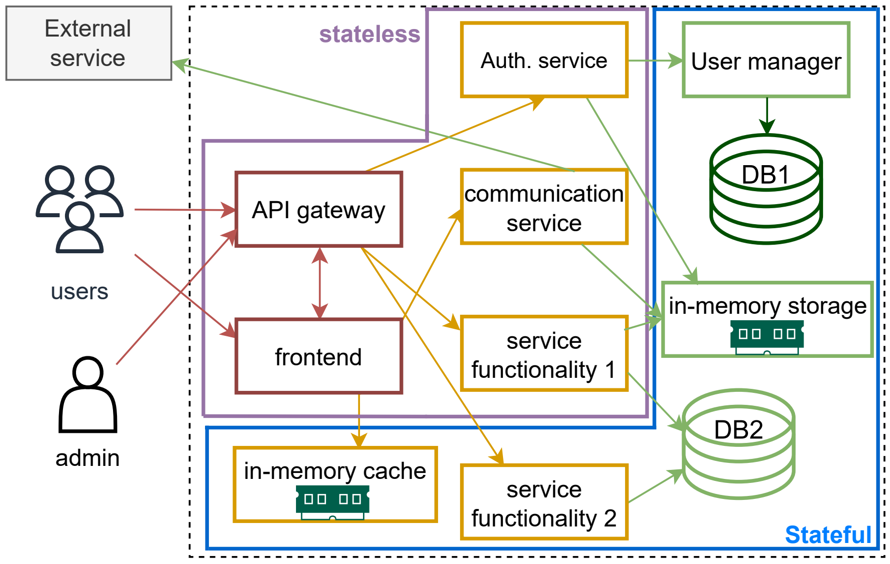


Figure 4.11: Microservices generic architecture of a web service with stateless containers on the forefront of the service. Red represents high exposure to security threats. Orange has a diminished risk profile, whereas green exhibits minimal exposure.

Illustrated in Figure 4.11 is an example application of this architecture for a generic web service comprising a front-end GUI, a user management and authentication system, a communication and message/queue service application, and other potential stateful applications requiring a database and/or an in-memory cache. The proposed security solution involves granting access to the stateful microservices through the stateless microservices, such that compromising a stateful container would require first breaching and compromising the stateless container. This configuration facilitates more frequent stateless LiMis for containers at the service forefront, which are less critical but most susceptible to attacks and undetected infections—akin to a demilitarized zone in a networked system.

This approach enables the frequent "*cleaning*" of the service from potential malware and backdoors, significantly reducing the likelihood of attackers advancing to the stateful critical containers sitting in the back. Furthermore, this setup would limit the necessity of frequent stateful LiMis of the latter containers, not used for infection cleansing but as a proactive MTD strategy to reduce the ASP of side-channel attacks.

4.4 TopoFuzzer - A Seamless Live Connection Handler

TopoFuzzer is the third solution proposed in this thesis and covers the implementation of the "network reconfiguration" operation of *MERLINS* EL layer, extending answers to the research questions ***RQ1*** and ***RQ2***. As covered in the foundational chapter in Section 2.3.3, operations such as LiMi require an efficient redirection of clients' requests and connections to keep the continuity of the provided service when it is being moved to a new location. A relevant challenge here is posed when dealing with session-based protocols such as TCP, especially in the event where the service changes its IP address, as in MOTDEC's IP and port shuffling performed with *soft MTD actions*. By maintaining long-lived sessions between the two endpoints, Topofuzzer effectively enables the live continuity of the migrated server, reducing the overhead of MTD operations on QoS and QoE of NFs. TopoFuzzer also enables the creation of other sets of *soft MTD actions* with a limited network overhead and resources required, using a fuzzing virtual network to change the apparent topology of the protected network.

4.4.1 TopoFuzzer Architecture

TopoFuzzer's architecture is depicted in Figure 4.12, encapsulating three main components: *1)* The IP mapping table mapping the public IP to the private IP, *2)* The redirection module composed of multiple proxy-NICs, and *3)* The fuzzing virtual network allowing to implement Soft MTD actions. The following subsections describe the main components of the architecture and their objectives.

IP mapping and interface to MOTDEC

Public IPs are predefined for each VNF as they remain fixed during hard MTD actions. When a VNF is deployed, the private IP assigned by the VIM hosting the VNF is given to the MTD controller MOTDEC, who then sends both IPs to TopoFuzzer. TopoFuzzer maintains the mapping between private and public IPs. If the VNF is deployed for the first time, a new mapping is registered and the public IP is assigned to a new vNIC, which will be used by an instance of the 2-socket proxy. If the

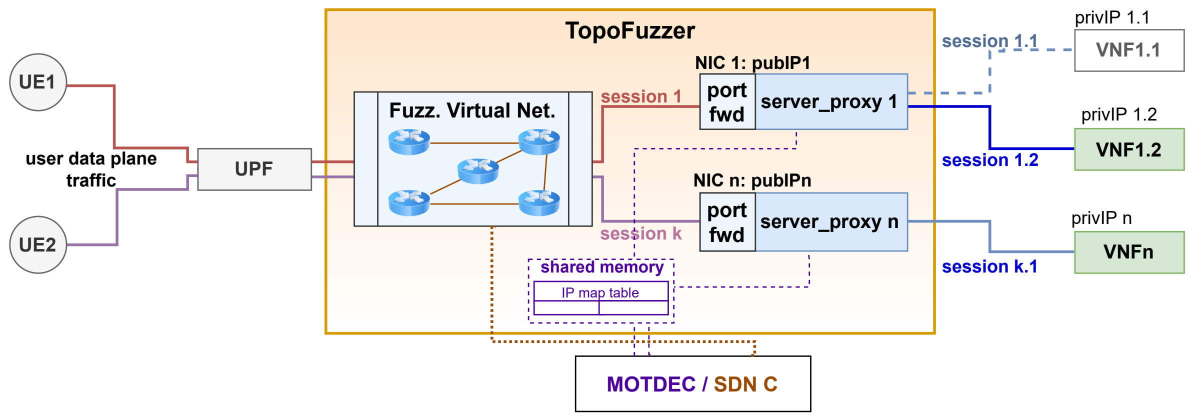


Figure 4.12: TopoFuzzer architecture.

VNF is already deployed, only the mapping is updated, while the vNIC and the proxy instance are preserved. The mapping is a hash table with two entries per one mapping (i.e., `pubIP`→`privIP` and `privIP`→`pubIP`). This allows fetching the IP addresses in both directions with a constant complexity (i.e., O(1)).

2-socket proxy redirection improvement using `conntrack` and port forwarding

TopoFuzzer's solution to maintain TCP sessions when the IP and port of an endpoint change is inspired by [108], which implemented a two-socket proxy system to seamlessly redirect an attacker's traffic to a honeypot server without the latter being aware of it. While such a use case only considers redirecting the traffic of a service running behind one specific port (*e.g.*, a web server on port 80), VNFs have multiple ports open. Hence, the proxy should be able to redirect traffic with different destination ports simultaneously. Moreover, a proxy node can only redirect the traffic to one VNF, as it configures its vNIC to use the VNF's public IP to receive client traffic.

In TopoFuzzer, to enable multiple connections with different destination ports to the same VNF, the proxy instance assigned to the VNF binds the `in_socket` to a fixed known port. A port forwarding rule is going to change the destination port of all the traffic sent to a VNF. This allows the proxy to receive all the traffic despite listening to one port. However, once the port changes, the different traffic streams are not distinguishable, as the proxy needs to forward the traffic to the private IP with the original destination port used by the client. In order to get back to the original destination port, TopoFuzzer uses a Linux Kernel feature that keeps track of all the connections and that is used to enable NAT operations, `conntrack` [149]. To identify the right connection of forwarded packets, TopoFuzzer uses three available values: the connection type (which, in this case, is always TCP), the source IP of the client, and the source port of the client. Knowing the connection is session-based, the pair (`source_IP`, `source_port`) is unique per connection (contrary to UDP, where the same pair can send packets to different IPs and ports).

Fuzzing Virtual Network

To perform Soft MTD actions without affecting the real network topology but changing the topology view from the client perspective (e.g., if they scanned the network with tools like `traceroute` and `nmap`), a virtual network composed of switching and routing nodes is placed between the proxy nodes and the User Plane Function (UPF). Adding a gateway in the route, removing one, or replacing it with another gateway are all operations that affect only the visible session to the clients. In contrast, the sessions from the out_sockets of the proxy nodes to the VNFs are internal by nature and, hence, not modified. These operations can be enforced by using an SDN controller connected to the switches of the fuzzing virtual network. The SDN Controller is hosted in the MOTDEC module, as it orchestrates MTD actions.

4.4.2 TopoFuzzer Implementation

The TopoFuzzer framework is implemented in Python. The IP mapping is implemented using a Redis[1] hash table, which is then accessible via a REST API implemented in Flask, enabling communication with MOTDEC. As Redis is an in-memory store, fetching a new private IP for a specific connection is considerably faster than passing through storage.

The fuzzing virtual network is implemented in Mininet, enabling the dynamic generation of lightweight nodes equipped with vNICs. Each node serves a dual purpose: (1) as a standard network endpoint and (2) as a runtime-generated proxy for individual VNFs. To achieve this, the system spawns isolated Python interpreters (handling the 2-socket proxy function) per node while retaining all processes within the shared Linux kernel space of the TopoFuzzer framework. This design supports rapid deployment in large-scale networks with minimal incremental resource overhead per VNF, as kernel-space operation avoids the need for full virtualization. Furthermore, proxy nodes exchange data and directly access the IP mapping table without network-based data exchange, reducing communication latency and bandwidth consumption.

[1]https://redis.com

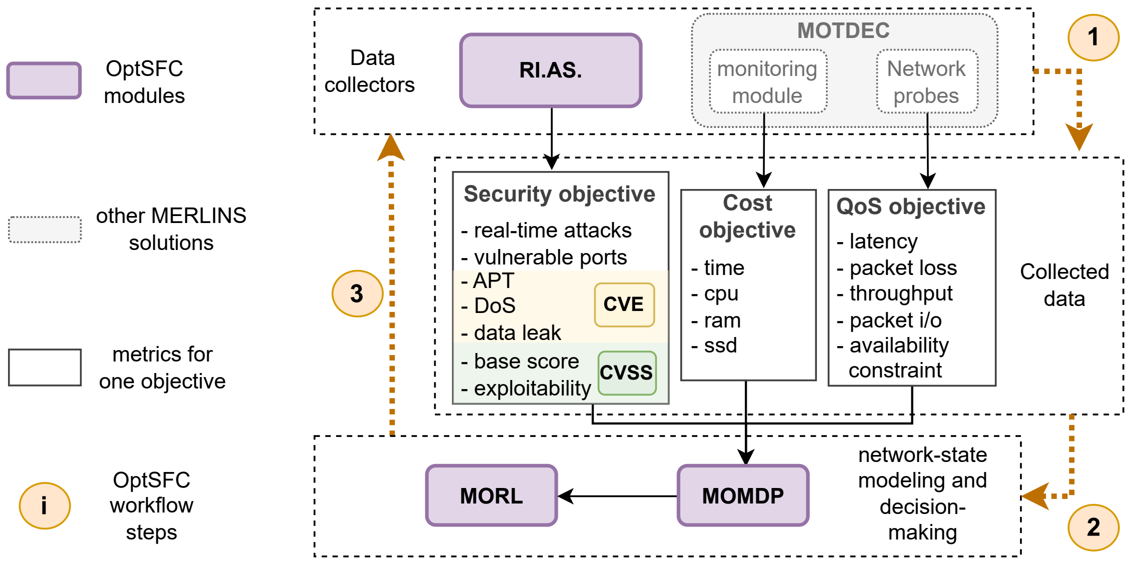


Figure 4.13: OptSFC architecture.

4.5 OptSFC - a Deep-RL Optimizer of MTD Strategies

The OptSFC solution implements the cognitive element of the *MERLINS* HLA, the DML, responsible for the tasks of decision-making and MTD strategy definition. OptSFC answers to the research questions ***RQ2*** and ***RQ3*** '*Which formal modeling approaches are most suitable for representing communication networks to enable near real-time monitoring and security assessments for MTD systems?*'.

4.5.1 OptSFC's Architecture

OptSFC generates optimal MTD policies for proactive security of network slices. By formally modeling telecommunication networks compliant with NFV into a multi-objective Markov Decision Process (MOMDP), OptSFC uses deep-RL to find the optimal balanced strategy to maximize security (*i.e.*, minimize threats), to minimize its operational cost, and to alleviate the impact on QoS and service availability.

OptSFC's architecture and the workflow of its decision-making process are depicted in Figure 4.13. *MERLINS* framework consists of the following modules: the risk assessment (RI.AS.) module, the MOMDP modeling module, and the deep-RL agent optimizing and deciding on the MTD policy. Finally, OptSFC is interfaced with the MOTDEC solution, using the data gathered by its monitoring module and sending the decisions of MTD actions to execute to its MTD controller, as depicted in the sequence diagram of Figure 4.14. The diagram details the workflow of OptSFC, operating

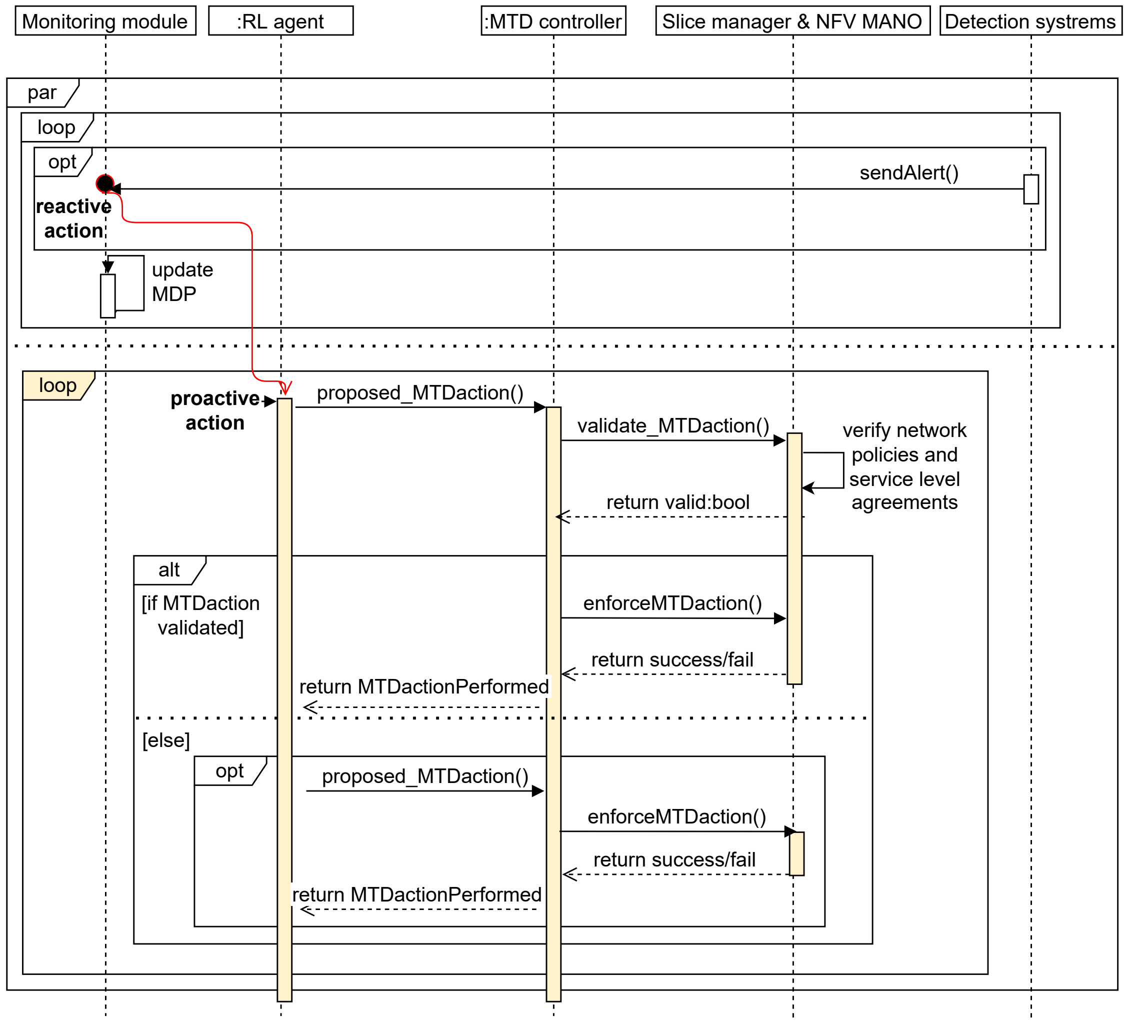


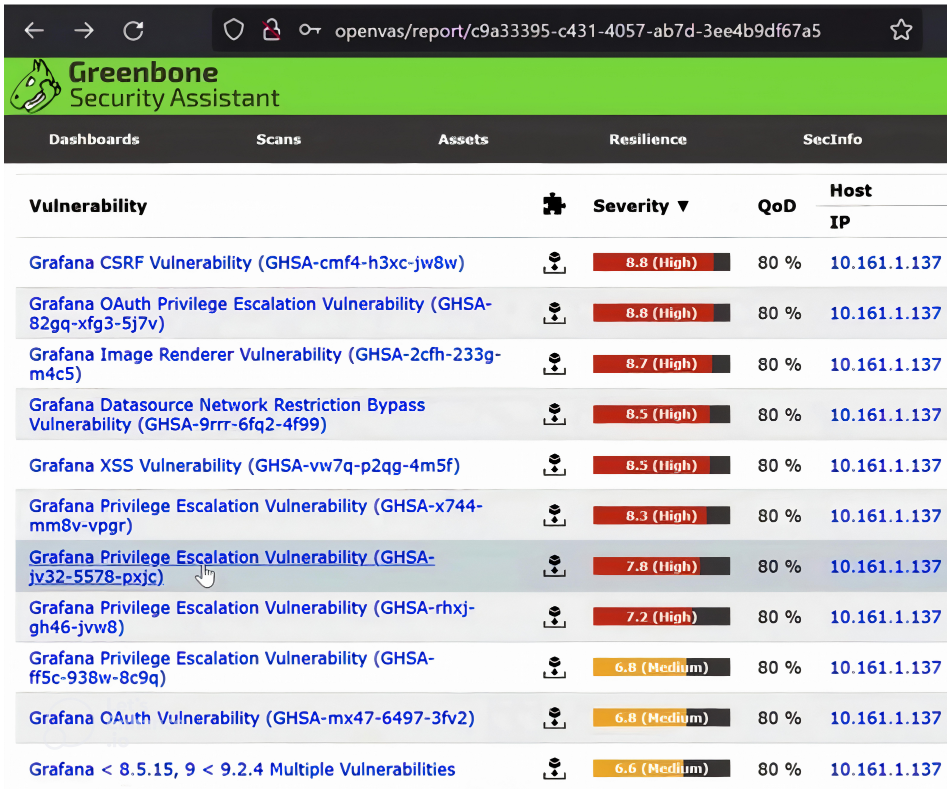
Figure 4.14: Decision-making process and MTD action enforcement in *MERLINS*.

the decision-making for MTD actions reactively, when a security agent sends an anomaly or attack alert, or proactively, when they are based on the MDP modeling of the state of the network to harden exploitability tasks of possible offenders. MOTDEC's MTD controller, upon receiving the decision from OptSFC, obtains a first-hand validation from the NFV MANO and network slice manager to verify the feasibility of the MTD actions against possible conflicts with other orchestrating operations or higher-level constraints.

The following sections describe the data collected by the RI.AS. module and other collectors in correlation to the three optimization objectives defined in the MOMDP.

Continuous threat analysis and risk assessment are performed to enhance the MOMDP for proactive MTD decisions. Using the open-source vulnerability scanner OpenVAS, the RI.AS. module identifies running services using the Common Platform Enumeration (CPE) and performs active and passive vulnerability scans using maintained public and private vulnerability databases such as the Common Vulnerability Enumerations (CVE) database and Network Vulnerability Tests (NVT) database.

The former allows finding possible vulnerabilities based on the CPE of the services running in the targeted host. These scans are passive and prompt but may contain false positives. NVTs instead are based on active and more precise scans using local security checks of a range of operating systems and solutions from different vendors (*e.g.*, Intel, Cisco, and Oracle products). The RI.AS. module schedules the scans for all VNFs in one or multiple network slices, periodically and every time a VNF is re-instantiated. Figure 4.15 illustrates the web GUI of OpenVAS and the example of CVEs detected when scanning a VNF in the 5G testbed with a vulnerable version of Grafana. The module



Vulnerability		Severity ▼	QoD	Host
				IP
Grafana CSRF Vulnerability (GHSA-cmf4-h3xc-jw8w)		8.8 (High)	80 %	10.161.1.137
Grafana OAuth Privilege Escalation Vulnerability (GHSA-82gq-xfg3-5j7v)		8.8 (High)	80 %	10.161.1.137
Grafana Image Renderer Vulnerability (GHSA-2cfh-233g-m4c5)		8.7 (High)	80 %	10.161.1.137
Grafana Datasource Network Restriction Bypass Vulnerability (GHSA-9rrr-6fq2-4f99)		8.5 (High)	80 %	10.161.1.137
Grafana XSS Vulnerability (GHSA-vw7q-p2qg-4m5f)		8.5 (High)	80 %	10.161.1.137
Grafana Privilege Escalation Vulnerability (GHSA-x744-mm8v-vpgr)		8.3 (High)	80 %	10.161.1.137
Grafana Privilege Escalation Vulnerability (GHSA-jv32-5578-pxjc)		7.8 (High)	80 %	10.161.1.137
Grafana Privilege Escalation Vulnerability (GHSA-rhxj-gh46-jvw8)		7.2 (High)	80 %	10.161.1.137
Grafana Privilege Escalation Vulnerability (GHSA-ff5c-938w-8c9q)		6.8 (Medium)	80 %	10.161.1.137
Grafana OAuth Vulnerability (GHSA-mx47-6497-3fv2)		6.8 (Medium)	80 %	10.161.1.137
Grafana < 8.5.15, 9 < 9.2.4 Multiple Vulnerabilities		6.6 (Medium)	80 %	10.161.1.137

Figure 4.15: Screenshot of the OpenVAS GUI and the CVEs found in the running VNFs.

then groups the CVE details of detected vulnerabilities into three general types of threats: 1) APTs, identified by vulnerabilities that allow the remote execution of code or the malware infection of the target; 2) Data Leak Threats, identifying vulnerabilities that allow gaining sensitive information: *e.g.,* SQL and XSS injection, directory traversals, and local file inclusion; 3) DoS Threats, grouping CVEs based on Buffer Overflow vulnerabilities and NVTs finding network-based DoS vulnerabilities.

The RI.AS module quantifies the risks of a VNF for each of the three major threat groups in terms of ASP. The ASP value is calculated by considering the number of ports open for vulnerable applications, by aggregating the relative maximum *exploitability_score* and *base_score* of the vulnerabilities, obtained from the the Common Vulnerability Score System (CVSS) [150] from the National Vulnerability Database (NVD)[151] maintained by NIST. The ASP values are then incremented with time, representing the advantage of attackers when they have a static target. When an MTD action is performed on the VNF, the ASP values of the mitigated threats are reset to their original value (i.e., removing the time advantage and only reconsidering the vulnerabilities of the VNF). Finally, similar to the QoS SLAs, OptSFC allows the definition of security SLAs (SSLAs), generalized to a value indicating the criticality of an attack to the VNF. The security risk is then defined for each VNF as $sec\_risk = max(ASP \times cvss\_score) \times vnf\_impact$.

MTD Action Cost Measurement

Table 4.3: Regression results on the CPU and RAM price from 66 percent of the cloud market in 2022 Q3.

	Dependent variable:
	Price ($/hour)
α_1 (CPU_core)	0.031^{***} (0.001)
α_2 (RAM_GB)	0.004^{***} (0.0002)
β (constant)	-0.082^{***} (0.018)
Observations	72
R^2	0.994
Adjusted R^2	0.994
Residual Std. Error	0.127 (df = 69)
F Statistic	$5{,}706.468^{***}$ (df = 2; 69)
Note:	$^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$

To measure the cost of MTD actions in terms of resource consumption, an empirical measurement of the cost of virtual resources is done to find the coefficients between CPU cost, RAM cost, and storage cost, based on the definition of $resource_{cost} = \beta + \alpha_1 \times cpu + \alpha_2 \times ram_{gb} + \alpha_3 \times storage_{gb}$ in USD per hour ($\$/hour$). For simplicity, the cloud providers' convention of $\$/hour$ as a measurement unit for virtual resources is used in this formulation. The prices of over 70 VM offers are collected in 2022 Q3, ranging from VM instances of 1 CPU and 0.5GB RAM to instances with 128 CPUs and 864GB RAM, from four major cloud providers: AWS, Azure, Google Cloud, and OVH (covering at least 66 percent of the worldwide market in Q3 2022, estimated by Synergy Research Group [152]). We did not distinguish between high-tier and low-tier hardware, and since the various prices have different coefficients, a closed-form polynomial system is not solvable. We use linear regression to find the approximate coefficients, which have a low P-value and a strong correlation with the different VM offers. The strong correlation of the variables in the linear regression, shown in Table 4.3, demonstrates the statistical relevance of the coefficients found per unit costs of CPU and RAM. From such coefficients, we derive the costs of 0.03147 \$/h per CPU-core, 0.004244 \$/h per GB of RAM, and a constant β of -0.082. As storage is a cloud service provided separately from the VM instance, its cost is measured separately from the average of 37 hot storage prices (*i.e.*, SSD-based fast, regularly accessed, and high data-volume storage, with no further distinction between high-tier SSD and normal SSD disks). Prices are given by the same four cloud providers, giving a final mean price of 0.000066 \$/h per GB of storage.

4.5.2 MOMDP Modeling of the Network

For the decision-making task, the MDP model represents the network as a tuple (S, A, P, R, γ), where:

- S is the set of all possible states. In practice, at each time t, a state S_t is defined by the status of the resources to be protected, e.g., the 5G core, the UPF, the virtual Evolved Packet Core (vEPC), the gNBs (the equivalent of eNodeBs in 4G-LTE networks), and the functional NSs. The status of a resource is defined by its runtime condition (running, idle, voluntarily stopped, or accidentally stopped), the resource consumption (CPU, RAM, and disk), and its network metrics (such as I/O frequency, bandwidth, and latency). It is also defined by whether an anomaly detection system found it as a target of attacks or anomalies. The total number of possible states in the environment is $\|S\| = N_{resources} \times (1 + N_{consumes} + N_{net\_perf} + N_{attacks})$, where $N_{resources}$ is the number of resources to be protected and on which MTD operations are performed. 1 formally denotes the constant set of features like the runtime condition and other network characteristics. $N_{consumes}$ and N_{net_perf} are respectively the number of resource consumption metrics and network metrics, which are checked against established requirements. Finally, $N_{attacks}$ is the number of attacks detectable by an anomaly detection system. Attack detection would endow *MERLINS* ability to act reactively and proactively (based on measured performance and other passive metrics).

- A is the set of actions a_i, $i < NA$, that the RL agent can take. In practice, in this MDP, the actions are the different MTD operations available for each resource to be protected. Thus, the action space $|A|$ is upper-bounded by $NA = N_{MTD} \times N_{resources} + 1$, where N_{MTD} is the number of MTD operations applicable to a network resource and 1 represents the action of "doing nothing".

- P is the transition probability matrix representing the probability that an action a_i changes a specific state s to the state s', *i.e.*, $\forall a \in A, Pass' = P[S_t + 1 = s' | S_t = s, A_t = a]$. This matrix is updated during training and represents the uncertainty of the RL agent in reaching its goal with a specific action: for instance, the RL agent can decide to perform the action a_i to mitigate an ongoing attack, knowing this action could not succeed with a probability of $1 - Pass'$.

- R is the reward function that defines the reward obtained at time t+1 when performing an action a_i from a state s at time t, i.e., $R_{as} = E[R_{t+1} | S_t = s, A_t = a]$. In practice, the reward/penalty will be given based on four factors:

  1. The status of the protected resource (e.g., a penalty is given when the resource stops accidentally after an agent's action);
  2. The distance of the measured metrics from the established minimum and maximum requirements (e.g., the bandwidth used by a resource is less than what is needed, which can be computed based on the frequency of I/O operations, and is inferior to the requirement);
  3. The mitigation of an attack, which is rewarded proportionally to the threat that the attack poses, is defined based on the CVSS [150], which is a widely-used industry metric. Vice versa, when an attack occurs, the MDP scores a penalty, *i.e.*, a negative reward;
  4. The cost of an action, as each MTD action has a different cost in terms of resource consumption and "time to enforce", translated in a negative reward, *i.e.*, a penalty when the action is performed. Another factor is the importance of each protected resource for the 5G infrastructure. For example, the vEPC is a core function upon which all network

slices depend. Thus, it is more critical than other network services. This can be introduced in the MDP through higher rewards or penalties compared to when the same action is performed on a different resource;

- γ is the discount factor used to define the importance of the immediate reward compared to future rewards. As *MERLINS* performs a single continuous task, the deep-RL model cannot be trained on a batch of episodes, but rather, on the continuity of the management task. Accordingly, γ is set close to 1 to have a more relevant consideration of future rewards during the task continuation, rather than a greedier decision-strategy focused on immediate rewards.

The reward vector $\overline{R}$ groups the collected data/features based on three optimization objectives:

Improvement of the risk assessment for proactive security

The risk assessment for proactive security uses the data from the RI.AS. module previously described in Section 4.5.1.

Reduction of the operational cost of MTD

As some MTD actions, such as hard MTD actions, require additional resources to reinstantiate or migrate a VNF, the operational cost of MTD actions is defined as $mtd_{cost} = resources_{cost} \times deployment_{time}$. The results in Table 4.3 are used to measure $resources_{cost}$, while $deployment_{time}$ is the measured mean value for every VNF running in the network.

Improvement of the network performance (and reduction of MTD network overhead)

As MTD actions may require redirecting the traffic of end users, which can affect mid-term performance of the service based on the distance of the new location, we need to consider the network performance in OptSFC decision policy. To measure it, OptSFC collects from the MMT probe the following network metrics with a regular frequency of five seconds for every protected VNF: number of UEs connected to a VNF, connection latency (derived from the RTT values of packets), connection throughput, packet loss rate (derived from packet retransmission requests), and the number of packets flowing in and out. From these monitored values, we derive the mean packet loss rate increase and the mean latency increase caused by the MTD action, and define its QoS overhead as $mtd\_QoS_{overhead} = (1 + p\_loss\_rate\_increase) \times latency_{increase}$. Moreover, if an MTD action causes a violation of an SLA, the overhead is increased by a penalty factor, which is a hyperparameter of the ML-training phase.

4.5.3 Deep-RL and Multi-Objective Optimization of MTD Strategies

OptSFC optimizes the MTD proactive decision policy using deep-RL. The deep-RL agent can propose an MTD action to the MTD controller periodically or after an alert from a detection system (reactive event-driven case). In both cases, the proposed MTD action has to be validated by the network slice manager and the NFV MANO for possible conflicts with other management and orchestration actions.

As the *MERLINS* architecture is agnostic of the decision-making engine, different deep-RL methods can be tested and benchmarked to find an optimal solution to the defined optimization problem. The training can occur offline, before deployment of the deep-RL agent in an operational 5G network, by using 5G testbeds, simulated environments, or digital twins; or it can also occur online during its deployment. It is reasonable to perform both types of training as each one will address a specific problem: 1) chaining MTD operations randomly can be costly, hence, the deployed RL agent should have an off-line trained policy to avoid groundless decisions; 2) With online training the RL policy can adapt to the dynamics of the network, leading to better decision making over time.

The deep-RL algorithms benchmarked in this work come mainly from two distinct families of RL: 1) legacy single-objective deep-RL and 2) multi-objective deep-RL algorithms. While the latter takes the vector reward $\overline{R}$ as it is, the former is trained by scalarizing $\overline{R}$ to a single reward value. From the single-objective deep-RL class, four algorithms are evaluated: a deep Q-learning network (DQN) [153], an advantage actor-critic (A2C, synchronous version of A3C [74]), a proximal policy optimization (PPO) algorithm [75], and a variation of PPO with action masking (MaskablePPO) proposed in [154].

DQN uses a convolutional neural network (CNN) with three convolutional layers that extract 512 linear features. For the CNN, the observation vector of the MOMDP is converted to an RGB image. The other three algorithms use a multi-layer perceptron (MLP) NN with 64 layers of 64 perceptrons and a Rectified Linear Unit (ReLU) [155] activation function at each layer.

From the multi-objective deep-RL class, two algorithms are used: Envelope MORL [156] and the expected utility policy gradient (EUPG) [77]. While Envelope MORL uses a conditioned DQN NN to embed multiple policies, EUPG scalarizes the rewards during training based on the accrued return on rewards.

4.5.4 Implementation

The MOMDP model is implemented using MORL Gym, a variation of OpenAI Gym [157] which enables multi-objective observations served to MORL algorithms for the training and inference phases (*i.e.*, the observation environment). The mono-objective deep-RL algorithms use instead the MDP model built directly with OpenAI Gym, factorizing $\overline{R}$ to R. The deep-RL agents are implemented using Stable-Baselines3 [158] for the mono-objective deep-RL algorithms, while MORL algorithms are implemented and deployed with MORL-Baselines [159]. The OptSFC module was implemented in Python, the common denominator of the modules mentioned above, as well as all *MERLINS* solutions implemented in the thesis.

4.6 MTDFed - a Federated Multi-tenant MTD Management

MTDFed, the final solution proposed in this thesis, addresses the federated cooperation component within the *MERLINS* framework's MOL layer. Specifically, MTDFed tackles Research Question 4 (***RQ4***): '*How to distribute the control system of the cognitive MTD solution in a multi-tenant environment?*'. MTDFed introduces a centralized FL-based system that distributes multiple instances of the OptSFC framework across different VNOs. This enables the joint training of their respective MORL/deep-RL models while ensuring the confidentiality of each operator's data. This includes both the monitored metrics of their respective virtual networks, created using isolated network slices, and the confidentiality of their individual RL models.

The MTDFed architecture, illustrated in Figure 4.16, reflects the separation of VNOs while they operate within the same infrastructure. Each VNO runs its instance of the *MERLINS* framework, owning a local MTD controller and all components required in the proposed *MERLINS* HLA. This means VNOs are independently deciding which MTD actions to perform and independently enforcing them on their own services. Since the infrastructure is typically owned by one of the VNOs, which also assumes the role of an Infrastructure Communication Provider (ICP), MTDFed implements a centralized FL framework with a single model aggregator. This aggregator resides outside the network slices of the VNOs but within the same infrastructure. This architecture necessitates a level of trust in the ICP, which is considered a reasonable assumption given that VNOs must inherently trust the ICP to host their network slices within its infrastructure. As discussed later, this trust can be further enhanced by employing TEEs. This allows for cryptographic verification of the FL model aggregator through remote attestation [160], ensuring its integrity and confirming that it is running within the TEE enclave of a designated server.

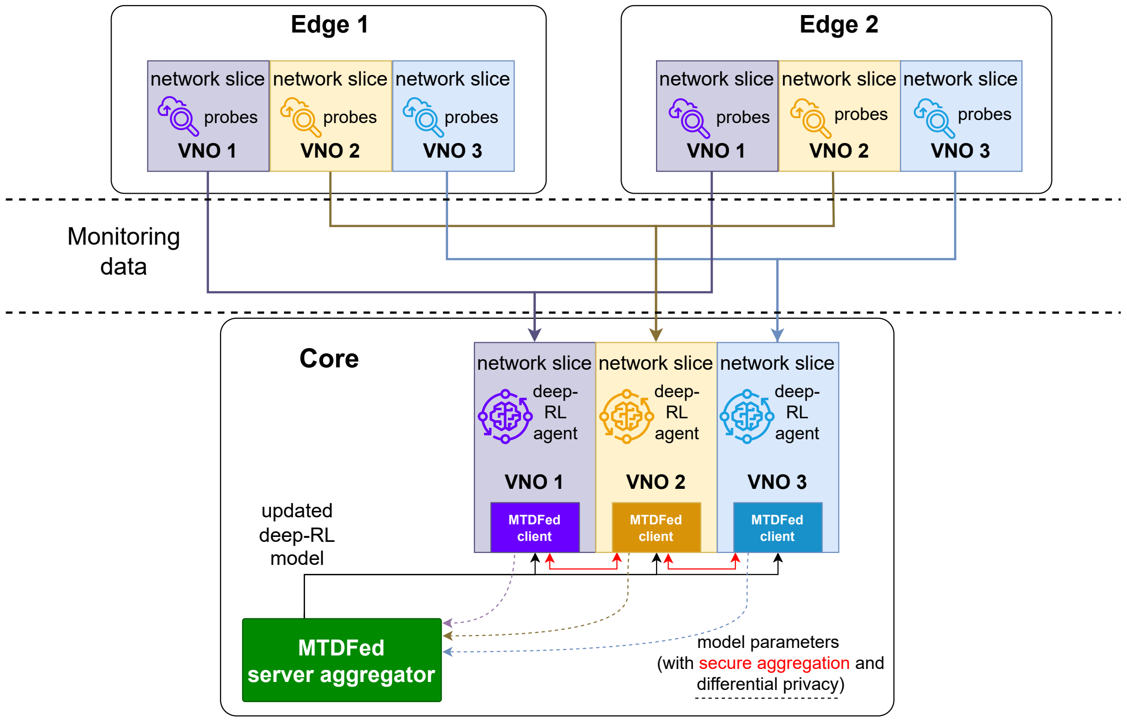
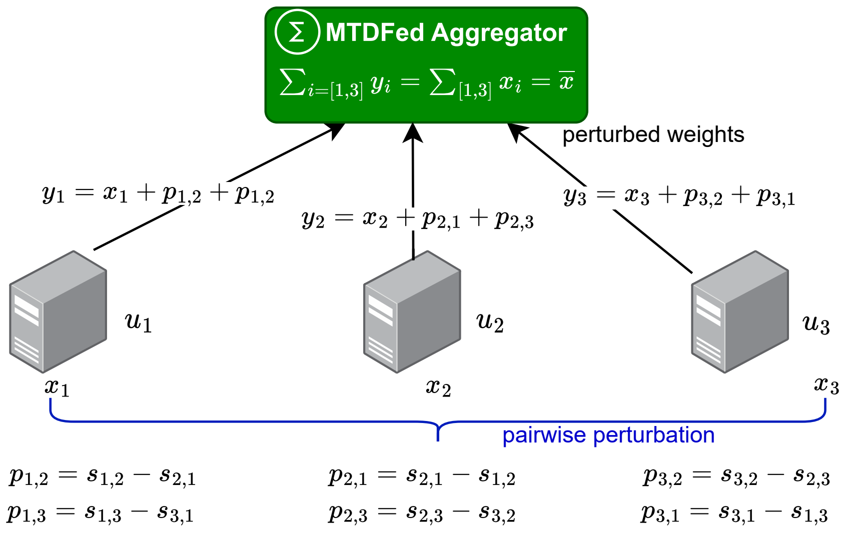


Figure 4.16: MTDFed architecture.

Privacy concerns in Data & model sharing

As an instance of *MERLINS* locally runs by each VNO, in each core and edge domain, VNOs install different network monitoring probes capturing only the traffic related to their own VNFs/CNFs, which in contrast runs by the same network slice manager and NFV MANO component of the main network, but with different access rights and management of the resources based the network slice and VIM credentials (*i.e.*, the credential to access an Openstack project space for the VIM or the network slice manager for handling VNFs/CNFs and NSs). MTDFed keeps OptSFC's approach of having a single decision-making system running in the core and deciding on the operations over all the domains of the VNO, both in the core and the edge. Thus, the data required by OptSFC for forming the MOMDP observation and deciding on the MTD action to perform is normally transferred from the edge nodes to the core, where the OptSFC agent runs. The security and confidentiality of these transfers are guaranteed by the isolation of the network slices. For the centralized FL process between the VNOs, only the deep-RL models used by the OptSFC agents are needed, while collected monitoring data are never shared.

Furthermore, the deep-RL model local to the VNO is also prevented from being obtained by other FL participants, and particularly by the ICP hosting the aggregator, as sharing the model weights equates with sharing the model, enabling the aggregator to obtain all the models of the participants. Moreover, from an NN model, malicious actors could leak the local data used to train the models using methods such as reversed inference attacks [161, 162]. MTDFed implements the secure aggregation strategy from Bell et al. [163], a Secure Multi-Party Computation (SMC) wherein a randomly selected subset of FL participants are paired to share incomplete portions of their model's weights and perform partial aggregations before the global aggregator completes the task to form the new global model without receiving the individual model of each participant. Formally, each participant $u \in U$ formats its FL updates as a vector x_u of dimension k and composed of integers on the range $[0, R)$ for some known R. The elements of the vector $\overline{x} = \sum_{u\ inU} x_u$, resulting in the sum of all the participants' vectors, should also be in the range $[0, R)$. Assuming a pair of participants u and v, u samples a vector $s_{u,v}$ uniformly from the $[0, R)^k$ for each other participant v. Participants u and v exchange $s_{u,v}$ and $s_{v,u}$ over a securely encrypted channel and compute perturbations $p_{u,v} = s_{u,v} - s_{v,u}$, obtaining perturbations such that $p_{u,v} = -p_{v,u} \pmod R$ and $p_{u,v} = 0$ when $u = v$. Each participant then sends to the aggregator $y_u = x_u + \sum_{v \in U} p_{u,v} \pmod R$. Finally, the aggregator simply sums the perturbed values as the paired perturbations in y_u cancel each other, and what is obtained is simply the sum of all participants' vectors, $\overline{x}$.



MTDFed Aggregator
$\sum_{i=[1,3]} y_i = \sum_{[1,3]} x_i = \overline{x}$
perturbed weights
$y_1 = x_1 + p_{1,2} + p_{1,2}$
$y_2 = x_2 + p_{2,1} + p_{2,3}$
$y_3 = x_3 + p_{3,2} + p_{3,1}$
u_1
u_2
u_3
x_1
x_2
x_3
pairwise perturbation
$p_{1,2} = s_{1,2} - s_{2,1}$
$p_{1,3} = s_{1,3} - s_{3,1}$
$p_{2,1} = s_{2,1} - s_{1,2}$
$p_{2,3} = s_{2,3} - s_{3,2}$
$p_{3,2} = s_{3,2} - s_{2,3}$
$p_{3,1} = s_{3,1} - s_{1,3}$

Figure 4.17: SMC secure aggregation workflow.

4.6.1 Federating the deep-RL Agents

In the context of OptSFC, which uses deep-RL to train the decision-making agents, MTDFed has to apply FL in the training of its deep-RL agent. In the case of the actor-critic methods, namely PPO and A2C single-objective algorithms, two NNs are deployed: an actor network and a critic network. The actor network (or policy network) directly implements the agent's policy, *i.e.* the mapping between a network state and the relative optimal MTD action to be taken; thus, this is the agent's main target model to be trained collectively. On the other hand, the critic network (or value network) implements the value function, which estimates the value of being in a particular state (*i.e.* $V(s)$) or taking a particular action from a particular state (*i.e.* $Q(s, a)$). While critic networks are typically environment-specific, in the context of *MERLINS* and the OptSFC solution, since both the action set and the MOMDP model are the same for all the participants, MTDFed aggregates both the actor and the critic models of the deep-RL agent.

The aggregation includes both weights and bias parameters, the latter adjusting the models' output and playing a key role in the models' behavior and performance. Nevertheless, MTDFed does not aggregate the parameters of optimizers such as the Adam gradient-based stochastic optimization [164]. This aggregation in MTDFed is omitted for four main reasons:

- **resilience to data heterogeneity**: participants can adapt their optimizers to their specific data distributions, allowing for better local learning. This is beneficial for non-independent and identically distributed (non-IID) data. In this specifically targeted scenario of NVOs, this can be related to certain characteristics of VNFs/CNFs, such as the exposure of a service to vulnerabilities, the size of a service, or the different types of resource and network loads as explored in the ContMTD solution (*c.f.* Section 4.3.2).

- **stability in training**: Optimizer parameters tend to reflect recent gradients, which may not generalize well across clients. Keeping them local avoids instability during aggregation.

- **faster aggregation**: Aggregating only the weights and biases without optimizer states reduces the communication overhead.

- **practical precedent**: Most FL systems, including the canonical FedAvg paper, aggregate only the model parameters and leave the optimizer states [165].

A risk of not synchronizing the optimizers is that the different NVOs diverge at each aggregation round and end up with very different optimizer states, leading to the aggregation of conflicting models and ultimately decreasing the performance and convergence time of the deep-RL agents. For this reason, at every aggregation round, MTDFed reinitializes the optimizer of all NVOs, ensuring the same momentum-free model for all participants each round and mitigating the risk of optimizer-induced divergence.

4.6.2 Improvements on OptSFC's MOMDP Model

During the MTDFed implementation, the MOMDP model underwent several refinements. To significantly reduce the exploration space (i.e., the total number of possible states), the following optimizations were implemented:

- **Observation's reduction**: Some features considered non-relevant based on the OptSFC evaluation were removed, such as the parent NS and NSi for each running VNF/CNF. Redundant features were also removed, such as the location feature, which is redundant with the "VIM_-host" feature.

- **Threat and risk values scaling**: Reduced the range of threat and risk features from $[-100^3, 100^3]$ to $[0, 10]$. This scaling was also facilitated by the subsequent update of the reward function.

The initial reward function incorporated an ASP that increased over time to represent the attacker's advantage against a static target. However, this resulted in the ASP rapidly converging to the upper bound, leading to a constant maximum penalty after a few steps. To address this, the reward function was modified to incorporate a sigmoid function, enabling a controlled, S-shaped increase in ASP:

$$p(t) = p_{\text{start}} + \frac{p_{\text{end}} - p_{\text{start}}}{1 + e^{k\cdot(tT/2)}}$$

where $p(t)$ represents the ASP at time t, p_{start} the initial ASP value, p_{end} the maximum ASP value (*i.e.*, one), T the total duration or timeframe considered (*i.e.*, one month), and k the constant that controls the steepness of the sigmoid curve. Following this formula, the reward function provides a more realistic representation of the gradual increase in an attacker's ASP and enables the deep-RL agent to more effectively perceive the evolving threat landscape during the exploration phase.

The MOMDP model was further extended to accommodate container-based CNFs. While the general modeling approach for CNFs remains similar to that of VNFs, key distinctions arise due to the inherent statefulness of container-based deployments. Specifically, stateful LiMis do not effectively remove malware from the service. Consequently, the ASP reset for malware and backdoor-related threats, as well as the associated LiMi MTD action reward, were removed. Only network-based attacks, such as reconnaissance and DDoS, are subject to ASP resets. Furthermore, the model incorporates a budget for LiMis and reinstantiations of CNFs. Based on performance measurements from both MOTDEC (for stateless VNFs) and ContMTD (for stateful containers), a budget was established for the number of operations permissible per service, assuming a 99.95% monthly availability guarantee.

To enhance training efficiency, two distinct variations of the MOMDP were designed and evaluated:

1. **Episodic task**: Despite the inherently continuous nature of the MTD management and orchestration task, the training process was transformed into an episodic framework. Each episode terminates at the end of the month or upon depletion of the MTD actions budget. The rewards for the remaining period are calculated and assigned in the final timestamp of the episode. This approach significantly increases the number of ”months” experienced by the agent within a given training period.

2. **Continuous daily budget task**: The training process remains continuous, but the MTD actions budget is divided into daily allotments. This constraint prevents extended periods of inactivity, thereby accelerating model convergence. However, this restriction may limit the exploration of diverse MTD strategies, potentially leading to sub-optimal solutions. Despite this potential limitation, this variation was included in the evaluation, as preliminary observations suggested that optimal solutions often exhibit a frequency of at least one MTD action per service and per day.

These variations effectively address the issue of inefficient training in the original MOMDP, where the agent often spent significant time selecting actions that would not be executed due to budget constraints. This not only wasted computational resources but also hindered the agent's ability to learn long-term strategies, as a substantial portion of the training time was effectively unproductive.

4.6.3 MTDFed Implementation

MTDFed's Python implementation uses the Flower FL framework [166], providing secure aggregation and differential privacy features. Differential privacy further prevents FL participants from exposing their local models to the aggregation server, *i.e.* the ICP, which is assumed to be also a network operator and a participant of MTDFed for federated MTD strategy optimization.

4.7 *MERLINS* Overview and Key Takeaways

This section outlines the *MERLINS* methodology, detailing the phases, components, functionalities, and interfaces required for a manageable closed-loop orchestration of MTD operations. This approach introduces an uncertainty-based security layer to networks, which have become increasingly flexible with the advent of NFV and softwarized networks. This work is relevant in identifying, based on the highly regularized and standardized telecommunication ecosystem, how to integrate an additional proactive security layer without requiring changes to the defined architectures, such as the NFV, and rather, using the NFV standard to enable it.

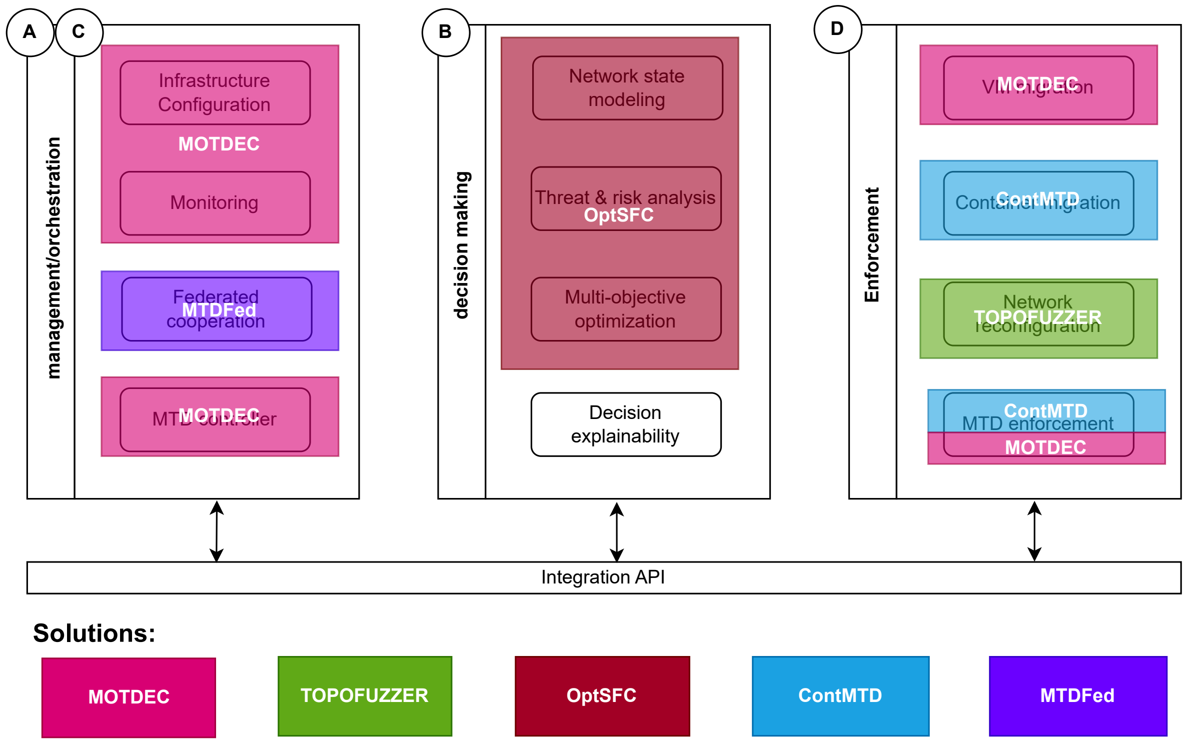


Figure 4.18: Overview of solutions covering the *MERLINS* HLA.

The *MERLINS* HLA can be further extended to incorporate new MTD actions or address additional requirements. For instance, the explainability component conceptualized in the HLA at the DML layer was not implemented in this thesis due to the limitations of the scope defined by the research questions (RQs). Nonetheless, the solutions cover the most critical components of the *MERLINS* layers, providing a functional implementation of the HLA, as illustrated in Figure 4.18. Here, each solution addresses one or more HLA components. For instance, MOTDEC implements the NBI, infrastructure configuration, monitoring, VM migration, and the relevant MTD controller's operations, while ContMTD implements the MTD controller's operations focused on container migration.

The contributions outlined in this chapter provide answers to all the RQs under investigation in this thesis: **RQ1**, **RQ2**, **RQ3**, and **RQ4**. **RQ1** is answered with the MOTDEC, TopoFuzzer, and ContMTD solutions, providing different types of MTD actions targeted at different threat scenarios and with different enforcement costs. **RQ2** and **RQ3** are addressed with the OptSFC and MTDFed solutions, which define a formal modeling of 5G/B5G NFV-based networks as a MOMDP. The model is utilized for training and near-real-time decision-making of MTD strategies using deep-RL. Finally, **RQ4** is answered with the MTDFed solution, designing and implementing a multi-tenant distributed approach to OptSFC, leveraging secure and privacy-aware FL.

The following chapter (Chapter 5) further elaborates on the answers to these RQs with the evaluations and discussions of each proposed solution.

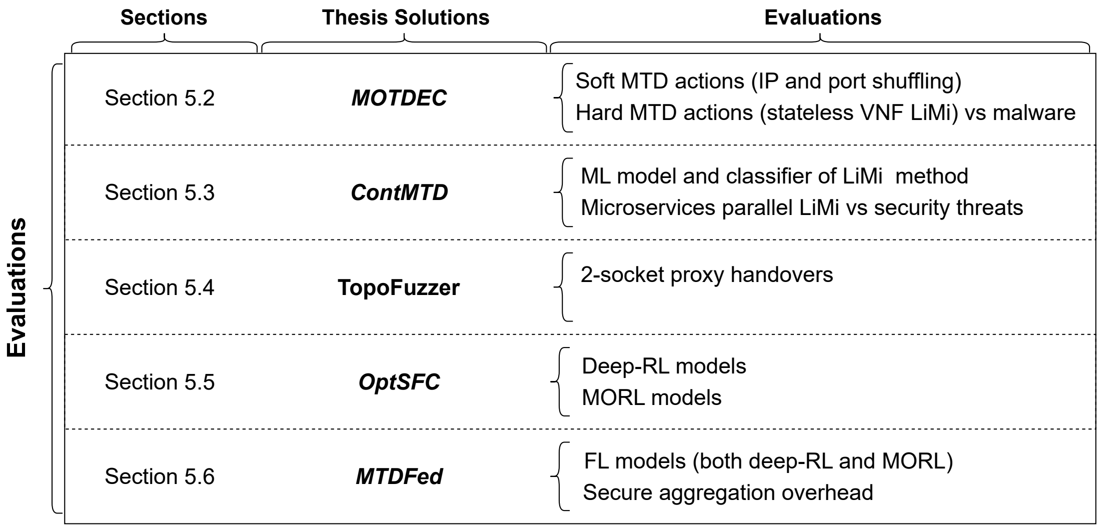
5 Evaluations and Discussions

THIS chapter evaluates the solutions implemented within the *MERLINS* framework. The primary objectives of such evaluations are to assess the feasibility of a cognitive MTD framework like *MERLINS* in orchestrating novel MTD mechanisms in Telco Cloud environments, demonstrate its practical application within a realistic 5G testbed, and provide measurable results for each implemented solution following the structure depicted in Figure 5.1.

While the testbed used in this thesis is built upon a 5G core (6G standards being not developed yet at the time), its architecture is deliberately aligned with the evolutionary trajectory of Telco Cloud networks. The infrastructure relies on the key cloud-native, software-defined, and fully virtualized principles that constitute the foundation for 6G's anticipated architecture, further characterized by network function management through AI-driven orchestration systems. Thus, by evaluating the orchestration of *MERLINS* within this environment, we are validating the MTD security paradigm against the operational agility required for the software-driven, AI-controlled, and zero-touch networks of the future.

Hence, the remainder of this chapter is organized as follows. Section 5.1 presents the 5G testbed developed for this thesis to evaluate the *MERLINS* framework. Section 5.2 evaluates the MOTDEC solution, focusing on the performance of *soft* and *hard MTD actions* (detailed in Section 5.2.1 and

Section 5.2.2, respectively). Section 5.3 evaluates the ContMTD solution, with a specific focus on optimizing the LiMi for microservices and interdependent CNFs. Section 5.4 evaluates TopoFuzzer's performance in the handover of TCP and QUIC-based connections during MTD operations. Section 5.5 presents the results of strategy optimization achieved by the OptSFC solution, while Section 5.6 evaluates the MTDFed solution. To conclude, Section 5.7 summarizes the key findings and conclusions drawn from this chapter.



Sections
Thesis Solutions
Evaluations
Evaluations
Section 5.2
MOTDEC
Soft MTD actions (IP and port shuffling)
Hard MTD actions (stateless VNF LiMi) vs malware
Section 5.3
ContMTD
ML model and classifier of LiMi method
Microservices parallel LiMi vs security threats
Section 5.4
TopoFuzzer
2-socket proxy handovers
Section 5.5
OptSFC
Deep-RL models
MORL models
Section 5.6
MTDFed
FL models (both deep-RL and MORL)
Secure aggregation overhead

Figure 5.1: Structure of the evaluation with respect to the *MERLINS* solutions.

5.1 5G Testbed

Fig. 5.2 illustrates the topology of the 5G testbed where we evaluated the MERLINS framework. It comprises two cloud environments, operated with OpenStack; one for the Edge and Radio Access domains, termed "Edge NFVI", and another for the Core domain, *i.e.*, the "Core NFVI." This deployment implements a distributed UPF architecture, where UPFs are co-located with the base stations (gNBs) in the Edge domain. The Edge NFVI includes Radio Access elements (5G UEs and gNBs) and Edge Cloud elements (the UPF and a VNF that provides an application service to connected users). The Core NFVI hosts the control plane of the 5G Core Network, the subscriber database, and generic VNFs for service provision. The Core NFVI also hosts the Katana Slice Manager [140], and the NFVO implemented with OSM. The proposed solutions implemented in the MERLINS framework are also hosted in the core NFVI. In addition, the TopoFuzzer is deployed at the Edge NFVI.

This deployment allows emulating a distributed UPF architecture, where the UPFs are co-located with the gNB on the Edge domain.

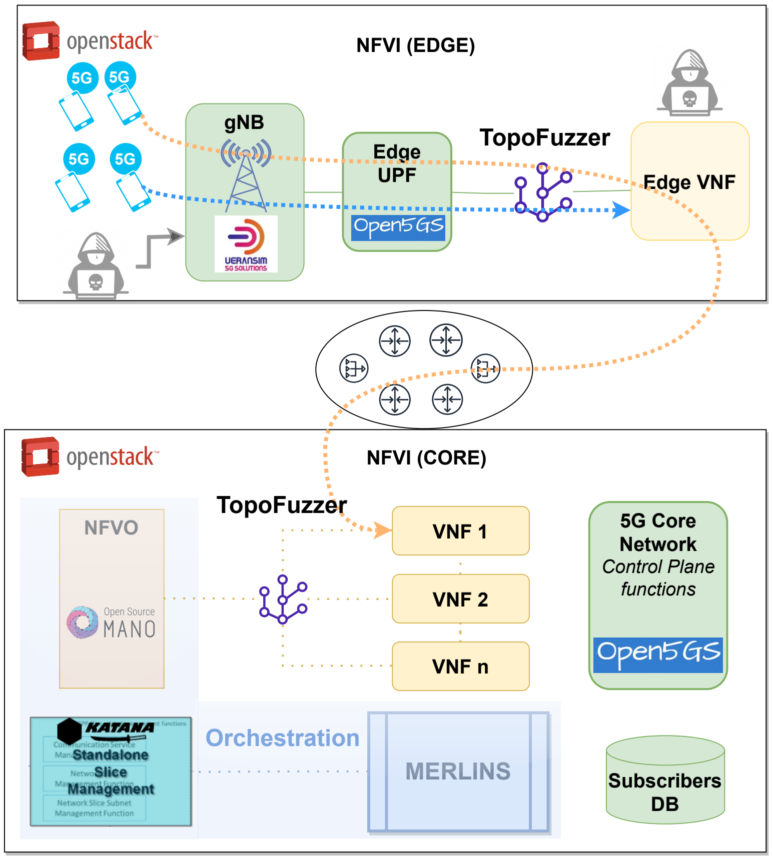


Figure 5.2: 5G Testbed Deployment for *MERLINS* evaluation.

The 5G Core is implemented with Open5GS [167], an open-source 3GPP Release-16 compliant 5G core. Open5GS provides the following network functions as discrete services, allowing the separation of the control and data planes: *(i)* AMF, *(ii)* SMF, *(iii)* UPF, *(iv)* AUSF, *(v)* NRF, *(vi)* UDM, *(viii)* PCF, and *(ix)* NSSF.

The RAN and mobile UEs are implemented by UERANSIM[168], an open-source UE and gNB simulator. The 5G architecture is Standalone (5G SA). UERANSIM connects to Open5GS via a control interface with the AMF and a user interface to the UPF. The UEs and the gNBs connect via a simulated radio interface. UERANSIM does not support signaling procedures in the PHY, MAC, RLC, and PDCP layers. It focuses on protocols found in the RRC layer and above. However, this does not affect evaluation results, since the focus is on the operational cost of MERLINS MTD actions, which is reflected in the QoS in the application layer. Unlike actual hardware equipment, UERAN-

SIM allows the deployment of a significant number of virtual UEs to test the solution's scalability under an increasing network workload.

Ethical Considerations:

All the experiments performed in this testbed follow ethical norms and considerations about privacy and handling of sensitive data. The user traffic generated is entirely artificial and does not correlate to any activity of real users. Simulated attacks, such as malware infection and C&C remote control, are performed within the limits of the isolated 5G testbed built for this thesis.

5.2 Orchestrating and Enforcing MTD Actions using MOTDEC

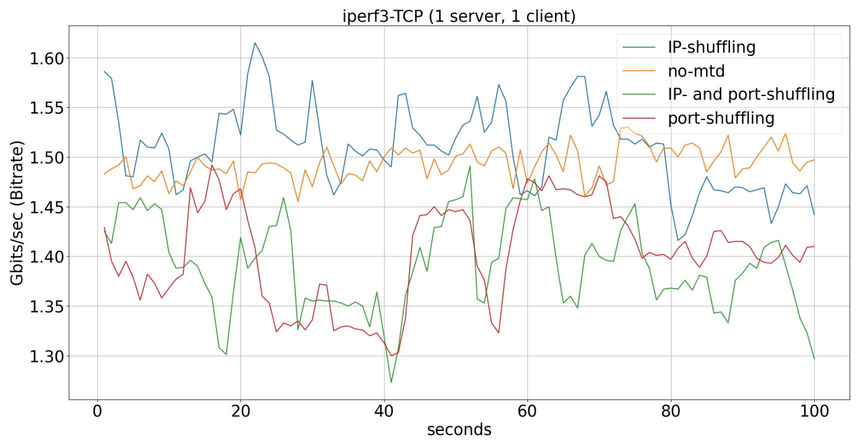
This section presents the results obtained from MOTDEC, specifically, from the evaluation of the *soft MTD actions* and the *hard MTD actions* implemented and tested on the testbed.

5.2.1 Soft MTD Actions: Experiments and Results

To evaluate the performance of *soft MTD actions*, specifically IPv6 address and port shuffling [139], experiments were conducted on a varying number of client-server (C-S) pairs, with servers being VNFs performing the connection shuffling for MTD.

The performance was assessed using a combination of tools, namely `iperf`, for measuring the bitrate, `ping`, for measuring the RTT, and Dynamic Adaptive Streaming over HTTP (DASH) [169], for measuring data stream performance. Each performance metric was measured 10 times, and the mean values were calculated to account for variations inherent in the network infrastructure. The tool `iperf3` was used to create streams inside the SDN infrastructure and measure the overall bandwidth of TCP and UDP connections for different numbers of hosts. To ensure consistent results, both TCP and UDP connections were conducted at a fixed bitrate of 2 Gbps, thereby minimizing potential variations in UDP performance. Each `iperf3` run lasted 200s. To ensure proper initialization of `iperf3`, the analysis did not consider the first 100 s of each evaluation run. This approach allowed sufficient time for `iperf3` to stabilize and provide accurate performance measurements.

With the iperf3 traffic running in the background, the RTT was measured using the `ping` command being run continuously for 100 s. The average latency over this duration was used for comparison. To conduct practical performance measurements at the application layer, the DASH streaming protocol was utilized along with the GPAC tool, again with the iperf3 traffic running in the background. GPAC creates streaming-ready files from media content and provides an integrated client for viewing and



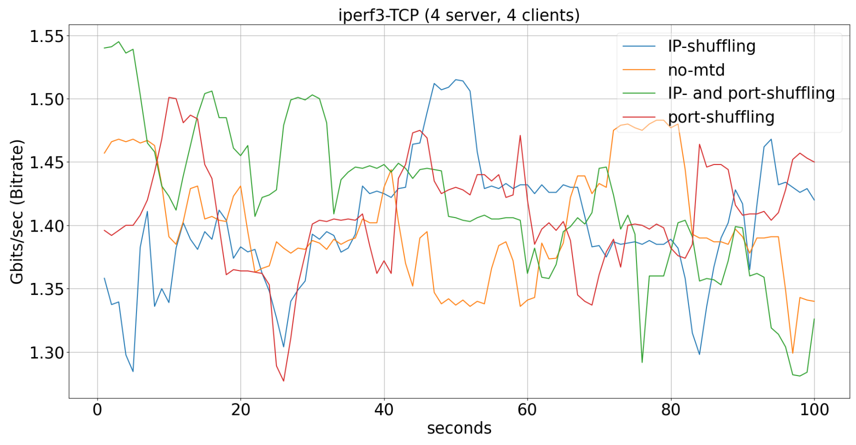


Figure 5.3: `iperf3` TCP evaluation with 1, 4, and 100 servers (each with the corresponding number of clients).

analyzing the streamed media [170]. The media to be streamed is a 7-minute-long video placed on one of the servers acting as a Web server [171]. The client then connects to the Web server, which sends the stream data with the additional meta files to the client.

Results

The TCP and UDP columns in Table 5.1 are generally coherent, but an interesting observation is that the RTT values decrease as the number of active iperf streams increases (*blue highlighted*). One possible reason for this is the CPU scheduler's optimization, which reduces context switches and, therefore, prioritizes the Open-VSwitch (OVS) forwarding process over other processes. It is also visible that the overall iperf3 bit rate for the UDP case performs better than TCP if the load on the system does not rise.

There is a significant decrease in the DASH video stream's quality for case *8S-8C* when compared to case *1S-1C* and *4S-4C*, visible in Table 5.1. This is due to the fact that the load on the system with sixteen vCPU cores rises due to the number of iperf3 sessions and therefore the DASH video stream performance decreases. Consequently, unpredictable behavior of the performance arises, which can also be seen on the iperf3 bit rate values for the *8S-8C* case (*yellow highlighted*) and in Figure 5.3. The overhead of the proposed solution is minimal, as we assume that CPU scheduling is significantly more effective in performance simulations. It is also evident that the Port Shuffling technique negatively influences the performance of the GPAC DASH video stream use case (*green highlighted*). This observed phenomenon might be due to the addition of an extra layer. In IP shuffling, only the L3 header values (such as the checksum) are altered. When port shuffling is active, the L4 header (including the L4 checksum and fragmentation) is also involved. Consequently, in a heavily loaded environment, this can significantly decrease performance.

TCP / UDP Performance

Regarding the TCP performance, the *no MTD* mode is, as expected, the most stable variant, which is visible in Figure 5.3. In Figure 5.3, it is also visible that all modes have an increasing number of outliers because of the unpredictable behavior once the load of the system rises, as mentioned in Section 5.2.1. The performance gap remains marginal when both IP Shuffling and Port Shuffling are implemented, compared to when only one of them is enforced. The results in Figure 5.3 with *100S - 100C* exhibit higher variations across all modes due to the presence of outliers. However, they also demonstrate similar performance levels for all modes, suggesting that the overhead of the MTD shuffling method is minimal when the system is subjected to elevated traffic (*i.e.*, iperf3 traffic in this case). UDP results share many similarities with their TCP counterparts and are, thus, omitted in Figure 5.3.

Table 5.1: Performance measurements for the *soft MTD actions* (C: Client, S: Server. The reported bit rates are average values per C-S pair).

Case	Mode	Ping RTT [ms]		DASH video stream bit rate [Mbps]		`iperf3` bit rate [Gbps]	
		TCP[a]	**UDP**[a]	**TCP**[a]	**UDP**[a]	**TCP**	**UDP**
1S,1C	no MTD	0.0745	0.055	1295	1491	1.494	1.684
	IP	0.05	0.052	1450	1630	1.509	1.708
	port	0.051	0.055	159	285	1.402	1.781
	IP+port	0.055	0.051	147	266	1.392	1.583
4S,4C	no MTD	0.065	0.045	1545	1735	1.401	1.672
	IP	0.052	0.048	1588	1604	1.402	1.547
	port	0.051	0.054	213	324	1.409	1.517
	IP+port	0.045	0.053	237	313	1.419	1.542
8S,8C	no MTD	0.041	0.042	1350	1051	1.546	1.557
	IP	0.037	0.054	1078	1061	1.374	1.598
	port	0.041	0.036	493	486	1.502	1.465
	IP+port	0.038	0.052	421	421	1.416	1.542

[a]The TCP and UDP columns under "Ping RTT" and "DASH video stream present the" measurements with the TCP and UDP `iperf3` traffic running simultaneously in the background.

Table 5.2: Parameters Used in Section 5.2.1.

Parameter	Definition
N_1	Average number of attempts needed to guess the address
N_2	Average number of attempts needed to guess the port
n	Number of addresses in the address space
m	Number of ports in the port range
I	Shuffling interval in seconds
S	Scanning rate (scans/seconds)
T	Time to guess the address/port in seconds
P(X)	Probability of guessing the correct address/port

Proactive Security Gain

To analyze the security impact of the scheme developed, both theoretical and simulated approaches are applied. The security of the targeted system is measured with no MTD, with only IP Shuffling, with only Port Shuffling, and with both MTD techniques combined. The parameters used in the formulas are listed in Table 5.2.

Case 1: Undefended Network – We determine the average number of attempts required by an attacker to correctly guess addresses or ports in a non-MTD network as $N_1 = \frac{n}{2}$ and $N_2 = N_1 + \frac{m}{2}$.

In the context of port scanning, it is assumed that the port can only be identified once the IP address of the device is known. The time required for an attacker to make an accurate guess of the correct address is then $T = \frac{N_1}{S}$ for the average case and $T = \frac{n}{S}$ for the worst case. Similarly, the duration for an attacker to accurately guess the correct port is $T = \frac{N_2}{S}$ for the average case and $T = \frac{n}{S} + \frac{m}{S}$ for the worst case.

Case 2: Network with MTD – In the context of MTD, an additional metric is introduced and denoted as "z," which is calculated as the reciprocal of the number of shuffling operations performed during a scan. This quantity is defined by the following equation:

$$z = \frac{S * I}{n}$$

The likelihood of correctly guessing the IP address in an MTD network, where only IP Shuffling is implemented, is as follows:

$$P(X) = \sum_{y=1}^{\infty} z(1 - z)^y$$

In this formula, y is defined as the current number of utilized full scans. The outcome of this formula will approximate a value at which the probability of correctly guessing the address/port becomes confident, with the speed of convergence depending on z and y. To determine the average number of attempts required for an attacker to guess the address correctly, the following formulas are utilized:

$$N_1 = \frac{n}{P(X)} = n \qquad since \qquad P(X) = 1$$

The duration it takes for an attacker to guess the correct address successfully is:

Average case: T = $\frac{N_1}{S}$ and Worst case: T = `Never`

These calculations are based on the assumption that a single source address is utilized for scanning. However, when multiple sources are involved in scanning, the necessary number of attempts also becomes valid. Nonetheless, the scanning durations should be divided by the number of sources involved.

In this context, P(X) equals 1 when *z* falls within the range of 0 to 1 $(0 < z < 1)$, which is the case for shuffling. Consequently, it appears that shuffling effectively doubles the value of N_1 compared to the scenario without MTD. Given that Port Shuffling operates on similar principles, the combination of IP Shuffling and Port Shuffling results in a fourfold increase in N_1 compared to the scenario without MTD. Interestingly, P(X) and N_1 remain unaffected by the scanning rate or shuffling interval.

An intriguing observation is that, in the worst-case scenario, an attacker may never successfully identify the correct IP or port. Several numerical simulations have been conducted to determine how many full scans are required for a successful attack when IP Shuffling is implemented. As depicted in Figure 5.4, the results indicate that, on average, an additional full scan is needed to find the target host with an 87% probability. After a third full scan, the success probability is above 95%. Remarkably, this outcome remains consistent regardless of the scanning rate or shuffling interval. Furthermore, attackers typically halt the scanning process if a full scan yields no results. Consequently, the actual value of shuffling may be the act of discouraging the attacker from rescanning the network when nothing is found with the first full scan.

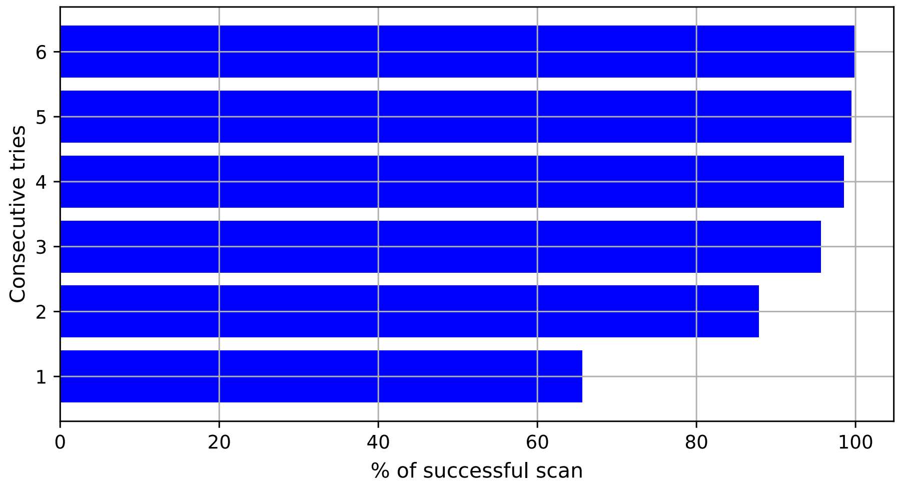


Figure 5.4: Cumulative probability to find a moving host per number of full address-space scans with activated *soft MTD actions*. In contrast to these results, without *soft MTD actions*, the first scan is always successful (*i.e.*, 100%).

Illustrative Case Study: IPv6 Network with /104 Mask

This section evaluates the soft MTD action security gains in a specific scenario by applying the equations from Section 5.2.1. An IPv6 network with a /104 mask is assumed, and IP and Port Shuffling are enabled. This allows for 16^6 unique addresses. An attacker has gained access to the network and is trying to locate the IPv6 address of a device called `srv-1`. By using $N_1 = \frac{n}{2} = \frac{16^6-1}{2}$, the attacker would require 8,388,607.5 attempts on average to locate one of the vulnerable servers. `Nmap` estimates a scan of all 65,536 ports to take 21 min. By dividing 21 minutes by the number of ports, a scan of a port takes roughly 0.02 seconds. The same value is used for the IP scanning operation since the scanning rate is similar to port scanning in that case, as explained above. If no soft MTD action is performed, the attacker would require 167,772 s or 46.6 h to execute the 8,388,607.5 attempts to scan IP addresses of an undefended network. If only IP Shuffling is activated, it takes 93.2 h, doubling the 46.6 h. This is long but still feasible for an attacker to find, for instance, a server IP address in the network. However, if port shuffling is also activated, the attacker must scan 65,536 ports to see if an IP address is communicating. Now, the attacker needs 697.3 years on average to locate the correct IP address and port number. Obviously, the minimum time (i.e., the best-case scenario for an attacker) to find a server is still 0.02 s.

5.2.2 Hard MTD Actions: Experiments and Results

This evaluation, as illustrated in Figure 5.5, showcases the integration of MOTDEC to OSM and Katana in a realistic 5G network and has a practical demonstration of hard MTD actions enforced on VNFs, focusing on a reactive security scenario against a malware infecting a stateless VNF service and tampering its filesystem. The VNF service implements a generic application exchanging data with UEs through its REST API interface, which can be HTTP/2 or HTTP/3 based, and is executed in a container within the VM, the latter representing the VDU of the VNF, deployed in OSM.

It is important to note that this section strictly focuses on MOTDEC's evaluation of stateless VM-based VNFs, in which the QoS overhead occurs only due to the service downtime from the traffic redirection and connection handover handled by the TopoFuzzer module. Hence, these results directly depend on the results of TopoFuzzer, additionally detailed in Section 5.4. Finally, no use of a cognitive decision-making system in OptSFC is used, as this is evaluated separately in Section 5.5, and a static security policy is used instead.

Hence, OptSFC is deployed as a reactive system interfaced with Solidshield Systemic, a tampering detection system performing binary integrity checks at run-time [172]. A proof-of-concept C&C malware, for which remote control is emulated with a REST API interface, is installed in an edge VNF of the 5G testbed. When triggered, the PoC malware modifies the binary code of the running VNF application, simulating a tampering attack (*i.e.*, upon receiving the request, the VNF application corrupts its binary). As Systemic detects the attack, OptSFC receives the attack alert, which enforces a static security policy for the specific threat. This static policy re-instantiates the application at the container level first, then at the VM level, in case the attack is detected again, and finally, if the attack is performed again, the VNF is migrated to the other NFVI. The first MTD action neutralizes the attack if the malware is installed in the application's container. The second MTD action re-instantiates the VNF, removing the malware, while the third addresses issues where the NFVI node would be the attack vector for the malware infection.

In this scenario, the focus is on evaluating the feasibility and potential QoS overhead of MTD migration and re-instantiation actions on stateless VNFs. To this end, a test is conducted on 10

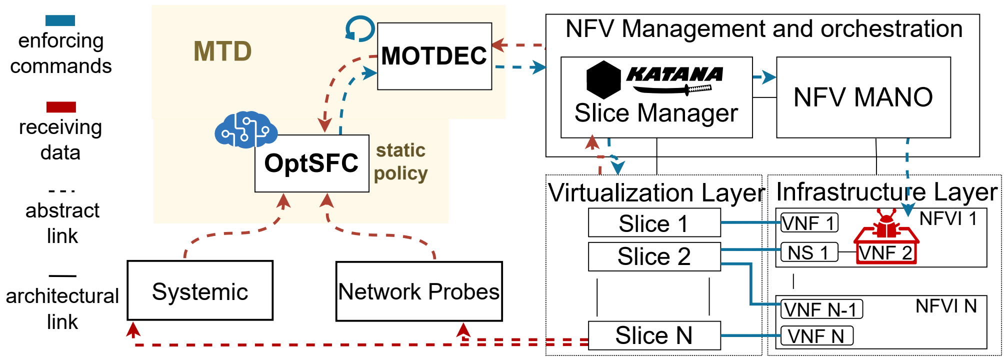


Figure 5.5: Evaluation setup for MOTDEC *hard MTD actions*.

UEs simultaneously communicating with the target VNF during the MTD operations. Two types of client connections are tested: HTTP/2, based on the TCP protocol, and HTTP/3, based on the QUIC protocol, built on UDP. The QoS overhead is evaluated based on the communications' latency and packet loss rate during the MTD actions. Results show that the highest MTD network overhead occurs with HTTP/2 and TCP connections, with an average 7% increase in the packet loss rate in a one-second timeframe for re-instantiations and 33% for migrations, resulting in an average service downtime of 330ms. This overhead reflects the performance of the TopoFuzzer connection handover, also detailed in Section 5.4.

Ultimately, *Hard MTD actions* on the attacked VNF with an authenticated uninfected image of the service, coupled with the seamless handover of client connections performed with TopoFuzzer, moves both UE connections and the VNF application and neutralizes the tampering attack's effects. Moreover, to iterate the attack, the attacker's effort increases drastically as they have to re-infect the VNF, possibly a non-trivial task. This gives the service provider more time to find the attack vector and rectify it.

5.2.3 Discussion and Limitations

The evaluation on MOTDEC *soft MTD actions* shows that IP shuffling in the large IPv6 address space, combined with port shuffling, creates the entropy needed to obfuscate and slow down attackers. Performance and security-related results indicate that it is applicable in practical use cases. However, further research is needed to analyze different factors on the shuffling performance. A concern is that clients should know how to communicate with the shuffled host. This means that signaling is needed for distributing dynamic connection information, or an existing dynamic DNS implementation must be extended to save port numbers in DNS records. Another concern is the "Address Block Ownership". If the defender does not own an IPv6 block, they can use a purely local network if the hosts do not need an Internet connection. In the other case with an Internet connection, they can use a single address combined with NAT for IPv6, *e.g.,* NAT66.

The evaluation on MOTDEC *hard MTD action* demonstrates a simple yet powerful mechanism for removing backdoors and potentially disruptive malware from a stateless VNF. However, as stated on previous occasions, such an operation is not feasible for stateful services, where maintaining continuous operation and state consistency is critical. The ContMTD solution, evaluated in the following section, is a natural continuation addressing the broader challenges of protecting stateful services by minimizing downtime during MTD actions and ensuring state consistency.

5.3 Orchestrating Microservices Live Migrations using ContMTD

This section presents the results obtained from ContMTD, the variations made to the testbed to evaluate the solution, the performance of its regression models, as well as the outcomes of applying the ContMTD scheduling algorithm to microservices applications. We compare the performance measurements of ContMTD against those of the methods implemented in CloudHopper [56], which represent the current state of the art in parallel container LiMi. Finally, we showcase the preliminary security evaluation of ContMTD against two main threat models: side-channel attacks and undetected malware infection and propagation in a multi-tenant multi-service provider scenario.

Experimental Setup

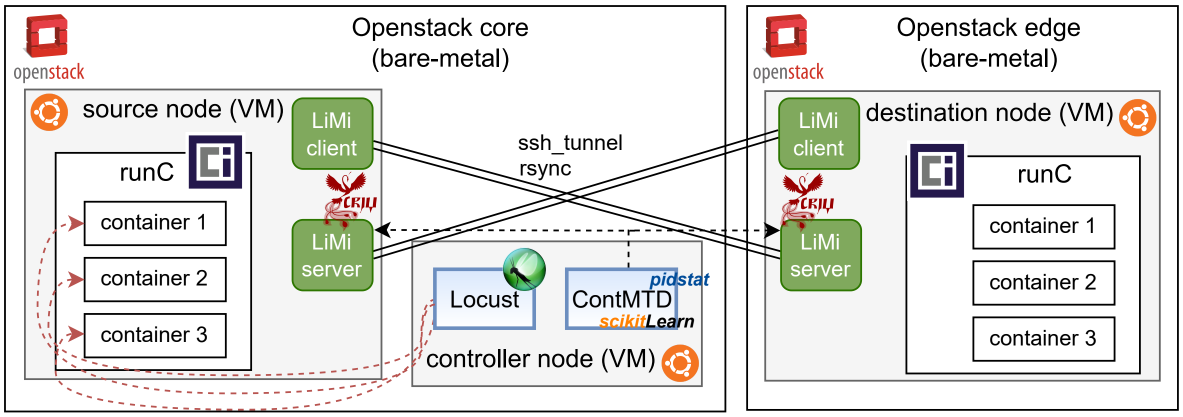


Figure 5.6: Cloud infrastructure used for ContMTD's experiments.

The migration experiment is conducted between the code and edge OpenStack VIMs of the 5G testbed, but without passing through the NFV MANO and network slice manager. This is because OSM MANO orchestrates CNFs using Kubernetes, which does not enable the LiMi optimization algorithms such as pre-copy, post-copy, and hybrid. Thus, a separate controller that manages runC containers is implemented in ContMTD alongside the MOTDEC controller for VNFs. Two Ubuntu 22.04 VMs are deployed on separate VIMs, representing the microservices orchestrator, built upon the runC container runtime. Each VM is provisioned with 4 virtual CPUs (vCPUs), 8 GB of RAM, and 80 GB of SSD storage and is interconnected with a 10 Gbps network link. The CRIU library is used with *runC* to perform the pre-copy, post-copy, and hybrid migration optimizations, while at the orchestration level with Docker and Kubernetes, CRIU is still integrated with limited optimization support. Finally, we run an additional Ubuntu VM with 2vCPUs, 4GB of RAM, and 40GB of SSD storage. This VM hosts the ContMTD framework and its modules, with the exception of the LiMi clients and servers of the LiMi controller module. These are installed in the interchangeable source and destination nodes to enforce the CRIU migration both ways.

5.3.1 Experiments and Results

After training the regression models estimating the downtime and total migration time on the created dataset, we evaluate each model individually and then evaluate the LiMi method classifier of ContMTD that uses the downtime regressor against the statistical model and the CloudHopper method, the latter simply using pre-copy on all selected containers. The results of the evaluation are presented in Table 5.3. The regressor models are evaluated using the coefficient of determination (R2) and the Mean Squared Error (MSE), defined as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y_i are the observed values, $\hat{y}_i$ are the predicted values, $\bar{y}_i$ is the mean of the observed values, and n is the number of observations. The results (shown in the last two columns of Table 5.3) indicate random forest and NN as the regression models with the best performance. Results also reveal that the total migration time is much easier to predict with a very small error margin, down to 1.07 seconds for random forest and NN, compared to the downtime, with 3.36 seconds for the same models, based on the square root of MSE given in the table.

The classifiers' accuracy is calculated using 10-fold cross-validation, each fold using a different 10% of the dataset as the test set, excluding it from the training phase. We also use stratified folds, ensuring that each fold has a similar balance of container types as the original dataset, avoiding biases that could be introduced by imbalanced classes when splitting the dataset.

The ground truth for the accuracy calculation is the LiMi method with the lowest downtime based on the average values per container category of the test set used at each fold. The results (listed in the first column of Table 5.3) show that the Bayesian ridge performed poorly, while the other three classifiers, namely random forest, SVR, and neural network performed closely, with the heuristic classifier slightly behind them. Note that an accuracy of 49-50% here does not represent a random guess, as would be the case with a binary classification problem. In this context, given that the choices of the model are four, the success probability of a random guess would be 25%. In the following evaluations, we decided to use the random forest regressor as the ML-based method while using the heuristic method as a second proposed model. Random forest's regressor model is preferred over SVR for being more accurate and over NN for having a considerably shorter inference time.

Table 5.3: Comparison of regression and classification models' performance against the CloudHopper baseline and the proposed heuristic model across two different benchmarks.

Method	**Classifier's accuracy (%)**	**Downtime**		**Total time**	
		MSE	**R^2**	**MSE**	**R^2**
Heuristic	46	*n.n.*	*n.n.*	*n.n.*	*n.n.*
CloudHopper	24.3	*n.n.*	*n.n.*	*n.n.*	*n.n.*
R.Forest	49.2	11.3	0.80	1.13	0.95
SVR	47.8	15.0	0.73	1.33	0.94
NN	47.3	11.3	0.79	1.16	0.95
B. Ridge	23.5	23.1	0.59	5.55	0.76

Evaluation of ContMTD with Benchmarking Container Application

To conduct a generalized performance evaluation of ContMTD across diverse scenarios, we use the benchmarking container application described in Section 4.3.5 and test it on multiple microservice setups. Here, we consider a simple microservice-based web application with four components: back-end, front-end, database, and cache. For the sake of simplicity, we assume a single-logic application, enabling us to assign one container for each component. Even in this case, due to the exponential growth of possible combinations of resource load configurations (34 per container), exploring all combinations would be computationally prohibitive. Instead, we randomly sample 11 load configurations for each container in a four-containers microservice and generate an input space of $11^4 = 14641$ simulated scenarios.

For each scenario, we test three methods (CloudHopper, ContMTD with the heuristic classifier, and ContMTD with the RF classifier) across 20 trials and report the number of times each method prevails (*i.e.* obtains a lower LiMi downtime). As depicted in Figure 5.7, the heuristic based on statistical analysis outperforms the other methods by achieving the best results on 28.21% of generated scenarios, while the random forest regression model and the state-of-the-art method achieve the best results on 4.19% and 4.96% of the scenarios, respectively. However, in 62.65% of the scenarios, the heuristic method and the regression model result in a tie, both achieving the same results and outperforming the current state-of-the-art method. Moreover, the average downtime reached across

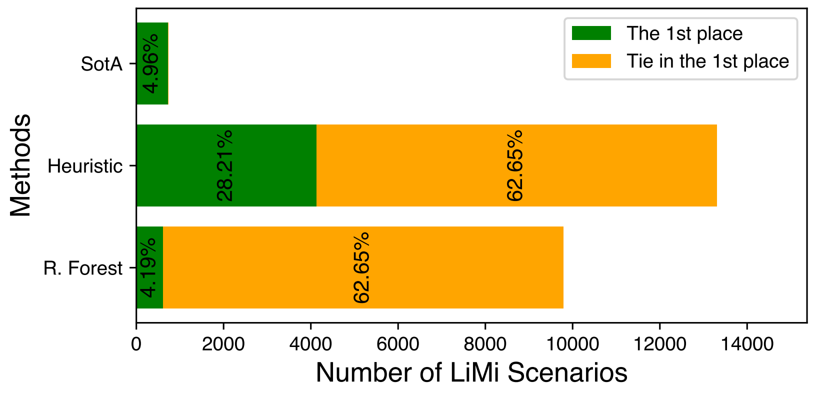


Figure 5.7: Evaluation of different migration techniques for randomly configured microservice applications.

Table 5.4: Evaluation of ContMTD against CloudHopper for the web microservices application.

Scenarios	**CloudHopper**		**ContMTD**		**Improve (%)**	
	Mean	**S.D.**	**Mean**	**S.D.**	**Mean**	**S.D.**
LiMi time	19.56	1.19	17.38	0.46	11.1%	60.86%
Downtime	12.16	15.23	7.15	4.00	41.17%	73.7%
Misalign	9.11	0.96	10.71	0.45	-17.5%	53.4%

scenarios with the heuristic method, the regression model, and the current state-of-the-art approach is 14.3, 14.88, and 24.85 seconds, respectively. This further information indicates that the heuristic method and the regression model achieve a downtime reduction of 42.45% and 40.12%, respectively, compared to the current state-of-the-art solution.

Evaluation of ContMTD with a Web Microservices Application

To investigate the applicability of ContMTD in a real-world microservices scenario, we deploy a WordPress microservices-based application consisting of three containers: a WordPress back-end (BE), a Nginx gateway (GW), and a MariaDB database (DB). The GW acts as the user interface, interacting with the end-users and relaying their API requests to the BE, while the BE instance queries the DB instance and provides a response for the corresponding request. Moreover, to generate network traffic, we leverage the Locust app to simulate end-user requests for our WordPress application, with a total frequency of up to 100 HTTP requests/s. Furthermore, we allow each container within the application to run a *pidstat* process, which monitors the resource consumption of the container, and set it to provide the average CPU, RAM, and disk usage in the last 5 seconds prior to the LiMi.

In this experiment, our objective is to evaluate how effectively ContMTD is able to migrate an entire multi-container application, aiming to obtain the following performance metrics: the service downtime during the migration, the total migration time, and the misalignment of parallel LiMi of containers. To clarify, the misalignment is defined as the time between the end of the first container migration and the end of the last container migration. The ideal case is to have all the application containers migrate simultaneously; hence, a lower misalignment means better results.

We perform 10 migrations each with ContMTD and CloudHopper, and we report the mean and the standard deviation of the metrics as per Table 5.4. The results obtained show that ContMTD, across 10 trials, significantly improves the service downtime compared to CloudHopper, with a reduction of 41.17%, from 12.16s to 7.15s. The standard deviation is greatly reduced by 73.7%, demonstrating the greater robustness of ContMTD performance. Similarly, ContMTD's total LiMi time also improved by 11.1% compared to CloudHopper. Its stability is further illustrated with the variance of this metric as well: whereas CloudHopper produces a standard deviation of 1.19s for the total LiMi time, ContMTD displays a smaller standard deviation of 0.46s, marking an improvement of 60%.

Although ContMTD displays superior performance with regard to the downtime and migration time, the results suggest that CloudHopper outperforms ContMTD in the migration scheduling part, reaching a lower misalignment value. A further investigation reveals that this discrepancy is due to the different approaches employed by ContMTD and CloudHopper in the scheduling process. While the former considers the resource usage of containers as the primary input, the latter focuses solely on the container size to estimate the migration time. However, the containers used in the WordPress application are identical based on the metrics monitored for resource consumption, all three displaying "low" levels of CPU, RAM, and disk consumption, even under a higher traffic load (circa 200 HTTP requests/s). In fact, only a very limited CPU spike to "medium" in the DB container is observed during our experiments, which is not enough for ContMTD to make sophisticated decisions. On the other hand, in terms of container size, the BE container is considerably bigger than the DB instance, which is significantly bigger than the GW container. Such distinct container sizes within the application allow CloudHopper to perform more effective scheduling, thus outperforming ContMTD in this specific scenario.

The experiments conducted on WordPress deployment provide tangible findings for our research. Nevertheless, we would like to emphasize that those tests explore only two container load configurations, with the CPU usage of the DB container fluctuating between "low" and "medium" depending on the user request rates. Since other metrics, such as RAM and disk usage, have a constant level throughout the experiments, the generalization of our results is severely limited. For this purpose,

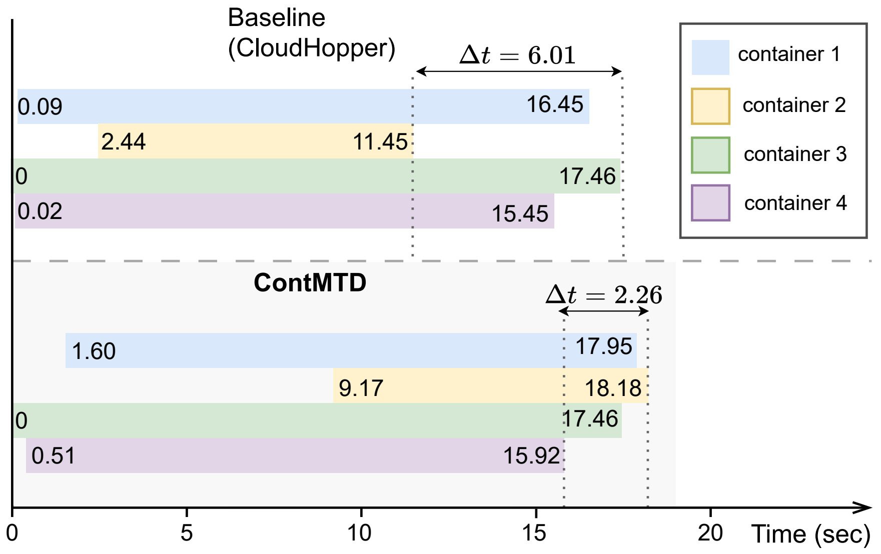


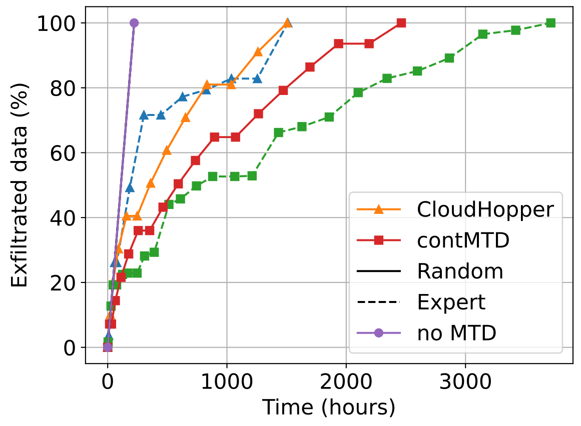
Figure 5.8: ContMTD vs CloudHopper in migration scheduling on a sample scenario.

we compare the scheduling performance of ContMTD and CloudHopper in different scenarios from Section 5.3.1. Figure 5.8 illustrates a sample scheduling process for both ContMTD and CloudHopper. Apparently, both methods migrate the four containers in the same order (*container 3*, *container 4*, *container 1*, and *container 2*), but differ in determining migration start times, particularly for the gateway container. Although ContMTD's total migration time slightly exceeds the optimal solution, it still outperforms the state-of-the-art in these scenarios.

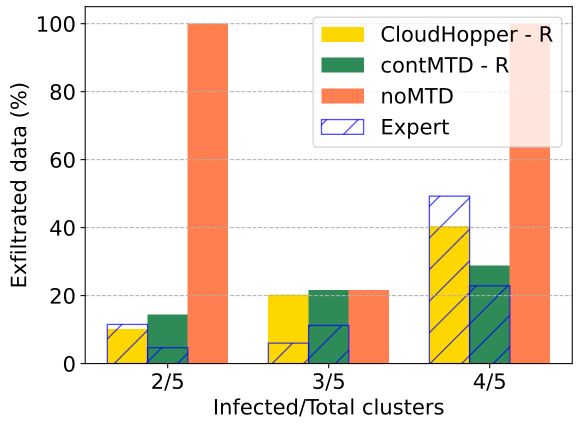
These results suggest that, in addition to container size, the container load configuration also has an impact on the LiMi performance. The improvement in this scenario over the WordPress case results from the regression model employed by ContMTD, which captures the strong correlation between container load configuration and the total migration time. Thus, ContMTD is able to predict the total migration time with a relatively low deviation, achieving an MSE of 1.13 seconds. Conversely, the lower performance displayed by ContMTD in the WordPress scenario indicates a lack of consideration for container size, which prevents the model from achieving better performance. This downside needs to be solved at the dataset creation level, to be addressed in future improvements of this work.

Data Exfiltration Attack

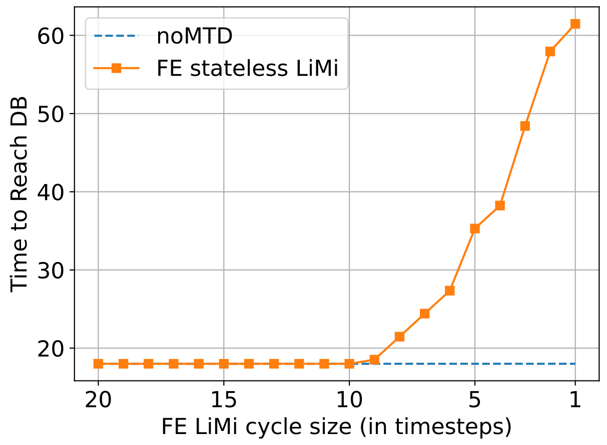
For this threat-based scenario, we assume a vulnerable multi-tenant cloud infrastructure enables malicious attackers to use co-hosted containers to exfiltrate data from neighboring containers. This threat



(a) Data exfiltration rate with one app and 4/5th of clusters infected.



(b) Exfiltrated data in the first 150 hours with one app.



(c) Necessary time for malware/attacker to reach the database container.

Figure 5.9: Changes in the data exfiltration rate with MTD.

is independent of the underlying vulnerability mechanism, which could target hardware-level flaws (*e.g.,* CPU cache side-channels and memory bus contention) or OS/kernel-level weaknesses (*e.g.,* container orchestration misconfigurations and privilege escalations). We assume using four clusters from different cloud providers, each cluster composed of five nodes. Single containers can be migrated independently between nodes of the same cluster and provider. However, migrating to other providers requires all microservices related to a single application to be migrated in parallel. We measure the amount of exfiltrated data when 25%, 50%, and 75% of the providers are vulnerable (respectively, one, two, and three providers out of four). We also adopt two assumptions strengthening the attackers' capabilities: *1.* attackers can resume the data exfiltration when an application is migrated back to an infected cluster, and *2.* attackers can aggregate exfiltrated data from different clusters for the same application. Finally, we consider a strict SLA with 99.95% of availability guaranteed, as provided by cloud providers such as Google and AWS [173]. If we allocate 50% of this downtime for LiMis, while the remaining downtime is used for eventual malfunctions and disruptions, it means that over the course of a month, a maximum of 10.95 minutes of downtime per application is tolerated.

When considering the ContMTD performance on the WordPress scenario presented in Table 5.4, this translates into a budget of 92 LiMis per application per month when using ContMTD with the heuristic classifier. The budget for CloudHopper's only pre-copy strategy is 54 application LiMis per month. We also assign a criticality value to each container running in the application scenarios, representing the value of the data exfiltrated: the database criticality is set to 3 (*i.e.* highly critical), the back-end and cache containers are set to 2, while the gateway is set to 1 (*i.e.* less critical). For the MTD strategy, we consider two different MTD policies: a random policy and an expert-knowledge policy. The random policy selects the microservices application, the new destination for migration, and the migration frequency randomly until the monthly LiMi budget expires. The expert-knowledge algorithm applies a simple greedy algorithm that migrates the microservices application with the highest criticality. It also migrates between clusters in a round-robin fashion to equate the level of exposure to all the providers. The time-frequency between migrations is set to be homogeneous throughout the month's duration.

Figure 5.9a presents an exposure to data leakage of a microservices application and the time required to exfiltrate all data from each container in the host, with an exfiltration rate of 1kbps (such as with *Prime+Probe* and *Flush+Reload* cache-based side-channels [174]). The figure shows a 6-fold increase in the time required to exfiltrate all app data when using MTD compared to when no MTD strategy is applied. In this measurement, the higher LiMi budget per month of contMTD vs. CloudHopper improves the exfiltration rate reduction by 10-fold compared to when no MTD strategy is applied. The best result is obtained when using the "expert knowledge" MTD policy with ContMTD, reaching a 16-fold reduction in the exfiltration rate. Nonetheless, we clarify that such improvements can vary extensively based on two considerations: *1.* When starting an application, there is a first probabilistic factor of whether the cluster is infected or not. When assuming the infections are static, applications without MTD deployed in secure clusters are protected from data exfiltration. On the other hand, with moving containers and the assumptions of attackers' capability earlier stated, all applications will eventually see all of their data exfiltrated. *2.* The ratio of infected clusters k over all clusters n is directly relevant to the previous consideration, as the higher k is, the lower the chances of an application being in a safe cluster. In fact, given a applications, the number of applications on infected clusters can be modeled as a binomial distribution: $X \sim \text{Binomial}(a, \frac{k}{n})$. Therefore, the expected number of exposed applications is the mean of the distribution X, *i.e.*, $\mathbb{E}[X] = a \cdot \frac{k}{n}$.

When introducing the migrations of MTD, the long-term average probability that an application is on an infected server remains the same, as observed in the results. Therefore, MTD only addresses the problem of continuous exposure to an infected cluster. Noting z a certain migration frequency over time t, the exposure duration of an application is $\textit{Exposure time} = \frac{k}{n} \cdot \frac{t}{z}$. This is shown in the short-term measurements taken at the beginning of the data exfiltration process, depicted in Figure 5.9b, where MTD can sensitively slow down the exfiltration rate in initial periods, critical to give more time for the attack detection and/or mitigation.

Stealthy Malware Infection

A preliminary evaluation demonstrates the benefits of incorporating stateless LiMis as a Moving Target Defense (MTD) mechanism to mitigate stealthy malware infection and propagation across interconnected containers. The same microservices application scenario, consisting of four containers, is used in this scenario. The gateway API container is configured as a stateless service and is set as the initial infection point, allowing subsequent propagation to the other containers. This setup aligns with the "statelessification" of exposed microservices described in Section 4.3.4. We define a threat model based on widely-used gateway, back-end, cache, and database technologies, each vulnerable due to using outdated versions. Each vulnerability is identified through the CVEs in the NIST NVD [151] database. The attack graph for the application is built using the Common Vulnerability Scoring System's (CVSS) [150] exploitability score to estimate the likelihood of container infection and to calculate the timesteps required for the attacker to compromise all containers. Figure 5.10 outlines the attack graph, where the starting state is denoted as *S*, and container states are labeled as *GW* (gateway), *BE* (backend), *CA* (cache), and *DB* (database). The graph edges represent attack vectors, with each vulnerability linked to its corresponding CVE and CVSS exploitability score.

The threat model accounts for the increasing ASP as the attacker gathers more information over time. Stateless LiMis of the *GW* container counteract this by removing the malware from the container at regular intervals, forcing the attacker to reinfect the *GW* and resetting the exploitation probability of attack vectors originating from the *GW* to their initial CVSS-based values. We formulate the time required for the attacker to compromise all containers without LiMi-based MTD. The probability p of infecting a container at timestep t is given by: $p(t) = p(0) \cdot (g)^t$, where g is the growth rate of the attacker's advantage over time and $p(0)$ represents the probability of the container being infected right after being deployed. The probability of **not** infecting a container at time t is $1 - p(t)$, and the

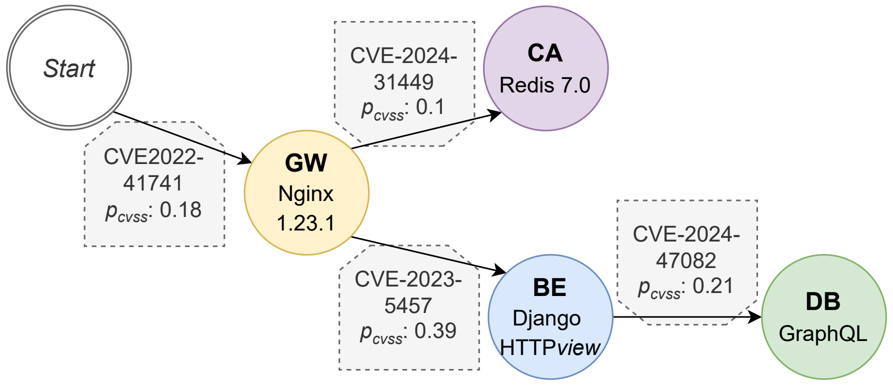

Start
CVE2022-41741 p_{cvss}: 0.18
GW Nginx 1.23.1
CVE-2024-31449 p_{cvss}: 0.1
CA Redis 7.0
CVE-2023-5457 p_{cvss}: 0.39
BE Django HTTPview
CVE-2024-47082 p_{cvss}: 0.21
DB GraphQL

Figure 5.10: The migration optimization workflow of ContMTD.

cumulative probability over time is the product of failed infection attempts at each prior timestep, followed by eventual success. Thus, the time of infection can be approximated by summing these probabilities up to 1,

$$1 - \prod_{i=0}^{t} (1 - p(0) \cdot (g)^{i}) = 1$$

which allows us to estimate the time for infection as:

$$T \leftarrow \arg_{t} \left(\prod_{i=0}^{t} (1 - p(0) \cdot (g)^{i}) \right) = 0$$

Now, the total time necessary to infect one container when no MTD is in place is formulated as the sum of the time required to exploit each vulnerability leading to it. For instance, the expected time to infect the database container is $T_{DB} = T_{S\rightarrow GW} + T_{GW\rightarrow BE} + T_{DB\rightarrow DB}$, by introducing MTD. When stateless LiMis are frequently applied to the *GW* container, the expected infection time approaches $\frac{1}{p}$, effectively neutralizing the growth rate g and significantly slowing infection. In addition, since the attacker continuously loses control of the *GW* container, progressing beyond it (*i.e.*, to *BE*) requires success in both $S \rightarrow GW$ and $GW \rightarrow BE$ approximately at the same time. Thus, with ContMTD enabled, the expected time to compromise the database becomes: $T_{DB} = T_{S\rightarrow GW} \cdot T_{GW\rightarrow BE} + T_{DB\rightarrow DB}$ The results in Figure 5.9c show that as the frequency of gateway LiMis increases, the time required for the malware to reach the database grows exponentially. Starting from a LiMi interval of eight timesteps, the required time more than doubles, and with a LiMi applied every timestep, the time increases by a factor of six. Given the very little overhead of stateless container LiMis compared to stateful LiMis, a timestep could be as low as 5 seconds, with a downtime of under one second [138].

5.3.2 Discussion and Limitations

In this section, we discuss the challenges faced and limitations of our framework and provide insight into how ContMTD could work on a more generalized setup with fewer assumptions.

Dataset characteristics - One of the most visible limitations stems from the way our dataset is generated. Due to the local cloud environment used for migrations, despite the similarity to real-life infrastructures, our measurements lack the variety of system loads that could be encountered in real-world cloud scenarios. Therefore, the actual time to complete a container migration in a production environment might deviate from our dataset, depending on various factors such as the hardware characteristics of the source and target nodes, the load on the physical or virtual machines that are external to the load of the containers themselves, and even the container runtime environment. To utilize ContMTD for a more realistic scenario and generalize its performance, a higher amount of environmental data is needed for the ML training.

Optimization model - Another limitation of ContMTD lies in the constructed optimization problem for the microservice application scenarios. In our study, we primarily focus on two objectives: 1) minimizing the downtime for each container migration by determining the most suitable method, and 2) scheduling the migration of containers within a microservice application such that the migration operations are finished approximately at the same time. Although these two goals do not have a conflicting nature, combining them does not necessarily guarantee an optimal migration on the application level. For instance, choosing the method with minimum downtime may actually increase the total migration time for that container, causing a delay in the migration of the entire application. We do not focus on this aspect during our study, since we only consider one independent microservice application in our scenarios. However, in a real-world setup, it is likely that there are some inter-application dependencies where the operation of one microservice relies on the existence of others. In such cases, ContMTD needs to be configured to focus on the required goal, total migration time, or other application-specific metrics for better optimization. The evident way to accomplish this is to adjust the method selection process by training the regression model (or using the heuristic model) on the targeted feature rather than on migration downtime values.

In relation to the previous limitation, ContMTD also has a limitation in its scheduling component. During the migration experiments, we simply assume that the containers are completely independent and ignore any possible co-dependency, which is usually not the case in a real-world application. Instead, components depend on each other for working properly, such as a back-end and database containers. In this case, simply arranging the migrations to end at the same time may not be the perfect solution in terms of total application downtime, and a more sophisticated approach needs to be considered. However, there are existing solutions for scheduling inter-dependent tasks, such as the critical path method used in software engineering projects [175], which could be adapted to the problem here. If the application has even more nuanced requirements, then we can treat it as a complex scheduling problem, and common solutions such as genetic algorithms, simulated annealing, or integer/linear programming can be integrated into the scheduling component of ContMTD [176].

5.4 Connection Handover of NFs using TopoFuzzer

This section presents the tests performed on TopoFuzzer and the results gathered for the evaluation. The analysis focuses on the QoS overhead caused by the traffic redirection in terms of bandwidth, latency, and packet loss rate. TopoFuzzer is deployed in the 5G testbed as a VM-based VNF with 8GB of RAM and 4 CPU cores. Symbols for the measurement metrics are listed in Table 5.5.

Table 5.5: Metrics and their symbols in TopoFuzzer experiments.

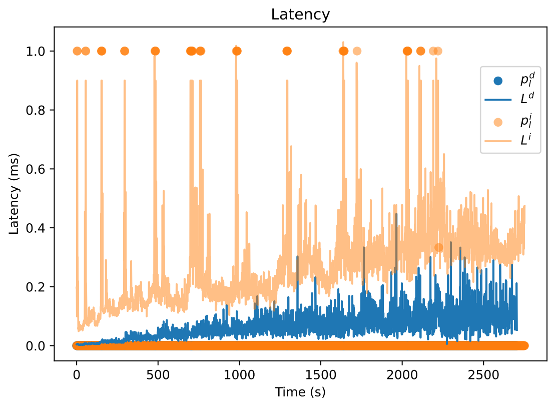
Symbol	Metric	Symbol	Metric
p_l	Packet loss rate	σ_{10s}	Std. dev. in a 10s window
L	Latency (RTT)	$CI_{95\%}$	95% confidence interval
x^d	x for direct conn.	m_r	MTD reinstantiation event
x^i	x for indirect conn.	m_m	MTD migration event

5.4.1 Experiments and Results

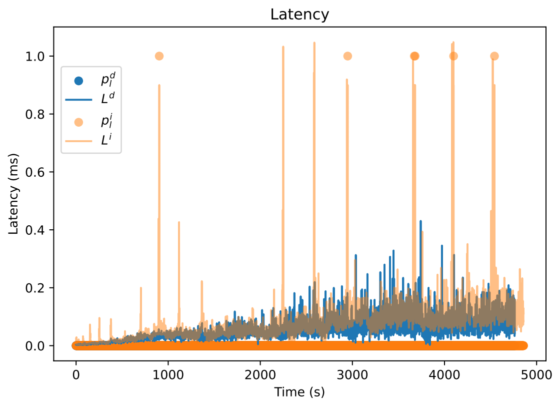
To evaluate the TopoFuzzer overhead on connection bandwidth, both TCP and UDP traffic are generated from a UE using `Iperf3`. As illustrated in Table 5.6, the average TCP bandwidth of direct connections is 159 Mbps for both sender and receiver. UDP traffic sees an average bandwidth of 225 Mbps, but is imbalanced with the full capacity for the sender and 138 Mbps for the receiver, which also has an increased jitter of 0.042 s on average.

When using TopoFuzzer, the standard deviation of measurements gets higher by 14.18% for TCP and 3.7% for UDP, indicating slightly reduced performance. The average TCP bandwidth capacity is measured at 157 Mbps for both sender and receiver, representing a negligible decrease of 1.25%.

The average UDP bandwidth capacity is instead 218 Mbps, also imbalanced due to the UPF performance, with 222 Mbps for the sender (1.33% decrease) and 136 Mbps for the receiver (1.45% decrease). Finally, the receiver's jitter is 0.0455 s, representing an increase of 8.33% compared to direct connections.



(a) Latency overhead and packet loss rate of TopoFuzzer when increasing UEs by ten every five seconds up to 100 UEs.



(b) Latency overhead and packet loss rate of TopoFuzzer when increasing UEs by five every five seconds up to 100 UEs.

Figure 5.11: HTTP/2 QoS overhead.

To conclude, TopoFuzzer did not show any relevant overhead on the bandwidth in the 5G testbed environment. However, what became evident during the tests is the importance of the allocated networking hardware resources and their capabilities. In our 5G testbed, the communication bottleneck was the UPF, which manages the data plane communication both at the core and edge domains (each domain with a different instance of the service).

Table 5.6: Average bandwidth for different traffic types.

Traffic type	**Avg. bandwidth Sender (Mbps)**	**Avg. bandwidth Receiver (Mbps)**	**Standard deviation**
Direct TCP	159	159	8.95
Direct UDP	225	138	37.85
Indirect TCP	157	157	10.22
Indirect UDP	221	136	39.26

TopoFuzzer QoS Overhead - HTTP/2 and HTTP/3

The first set of tests runs HTTP/2 communication traffic to evaluate the QoS overhead based on latency and packet loss rate metrics. The RTT is used to determine latency, while a request that does not receive a response within one second is counted as a packet loss. When establishing 30 connections with different UEs, adding a different UE every 5 seconds, TopoFuzzer overhead on the RTT is

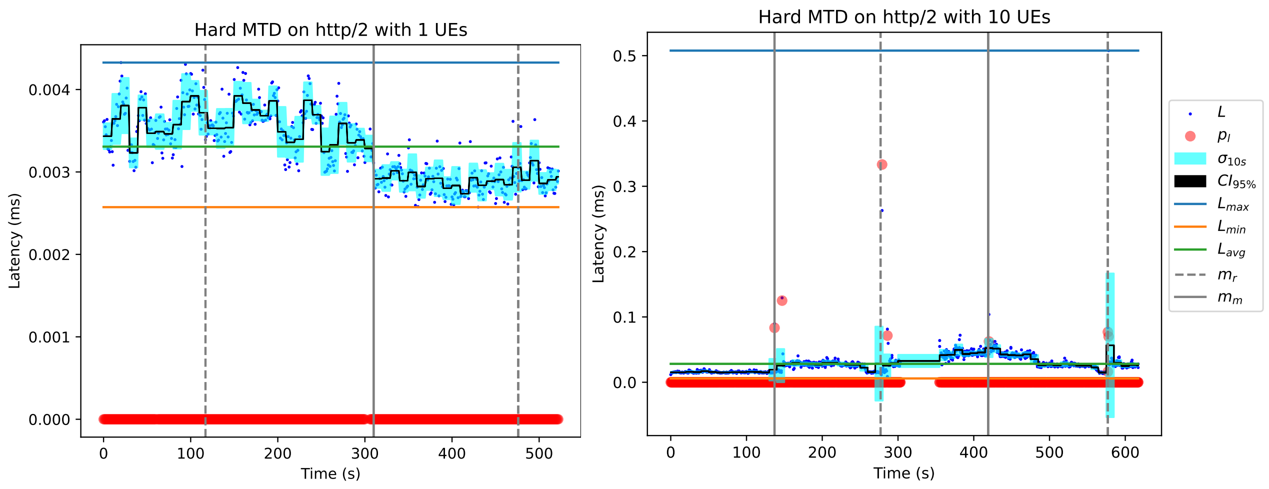
4.5%, while for 50 UEs, this increases to 29%, from an average RTT of 23.384 ms to 30.35 ms. In both cases, no packet is lost.

To test the scalability of TopoFuzzer, 100 connected UEs are deployed by adding 10 UEs simultaneously every 10 seconds. This leads to a rise in the RTT overhead, reaching 311% (test (a), depicted in Figure 5.11a). We assume that the TopoFuzzer overhead is affected mainly by the simultaneous connection requests rather than the total number of connected UEs, since packets are lost the moment ten new UEs establish their connections. This behavior may be caused by the *conntrack* functionality used in TopoFuzzer's proxies to map the connection request with the original destination port, modified beforehand with a port forwarding rule to allow redirecting traffic from all the ports. To test the hypothesis, we perform the same scalability test with up to 100 UEs, reducing the simultaneous connection requests to five. The hypothesis is confirmed, as the RTT overhead decreases to 38% for the same number of connected UEs (test (b), depicted in Figure 5.11b). Moreover, we observe a decrease in the number of packets lost, from 57 packets (in Figure 5.11a) to 6 packets (in Figure 5.11b). RTT values with and without TopoFuzzer are the same most of the time (represented by the overlap depicted in Figure 5.11b).

As TopoFuzzer can also redirect communications over the QUIC protocol (built over the UDP transport protocol), the QoS overhead is also tested on the recently standardized HTTP/3 protocol. The first noticeable difference compared to HTTP/2 is the reduction of the latency overhead, with 4% when having 50 UEs connected (against the previous 29% in HTTP/2) and 0.9% when having 70 UEs (compared to 37% in HTTP/2). With and without TopoFuzzer, the testbed could not scale to 100 HTTP/3 UEs, assuming that the cause is the additional TLS encryption the VNF has to support with its limited resources (1 vCPU and 512MB of RAM).

Another observable difference is the absence of the aforementioned limitation on the number of simultaneous connection establishments: instead of seven simultaneous connections with HTTP/2, the number of connections with HTTP/3 has to be over 48 to start observing packet losses. However, this limit is the same when running direct HTTP/3 traffic (*i.e.*, without TopoFuzzer in the middle), indicating that it might have reached the limits of the resources allocated for the VNF. Hence, the latency gap between direct and indirect communication does not increase, as observed in Figure 5.11a for HTTP/2.

To conclude, while it is already demonstrated in previous works [177] that QUIC generally outperforms TCP in QoS evaluations, this work also demonstrates a reduced complexity and lower overhead of redirecting HTTP/3 traffic compared to HTTP/2 for MTD. Moreover, it shows better man-



(a) Latency and packet loss rate during Hard MTD actions.

(b) Latency and packet loss rate during Hard MTD actions with 10 connected UEs.

Figure 5.12: QoS overhead of HTTP/2 traffic redirection.

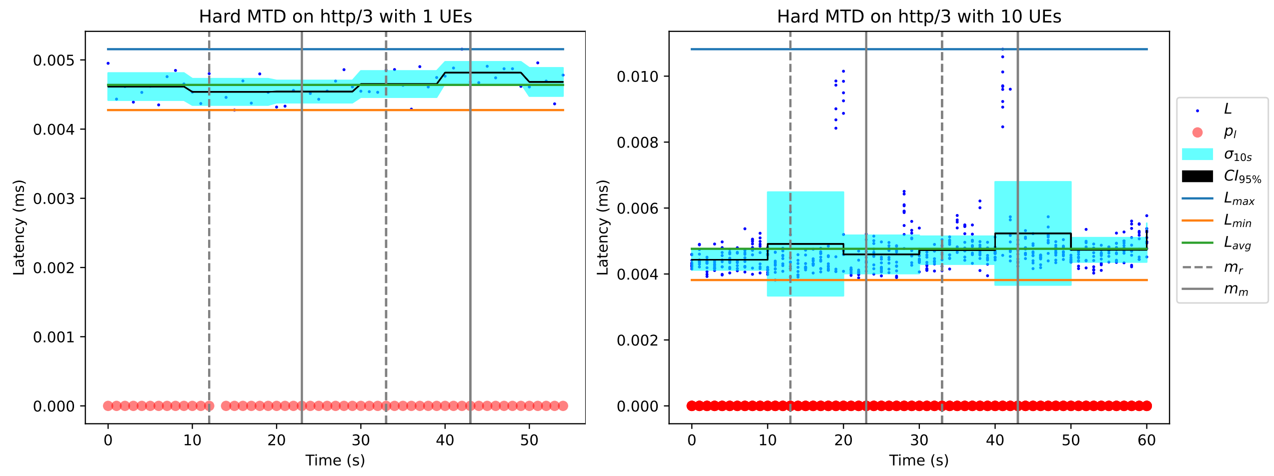
agement of instant surges in connections, motivated by the absence of tracking connections at the transport layer.

QoS Overhead for Re-instantiation and Migration of Live VNF: TCP and UDP

After measuring TopoFuzzer overhead with respect to direct connections, its redirection overhead is evaluated when performing re-instantiations and migrations of the edge VNF using MOTDEC. Tests alternate between re-instantiating and migrating operations on the VNF 30 times using HTTP/2 and 30 times using HTTP/3. Figure 5.12a shows the measured traffic of a VNF during a [re-instantiate – migrate – re-instantiate] sequence.

The measurements during the migration show the QoS differences between the core and the edge locations. The edge has better latency (second half of the measurements in Figure 5.12a) due to its proximity to the UE. Concerning the QoS overhead of *Hard MTD actions*, there is no packet loss or statistically noticeable latency and throughput overhead, indicating a smooth redirection of the UE traffic when moving to the new instance of the VNF.

Figure 5.12b shows the HTTP/2 traffic of 10 UEs connected to the moving VNF in a [re-instantiate – migrate - re-instantiate – migrate] sequence. Here, the QoS overhead is more observable. The average packet loss rate increase in a one-second frame window is 7% for the VNF re-instantiations and 33% for VNF migrations. Packet losses affect the latency (*i.e.*, the RTT), occasionally up to hundreds



(a) Latency and packet loss rate during Hard MTD actions.

(b) Latency and packet loss rate during Hard MTD actions with 10 connected UEs.

Figure 5.13: QoS overhead of HTTP/3 related traffic redirection.

of milliseconds in a one-second period. Considering the measured data, where a 33% packet loss rate in one second would correspond to a downtime of around 330 ms, if an SLA with 99.999% of availability has to be maintained, TopoFuzzer would allow up to 911 MTD migrations or 4293 MTD re-instantiations per service per year. This is equivalent to 17 migrations or 82 re-instantiations per week, a relatively good upper limit to proactively neutralize and kick off any infection from the VNF.

With HTTP/3, the traffic redirection of one UE does not show a clear overhead in the latency (see Figure 5.13a). As with HTTP/2, the traffic load of 10 UEs shows a latency overhead of migrations of 5 ms in a one-second window, a 100% increase, bringing final RTT values from 5 ms to 10 ms (see Figure 5.13b). However, unlike HTTP/2, there is no packet loss during the simultaneous redirection of the UEs' traffic, and this scales up to 50 simultaneous UEs. The downtime of 5 ms for HTTP/3 would theoretically allow to perform up to 60126 migrations per year (and 283'338 re-instantiations) under a 99.999% SLA availability requirement. However, this may decrease as packet losses start to occur once the simultaneous connections migrated are over 60 (and 8 for HTTP/2). Hence, to generally keep the values mentioned in terms of SLA requirements, TopoFuzzer could be further optimized to progressively migrate connections based on its vNIC capability and the protocol involved (TCP or UDP) rather than simply redirecting all existing sessions at once. This comes with an additional cost stemming from running two instances of the service for a longer time, which might be a negligible cost compared to having an occasional service downtime.

5.4.2 Discussion and Limitations

TopoFuzzer's usage of the *conntrack* feature, paired with *iptables* port forwarding to proxy all of the VNF's TCP traffic, comes with an additional overhead that is not present in UDP traffic (*i.e.*, QUIC and HTTP/3). TopoFuzzer could be improved in different aspects to reduce this overhead compared to traditional reverse proxies that target only one port of service, avoiding the initial port forwarding of the incoming traffic. The management of background sessions (between the `out-sockets` and the VNFs) can also be improved, *e.g.*, by connecting the new `out-socket` with the new VNF instance before closing the old `out-socket`, and then redirecting the data in the pipe to the new `out-socket`. This would reduce the service downtime by taking out from the blocking procedures the 3-way TCP handshake of the new `out-socket` with the new VNF instance.

Furthermore, the live migration of traffic encrypted at the network layer or the transport layer, such as IPSec and HTTPS, poses a trust condition as the TopoFuzzer proxy needs the TLS certificate of the VNF to authenticate itself with the UEs. In practice, this trust is already well established with cloud providers, as reverse proxy services are widely proposed by them and require such conditions [178]. With the advent of HTTP/3, such a trust condition might fall as the encryption in QUIC is purely handled at the application layer, and cloud operators do not need TLS certificates of services, as demonstrated in Section 5.4.1 with the HTTPS/3 tests. Alternatively, as TopoFuzzer is open-source, clients can require their private TopoFuzzer instance to configure it confidentially with their certificates.

5.5 MTD Strategy Optimization using OptSFC

This section evaluates the OptSFC solution, specifically examining its ability to enhance MTD strategies through the application of different deep-RL algorithms.

5.5.1 Digital Twin of the 5G Testbed for OptSFC and MTDFed ML Training

The convergence of a deep-RL model typically requires a substantial number of timesteps, in the order of millions per training [75]. On the real-time MOMDP model derived from the 5G testbed, proactive security operations are executed every 40 seconds (a parameter that can be adjusted for longer or shorter intervals). Training a single deep-RL on the MOMDP is done in one million timesteps, excluding the additional timesteps required for evaluation. This translates to roughly $(40s \times 10^6)/(3600s \times 24) \approx 463$ days of continuous training, just over 15 months. When considering the scope of this study, which involves training over 20 models between *OptSFC* and *MTDFed*, along with hundreds of evaluations (each model training includes at least 10 evaluations), the computational demands become prohibitive. Each evaluation alone consists of 33'000 timesteps, equivalent to 15 days of simulated time in the testbed. Consequently, the total time required for training and evaluation quickly escalates to the order of hundreds of years.

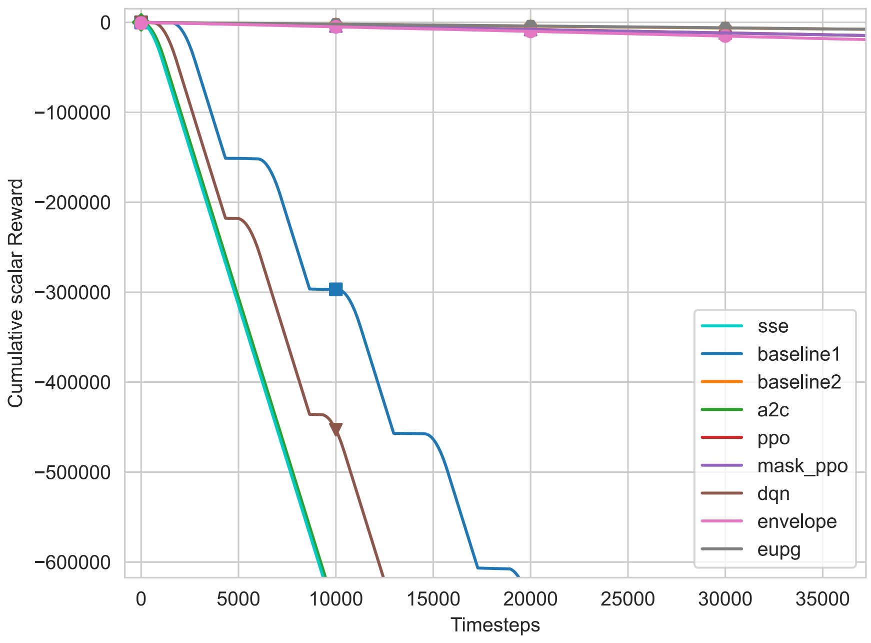
This scalability challenge is a well-known limitation of deep-RL, particularly in domains requiring to learn "physical" operations, such as robotics. A common approach to mitigate this issue is to train the RL model in a simulated environment and subsequently transfer or test the learned model in the real world. To address this, we developed a simulated environment based on statistical measurements obtained from the real 5G testbed. These measurements include network metrics for UEs' traffic, the incremental impact of additional UEs, the QoS overhead of MTD actions, their enforcement times, and risk assessment data for deployed VNFs. The digital twin generates metric values using a Gaussian distribution, parameterized by the mean and standard deviation of the measurements recorded in the 5G testbed, bounded by their respective minimum and maximum values.

Furthermore, deploying an untrained model in the 5G testbed would result in random, potentially costly decisions during the initial phase of training. To avoid this, we first train the model in the simulated environment to achieve a sub-optimal but reasonable performance level. The model then undergoes online training in the real testbed to adapt to the dynamic changes in network topology and composition over time.

5.5.2 Experiments and Results

A comparative study is done by training the deep-RL algorithms for one million timesteps each and comparing their model's performance with a random MTD policy (*baseline1*) and a static MTD policy (*baseline2*). *Baseline1* is a simple policy that randomly selects an MTD action from the possible ones, *i.e.*, actions that target an existing VNF, as the MOMDP is a static model and is oversized to the maximum number of possible simultaneous VNFs, multiple times bigger than the actual number of VNFs running in the 5G testbed. In contrast, *Baseline2* selects actions for the running VNFs based on the security priority level attributed to each of them. Next, it decides on the MTD action (*i.e.,* migration or re-instantiation) based on the availability values in SLA defined in the previous measurements (Section 5.2.2). *Baseline2* also runs one MTD action at a time, while the general MOMDP model allows non-conflicting MTD actions moving different VNFs simultaneously.

The performance of the models is also compared with a Stackelberg Markov Game solver from the MTD framework by Sengupta et al. using the Strong Stackelberg Equilibrium (SSE) [114]. The SSE solver has to calculate the full Q-value matrix and the reward values for all (state, action) pairs, imposing more rigorous constraints on the size limits of the MDP, in contrast to its utilization with



(a) Continuous management performance of the deep-RL models.

RL model	$\bar{x}$ rew/step	RL model	$\bar{x}$ rew/step
SSE	-29	**Baseline1**	-0.44298
Baseline2	-0.2123	DQN	-0.647
A2C	-29	PPO	-0.39
MaskPPO	-0.39	EnvelopeMORL	**-0.2114**
EUPG	**-0.1932**		

(b) Average ($\bar{x}$) reward per step

Figure 5.14: Cumulative reward and reward/step.

a deep-RL agent. To mitigate this problem in our experimental setup, we simplified the MDP model for the SSE solver and discretized continuous metrics to values such as 'low', 'average', or 'high' to avoid a state-space explosion due to high-range values.

The performance of the deep-RL agents tested in OptSFC is depicted in Figure 5.14a. The y-axis represents the average reward $\overline{\mathcal{R}}$ obtained in a simulated episode, while the x-axis is the number of episodes n used for the training phase.

The results from the benchmark clearly show a better optimization of the MOMDP return on rewards when using the multi-objective RL algorithms compared to single-objective models trained on the scalarization of the reward vector $\overline{R}$. In fact, only the Envelope MORL and EUPG algorithms learned a slightly better policy than *baseline2*. SSE and A2C appear to get stuck in a local minimum, avoiding performing any MTD action to circumvent the penalties of selecting an invalid action, such as a non-existing VNF or a VNF that is already undergoing a prior MTD action. The DQN policy learned that selecting the same MTD action is better than doing nothing, as it resets the ASP values of the given VNF's threats. However, since other VNFs are left with their threats increasing the ASP of possible attackers with time, the cumulative reward of the proactive MTD management drops exponentially as visualized in Figure 5.14. Both PPO and Maskable PPO models learned to re-instantiate the VNFs defined as most critical, leading to an average reward per step of -0.39, greatly improving over *baseline1* and improving the cumulative reward to a linear curve. One of the reasons is that once the availability constraint of 99.999% is reached, the VNF cannot be moved. *Baseline2* still performs better as it considers the SLA availability constraint of 99.999% and follows a more evenly distributed selection of VNFs, reaching this constraint less frequently. Envelope MORL's policy demonstrates similar performance, moving critical VNFs more often and also moving all other VNFs evenly and with lower frequency. Finally, we identify the improvement of the EUPG model over Envelope MORL by its tendency to select re-instantiations over migrations, as the former has less network overhead than the latter. The final EUPG model achieves an average reward per step of -0.193, allowing the accumulation of less than 20'000 penalty values through proactive management of 100'000 timesteps.

5.5.3 Discussion and Limitations

One limitation of the OptSFC solution is that MOMDP models are defined with a static number of states, and as networks dynamically change in size and shape, MOMDPs are not inherently adapted for network modeling. We address this issue in *MERLINS* by defining a number of MOMDP states equivalent to the maximal dimensions of the network infrastructure. This consequently leads to the action space of the MOMDP being dynamic: resources in the MDP that are yet non-existent in the actual network are considered idle and inaccessible states and actions for the deep-RL agent. This solution shifts the problem to the action space, which becomes dynamic, while in MOMDPs it should also be static.

The action space shift is realized both by the maskable actions mechanisms of the MaskablePPO algorithm as well as with penalties provided if the action is invalid, in an effort for the models to learn which actions are valid at a specific state. However, MaskablePPO did not yield a significant improvement over the conventional PPO algorithm, while fixing the MOMDP to an oversized network had affected the learning curve for smaller networks. For this reason, we limited the tests of OptSFC to a specific testbed size. While the used 5G testbed can approximate the scale of a private industrial 5G/B5G network, public operators handling a huge number of VNFs face a more complex challenge.

This complexity exacerbates the MORL learning process in the context of real-world public 5G/B5G networks. A prospective remedy to this limitation involves the learning of a solution set consisting of multiple models learned on different network sizes, rather than training a generalized *one-for-all sizes* policy.

Finally, OptSFC's deep-RL model can be improved by transferring the learning phases from the digital-twin environment to the real 5G testbed once the model's performance is deemed sufficient for its deployment. This fine-tuning process on what is then defined as a pre-trained model allows us to have better adaptability to the real dynamics of the network. The models' training should also be a continuous process, *i.e.,* performed all along the deployment of MERLINS, as the network can change its dynamics, deploying VNFs and services with different requirements and different SLA constraints that affect the deep-RL model performances.

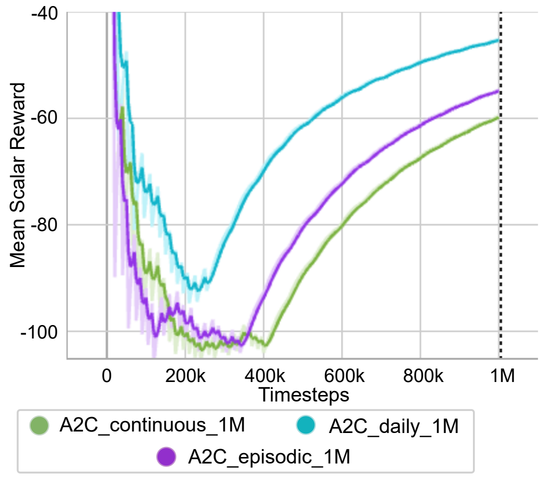
5.6 Multi-Tenant Federated MTD Strategies using MTDFed

This section presents an evaluation of MTDFed, focusing on two key aspects. Firstly, it analyzes how refinements to the MOMDP model, such as feature reduction and reward function optimization, contribute to the enhancement of MTD strategies. Secondly, it investigates the overhead associated with the FL-based training process compared to a single, centralized deep-RL training approach. This analysis specifically examines the impact of FL on both convergence time and the performance of the trained models.

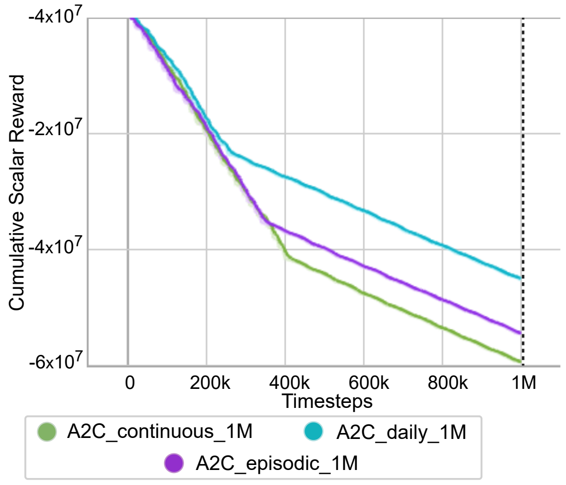
5.6.1 Experiments and Results

MTDFed evaluation employs a comparative analysis against *baseline1* and *baseline2*, following a similar methodology to OptSFC's evaluation. However, the SSE and DQN models are excluded from this study due to their subpar performance and inadaptability of their implementation with the MOMDP's nature (not a figure and a much bigger exploration surface than what SSE could cover). As a result, the evaluation focuses on four models: A2C, PPO, MaskablePPO EUPG, and Envelope MORL.

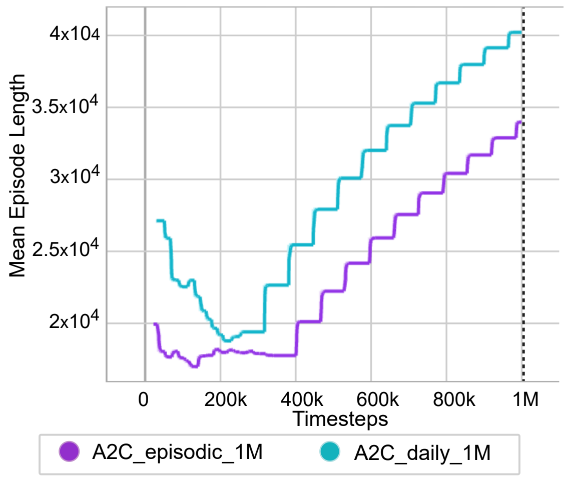
Each model undergoes training for one million timesteps. The training is conducted across three variations of the MOMDP model: continuous task, episodic, and daily. These models are then evaluated over 33'000 timesteps, equivalent to one month of deployment, with each timestep representing a 40-second interval. Similar to OptSFC, as these training times would be prohibitive in the actual 5G testbed, a digital twin is used to simulate the 5G testbed setup, running the same four VNFs. The digital twin is further extended with the simulation of three stateful CNFs using the measurements taken with ContMTD, one from the *nginx* load balancer, and two from the statistical aggregation of over 6'000 LiMis recorded in ContMTD's dataset.



(a) A2C mean scalar reward per step during training for the three MOMDP variants.



(b) A2C cumulative scalar reward during training for the three variants.



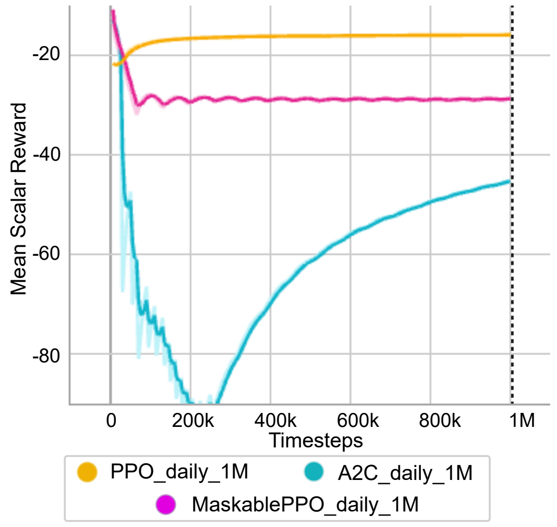
(c) A2C mean episode length during training for the episodic and daily variants.

Figure 5.15: A2C reward measurements during training for the three MOMDP variants.

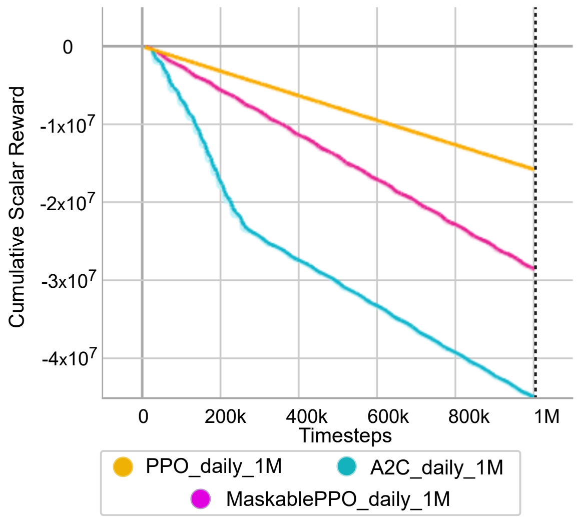
Single-Objective Models

In the single-objective deep-RL experiments, agents learned to extend episode lengths, as shown in Figure 5.15c, effectively preventing rapid budget depletion and prolonged inactivity. This strategy directly impacts security performance. Specifically, daily budget allocation consistently outperformed monthly and weekly allocations. This is evident in Figures 5.15a and 5.15b for the A2C algorithm, and similar trends were observed across PPO, MaskablePPO, and multi-objective models. The improved performance with daily budgets stems from the agents' ability to distribute MTD actions more evenly, mitigating the risk of overconsumption and subsequent inactivity. Prolonged inactivity periods, during which the ASP increases, resulting in higher accumulated penalties on the security objective.

Furthermore, Figure 5.16 demonstrates that the PPO algorithm exhibited a faster learning rate and converged to a more optimal solution compared to A2C and MaskablePPO. This is corroborated by the cumulative scalar reward achieved by each algorithm. PPO displayed a less steep negative slope, indicating a more stable and effective learning process. At 1 million timesteps, PPO achieved a cumulative reward of -15 million, while MaskablePPO and A2C reached -28 million and -44 million, respectively. The following subsection further discusses the strategies learned, comparing the results to those of the MORL algorithms.



(a) Mean scalar reward per step during training for the three mono-objective deep-RL models in the daily MOMDP variant.



(b) Cumulative scalar reward per step during training for the three mono-objective deep-RL models in the daily MOMDP variant.

Figure 5.16: Reward measurements of the three mono-objective deep-RL algorithms during training in the daily MOMDP variant.

Table 5.7: Models' average reward and action distribution in the test environment

Type	Mean reward	Strategy
Baseline1	-16.37	Randomly select among available actions. If no action is available, do nothing.
Baseline2	-25.59	Select an available action when no action is in progress. The selection is 50% doing nothing, and 50% selecting an action based on VNFs' criticality values defined in the MOMDP.
A2C 1M	-42.65	Always same action: reinstantiation of VNF3.
Maskable PPO 1M	-39.12	Almost always the same action: reinstantiation of VNF3 and a few times migrations of VNF3.
PPO 1M	-33.15	Always do nothing.
EUPG 1M	**-14.75**	Do nothing: 82%; VNF LiMi: 3.2%, VNF reinst.: 12.8% (per VNF); CNF LiMi: 2.1%.
ENVELOPE 1M	-20.36	Do nothing: 50%; VNF1's LiMi and reinst: 0%; VNFs 2 to 4: LiMi 49.62%, VNF reinst.: 0.17%; CNFs LiMi: 0.3%.

Multi-objective Models and Collective Results

The evaluation revealed that models utilizing the daily MOMDP consistently outperformed those using the episodic MOMDP, mirroring the trends observed with mono-objective agents. This advantage stems from the same principles of promoting a more consistent MTD action distribution and avoiding prolonged inactivity. Table 5.7 summarizes the results for the daily MOMDP variant, which demonstrated the best overall performance. Specifically, the A2C model converged to a strategy of repeatedly reinstantiating a single VNF, selecting the most vulnerable as identified by OptSFC's RI.AS. module (based on the vulnerabilities found with the OpenVAS scanner). This strategy, however, proved largely ineffective, yielding results worse than the near-inaction strategy adopted by the PPO model at 300,000 timesteps. It is worth noting that inaction performs better because selecting a non-enforceable MTD action results in a negative reward. This is designed in the MOMDP as an attempt to teach the agent which MTD actions are feasible.

PPO and MaskablePPO models exhibited a comparable strategy to focus on the most or second-most vulnerable VNF, resulting in a slightly better (but sub-optimal) performance. In contrast, MORL algorithms generated more diverse strategies. Notably, the EUPG model surpassed both the 'static' and 'random valid' baselines, similar to the findings of OptSFC. Table 5.7 details the Envelope model's strategy, which involved 50% inactivity and 50% equal migration of three VNFs, with minimal CNF migration (approximately 1%). The EUPG agent, however, demonstrated a broader range of MTD actions, migrating and reinstantiating all VNFs and containers. It prioritized stateless VNFs due to their lower resource overhead and enhanced security benefits against malware and backdoors. The EUPG model's action distribution was as follows: VNF reinstantiation at 0.8% each (3.2% total), VNF migration at 3.2% each (12.8% total), CNF LiMi at 0.7% each (2.1% total), and inactivity at 82% of timesteps."

MTDFed Performance

This section evaluates the performance of FL-based models trained across three VNOs, each operating within its own environment based on an instance of the testbed simulation. To ensure a fair and direct comparison between the FL-based models and single-objective models, all three VNO environments were configured to run the same set of VNFs and CNFs. The evaluation focuses on the following key metrics:

- Training overhead: how fast or slow is the training in the FL setup compared to a single VNO training the model on its own?
- Model performance: for the same effort provided by one VNO, how much better can the models get in the FL setup?
- Secure aggregation overhead: how much longer does the aggregation take when using the SMC-based secure aggregation?

To compare the convergence results, we first identified the convergence time of the mono-objective agents. Initial convergence measurements were performed by evaluating the models during training at intervals of 100'000 timesteps. These results revealed that most models reached their optimal performance within the first 100'000 timesteps, with the exception of the Envelope model. Thus, to obtain more granular insights, we train the models and evaluate them at shorter intervals of 5'000 timesteps, up to a total training duration of 100'000 timesteps per model. Additionally, model evaluations were conducted for 5'000 timesteps each instead of 65'000 in order to accelerate the overall experiment, since every model would be evaluated twenty times (*i.e.* the number of cycles within the training period).

As illustrated in Figure 5.17, the convergence of PPO and MaskablePPO occurs at 30,000 and 20,000 timesteps, respectively. In contrast, the FL versions of these models converge within the first cycle of 5,000 timesteps, although MaskablePPO saw a temporary deviation from the convergence

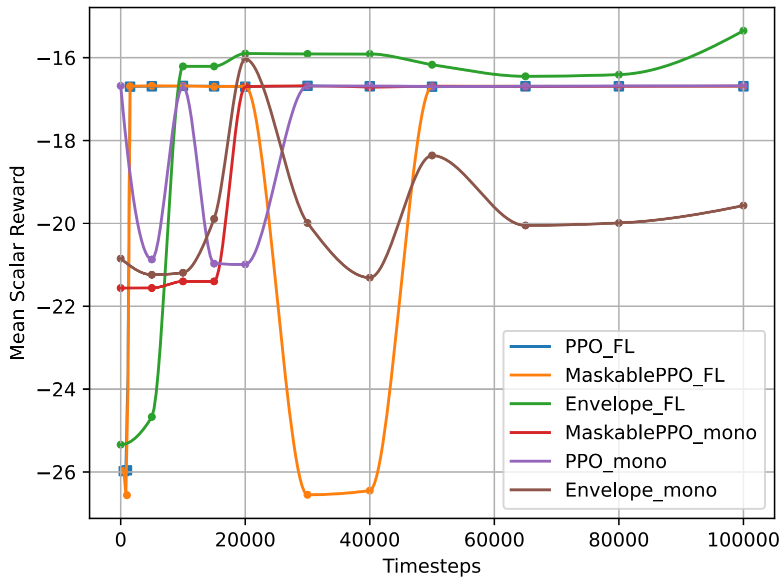


Figure 5.17: Cumulative scalar reward of different models within 100'000 timesteps of training.

at 30'000 timesteps, returning to the optimal value soon after. This represents a significant improvement in convergence time, with reductions of at least 83% and 75% for PPO and MaskablePPO, respectively (i.e., a sixfold and fourfold decrease in time). These results demonstrate that the FL-based models achieve superior performance compared to simply parallelizing the deep-RL training across three processes. The EUPG and A2C algorithms were omitted from the results due to their lack of convergence. Notably, EUPG exhibited the best performance from the beginning of training, with no observable convergence. This behavior can be attributed to the design of the EUPG algorithm, which includes an evaluation of the model every 1,000 timesteps for policy gradient estimation outside of the training steps. The assumption is that the policy selected during the first evaluation remains unchanged throughout the training, suggesting that it may represent a local optimum or the global optimum solution.

For all the models, after the convergence, the final performance of the models remains the same, except for the Envelope model, which sees an improvement of 20% after 100'000 timesteps, from a final average reward per step of -16.69 in the FL version to -16.69 in the FL version. The non-FL Envelope model actually struggles to converge within the tested timesteps and previous trainings to 1M timesteps showed a convergence after 300'000 timesteps.

FL Secure Aggregation Overhead

To quantify the overhead of the SMC-based secure aggregation, a set of FL aggregations is conducted between the three VNOs for 10 rounds. Aggregation times were measured for each trained deep-RL model, both with and without secure aggregation. As detailed in Table 5.8, secure aggregation resulted in an average aggregation time increase of 135%, rising from 7.09 seconds to 16.72 seconds. Notably, the overhead varied across models, with the Envelope model exhibiting the highest increase. This can be attributed to the Envelope model's larger neural network architecture, which necessitates the aggregation of a greater number of weights and biases per round. Consequently, the performance of secure aggregation is influenced by factors including model size, the number of FL participants, and the number of intermediate SMC nodes involved in summing perturbed values before global model aggregation. In this evaluation, we maintained a constant number of three VNOs, reflecting a realistic scenario where the telecommunication operator would limit the number of VNOs accessing their infrastructure. Therefore, the impact of varying participant numbers is not relevant here, but previous research has demonstrated that this is a factor increasing the SMC complexity by $\mathcal{O}(n \log(n))$ [163].

5.6.2 Discussion and Limitations

The results presented in this work demonstrate the feasibility of federating deep-RL training, which not only accelerates convergence but also holds the potential for performance improvements, even though the likelihood of reaching the same optimal policy remains high. While these findings are promising, they are preliminary, and further evaluations are necessary to determine if alternative variants of the MOMDP model could enhance training and enable the discovery of superior policies. The primary objective achieved by MTDFed is to demonstrate the feasibility of enhancing MTD strategies securely, without compromising sensitive information, through intelligence sharing with poten-

Table 5.8: Aggregation Time and Time Difference

Algorithm	Mean aggregation time per round (s)		Increase rate (%)
	without SMC	with SMC	
PPO	6.96	16.33	134.4
MaskablePPO	7.26	16.03	120.81
A2C	7.25	17.25	137.63
EUPG	6.94	16.29	134.75
Envelope	7.02	17.71	152.23
Overall Mean	**7.09**	**16.72**	**135.88**

tial business competitors (in the context of VNOs). However, several challenges and limitations have been identified, which warrant further investigation.

MTDFed retains OptSFC's centralized decision-making architecture, where a central core system orchestrates operations across all VNO domains, including core and edge domains. However, an alternative decentralized architecture, deploying local models at each edge domain, could be considered. This approach, especially when combined with online model training during inference, would necessitate a two-phase model aggregation in the MTDFed framework:

1. Local Aggregation: Each VNO would aggregate its edge models into a single core model.
2. Global Aggregation: The core models from each VNO would then be aggregated into a global model.

This decentralized structure would considerably reduce the volume of data exchanged between edge nodes and the core, as OptSFC decisions would be made at the edge level. Consequently, the primary data exchange between edge and core would be limited to the secure aggregation of the local aggregation phase. Despite this potential advantage, further decentralizing the OptSFC decision system introduces significant challenges. Informed MTD decisions require a comprehensive network view, necessitating centralized observation from the core. Moreover, independent edge-level OptSFC systems within the same network could lead to inefficient or conflicting decisions, complicating implementation and management. Therefore, to avoid this complexity, the MTDFed solution in this thesis adopts a federated approach with only one global aggregation.

Finally, while FL with SMC's secure aggregation guarantees model confidentiality and mitigates malicious but passively curious participants, even in a centralized setup, it does not guarantee security against active malicious participant who performs poisoning attacks such as adversarial samples, label flipping, or min-max attacks. Thus, MTDFed can be further secured by implementing additional security methods to increase the robustness of FL models against such malicious participants without overlooking privacy preservation.

5.7 Lessons Learned from the Evaluations

The development and evaluation of the solutions presented in this thesis —MOTDEC, ContMTD, TopoFuzzer, OptSFC, and MTDFed— have provided valuable insights into the challenges and opportunities in securing modern Telco Cloud 5G/B5G network infrastructures. Below, we summarize the key lessons learned from each solution:

Lesson 1. SDN-based soft MTD actions enhance security but introduce network overhead. MOTDEC demonstrates that MTD strategies, particularly soft MTD actions like IPv6 address and port shuffling, can effectively mitigate threats in edge computing by dynamically altering the attack surface. These techniques significantly increase the difficulty for attackers to identify and target services, as shuffling limits the time window in which attackers can locate and exploit vulnerabilities. However, this security enhancement comes at a cost: the computational overhead introduced by L4 header modifications (e.g., port shuffling) can degrade network performance, especially in heavily loaded environments. For instance, the evaluation revealed that port shuffling has a negative impact on the quality of the DASH video stream and increases latency under high network loads. To address this, future work could explore alternative implementations, such as TopoFuzzer's 2-socket proxy method as opposed to using SDN, to avoid the mangling of all passing packets, reducing the performance impact of soft MTD actions.

Lesson 2. Hard MTD actions are effective for stateless services but require broader adoption. Hard MTD actions, such as stateless VNF re-instantiation and migration, are highly effective in neutralizing malware infections and backdoors in stateless VNFs. The MOTDEC evaluation demonstrated this by successfully mitigating a tampering attack in the 5G testbed, with only minimal service downtime measured during the process. These actions are particularly valuable in scenarios where detection technologies are ineffective or absent, as they are triggered proactively and disrupt undetected threats. While ContMTD focuses on enabling stateful LiMi for containers, there is a pressing need to democratize stateless LiMi as a proactive MTD strategy in cloud-native environments as well. Stateless microservices, which are designed for replication and distribution across multiple servers and clusters, are inherently suited for such MTD actions. Integrating proactive re-instantiation and migration as standard features in platforms like Kubernetes and Docker could significantly enhance cybersecurity defenses in modern cloud environments.

Lesson 3. ML and heuristic models optimize stateful LiMi strategies in cloud-native environments. ContMTD's use of ML models to optimize LiMi strategies for containers and microservices significantly reduces downtime compared to state-of-the-art methods like CloudHopper. Interestingly, the heuristic model often outperforms ML models, highlighting the value of combining statistical analysis with ML for robust and interpretable solutions, as ContMTD does with the heuristic model for classification and RF for the regression of migration time. ContMTD's evaluations also show that LiMi can slow down attacks such as data exfiltration and malware propagation, but the effectiveness of these strategies depends on the frequency of migrations and the attacker's ability to reinfect containers.

Lesson 4. The protocol choice impacts MTD performance, with HTTP/3 outperforming HTTP/2. TopoFuzzer effectively manages connection handovers during MTD operations, but its performance varies significantly depending on the protocol used. HTTP/3 (using QUIC) outperforms HTTP/2 (using TCP) in terms of latency and packet loss, particularly during simultaneous connection redirections. This is because QUIC, being a UDP-based protocol, does not require connection tracking at the networking layer (*i.e.*, layer 3 in the OSI model), unlike TCP-based HTTP/2, which relies on mechanisms like *conntrack* and introduces additional overhead. For example, TopoFuzzer's evaluation showed that HTTP/3 connections experienced minimal latency overhead and no packet loss during migrations, even with multiple simultaneous connections. This underscores the importance of protocol selection in MTD strategies, especially in environments with high connection churn and requirements, such as 5G.

Lesson 5. MORL algorithms outperform single-objective models in MTD optimization. MORL algorithms, such as EUPG and Envelope MORL, outperform single-objective models in optimizing MTD strategies. These algorithms effectively balance competing objectives like security, performance, and availability, leading to more robust and adaptive MTD policies. For example, EUPG demonstrated superior performance by prioritizing stateless VNFs and minimizing downtime during migrations. Despite their advantages, MORL algorithms are relatively underutilized compared to single-objective models. An important challenge faced in the evaluation of OptSFC and MTDFed is the dynamism of the network and the lack of deep-RL agents to operate in an environment with changing action spaces and changing state spaces. Thus, more research is needed on how to generalize ML models or how to create a set of models to adapt cognitive decision-making components like OptSFC to dynamic and diverse environments, such as Telco Cloud networks.

Lesson 6. Data engineering is critical for MTD strategy optimization. In the OptSFC solution, the significance of data engineering during the MTD optimization process cannot be overstated. Given the diverse range of objectives under consideration for the optimization of the MTD strategy, a comprehensive aggregation of real-time data sourced from the network's monitoring infrastructure is necessary. Next, to harness the value of such data, it becomes imperative to establish a well-designed reward system. This system translates the collected data into quantifiable metrics that reflect the network's status toward the objectives outlined within a MOMDP or any other objective modeling. However, training ML models on monitoring data from public networks carries inherent risks, such as exposing sensitive end-user information and compromising privacy. Therefore, careful consideration must be given to data anonymization and secure aggregation techniques to protect sensitive information while enabling effective MTD optimization.

Lesson 7. FL enables collaborative MTD optimization but requires robustness. MTDFed enables secure and collaborative optimization of MTD strategies across multiple tenants without sharing sensitive data. FL-based models converge faster and can achieve better performance compared to single-tenant models, particularly in scenarios with shared infrastructure. The choice between centralized and decentralized architectures depends on the specific requirements of the network. Future work could explore hybrid approaches that combine the benefits of both architectures, such as local decision-making at the edge with global coordination at the core. Additionally, novel techniques to improve the robustness and fairness of FL in multi-tenant environments should be investigated.

6
Summary, Conclusions, and Future Research

Along with the advancements in B5G networks, the next generation of wireless communications, envisaged as 6G or NextG, will integrate novel technologies and techniques such as Reconfigurable Intelligent Surfaces (RIS), native AI, Internet of Everything (IoE), and ubiquitous cloudification [3]. It will also provide higher flexibility and more granular control of digital infrastructure and services compared to previous generations. Adopting cloud-native computing comes with considerable benefits as well as a greater risk of security vulnerabilities and impactful threats, already emerging as a pressing issue with current 5G developments [30].

Additionally, networks are expanding to a more distributed infrastructure using resources across edge to cloud domains. While this architectural setting is expected to significantly reduce communication overload, the larger infrastructure that is eventually created also presents new challenges as it increases the network's attack surface. Consequently, managing and securing the services along the edge-to-cloud continuum becomes increasingly critical.

This PhD thesis addresses these security challenges by proposing a novel cognitive MTD approach, *MERLINS.MERLINS* enhances both proactive and reactive security by leveraging advancements in AI/ML, virtualization in edge-to-cloud environments, and automated service orchestration. It developed provides: *(i)* a modular and closed-loop methodology enabling optimized and dynamic control of MTD operations following a closed-loop paradigm of *observing, orienting, planning, deciding, acting,* and *learning* tasks, while targetting security gains, reduced operational costs, and contained QoS/QoE overhead *(ii)* a framework and an HLA constructed that incorporates all phases of the methodology into layers and components and *(iii)* a set of solutions representing a first implemented instance of the *MERLINS* approach that covers all the layers designed in the HLA.

6.1 Summary

This thesis initially provides an overview of recent advances in telecommunication networks, including standards, architectures, and technologies, highlighting the increased complexity and attack surface these networks face. It then introduces the broader concept of MTD, the application of MTD in various fields of Computer Science, and subsequently focuses on MTD applied to networking systems. As LiMi is one of the main focuses of the thesis, given its usage as a new MTD operation in the Telco Cloud, the thesis discusses the different security properties of LiMi, the state-of-the-art and existing technologies, as well as the optimization algorithms, the traffic redirection methods, and the evaluations made in previous works, highlighting the advantage of containers over VMs for a minimal service downtime during migration. Other lighter MTD operations on networking configurations and topologies are also explored, highlighting the various advantages and challenges associated with each action. As MTD operations come with a cost, we tackled the issue of trade-off between operational cost (*i.e.* resource overhead) and security gains, adding to the equation the overhead on the performance of the moving services.

After this overview of the fundamentals and understanding of the overall subject, this PhD thesis proposed a closed-loop methodology that enables the use of dynamically adaptive, proactive, and reactive MTD operations on the telecommunication network. The methodology starts on the integration with a 5G network by following the ETSI NFV standard interface for monitoring and resource management (phase A), followed by the analysis, network state assessment and decision making phase (B), which triggers an MTD management and orchestration phase (C), and concludes with the enforcement of the MTD operation on the secured 5G assets (D).

A framework with an HLA is designed to provide the components necessary to fulfill each phase of the methodology. The framework's HLA previews three layers with components that communicate with each other. The first layer is the MOL (management and orchestration layer), connecting *MERLINS* to the 5G network via standard NBI provided by ETSI OSM for monitoring and execution of MTD actions. The second layer is the DML (decision-making layer), which represents the cognitive layer of the framework using formal modeling, such as MOMDP, to model the state of a 5G network and define multiple objectives to optimize, as well as AI/ML methods to optimize MTD strategies and decision making. The third layer is the EL (enforcement layer), which implements the MTD operations the framework can perform, such as IP and port shuffling, VNF/CNF re-instantiation, stateless LiMis, and stateful LiMis. The EL also strictly operates with the MOL to enforce them on the secured 5G assets.

Finally, a set of five solutions is designed and implemented, covering the layers of the framework and showcasing the feasibility and possible protection offered by the *MERLINS* MTD approach. The *MOTDEC* solution proposes a way of connecting to the 5G testbed using the ETSI OSM API for network monitoring. It also implements *soft* and *hard MTD actions* applicable to VNFs. The *ContMTD* solution optimizes stateful LiMi operations for containers and applies it to MTD scenarios, mainly focusing on interdependent container applications entailing parallel LiMi of multiple containers. The *TopoFuzzer* solution provides the live handover of open connections built on TCP and QUIC sessions necessary to enable LiMi and open a set of new possible MTD action operations. The *OptSFC* solution optimizes MTD strategies and decision-making for proactive security using deep-RL and MORL, defining a multi-objective optimization problem balancing security gains, QoS, and operational costs. The last solution, *MTDFed*, builds on top of OptSFC, further improving the cognitive solution and enabling a securely federated optimization process of the MTD strategy by applying FL and secure aggregation on deep-RL and MORL, preserving the confidentiality of VNOs' data during local model aggregation.

The PhD thesis presents different evaluations performed on the five solutions to measure their security effects and overhead over resources and service performance. Measurable results from the evaluations comprise *1)* IP and port shuffling *soft MTD actions* performance overhead and security gains; *2)* the effectiveness of stateless LiMi against malware infections and backdoors when using MOTDEC, and the little overhead presented only by the downtime of TopoFuzzer redirection; *3)* the downtime and the scalability of TopoFuzzer against an increased number of connections and the benefits of using QUIC instead of TCP; *4)* the performance of parallel LiMi of stateful containers in cloud-native environments; *5)* the performance of deep-RL MTD optimization algorithms and improvements with MORL; and *6)* the performance overhead using FL over normal deep-RL and MORL as well as the overhead of secure aggregation of non-confidential FL.

6.2 Review of Research Questions and Contributions

The contributions of this thesis (*i.e.*, the *MERLINS* approach and its methodology, framework, and set of solutions) answer all of the research questions defined in the first chapter of the thesis. This section presents an overview of the RQs and their relation with the contributions of the thesis, depicted in Figure 6.1.

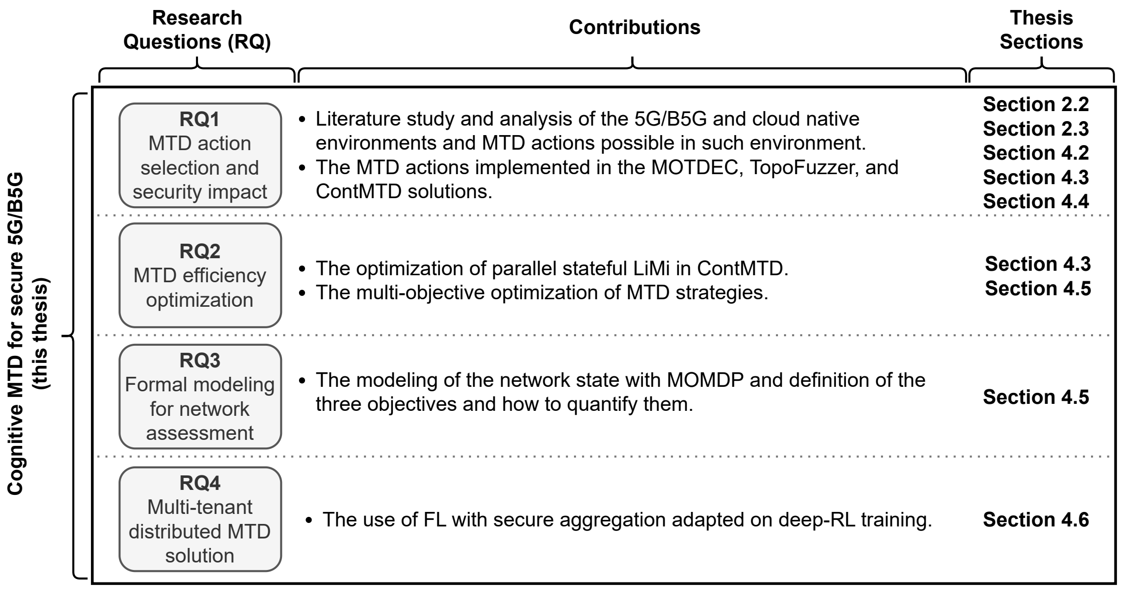

Research Questions (RQ)
Contributions
Thesis Sections
Cognitive MTD for secure 5G/B5G (this thesis)
RQ1 MTD action selection and security impact
• Literature study and analysis of the 5G/B5G and cloud native environments and MTD actions possible in such environment.
• The MTD actions implemented in the MOTDEC, TopoFuzzer, and ContMTD solutions.
Section 2.2
Section 2.3
Section 4.2
Section 4.3
Section 4.4
RQ2 MTD efficiency optimization
• The optimization of parallel stateful LiMi in ContMTD.
• The multi-objective optimization of MTD strategies.
Section 4.3
Section 4.5
RQ3 Formal modeling for network assessment
• The modeling of the network state with MOMDP and definition of the three objectives and how to quantify them.
Section 4.5
RQ4 Multi-tenant distributed MTD solution
• The use of FL with secure aggregation adapted on deep-RL training.
Section 4.6

Figure 6.1: Overview of research questions and corresponding contributions of this thesis.

RQ1: Which MTD actions can be taken on a 5G network and against which attack scenarios, considering the security properties they offer?

To address this research question, a comprehensive literature study and review were conducted to analyze MTD techniques in both traditional networks and broader IT environments. Understanding 5G/B5G Telco Cloud environments was also necessary to derive from the MTD literature study, which MTD operations from existing work can be employed, and what new MTD operations can be performed or improved.

This thesis provides a study on the fundamentals of 5G/B5G networks and the various standards and architectures of the environments, such as NFV and MEC. Moreover, MTD operations on traditional networks and targeted 5G/B5G networks are studied. This led to the design and implementation of MTD operations for 5G assets, including IPv6-based IP and port shuffling, stateless VNF reinstantiation, and stateless VNF LiMi in MOTDEC; live session handover TopoFuzzer; and stateful CNF LiMi in ContMTD.

RQ2: How can we minimize the network and resource overhead associated with MTD operations to ensure system performance and scalability?

MTD operations vary in their resource overhead, QoS impact, and security benefits. This research question focuses on optimizing the timing and selection of MTD operations to maximize security gains while minimizing resource and QoS overhead. Emerging AI/ML methods were explored to design a decision-making system for optimizing MTD strategies. Additionally, improving the efficiency of specific MTD operations was considered a complementary approach.

This thesis addresses RQ2 through two key contributions: *1)* the use of deep-RL and MORL in the OptSFC solution to optimize MTD strategies; and *2)* the enhancement of stateful container LiMi efficiency using a heuristic model for selecting optimal LiMi methods and an ML-based migration time regressor to improve scheduling of parallel LiMis.

RQ3: Which formal modeling approaches are most suitable for representing communication networks to enable near real-time monitoring and security assessments for MTD systems?

This research question arises as a natural extension of RQ2, as optimizing MTD strategies requires a formal representation of the network state to mathematically define and quantify optimization objectives, such as security gains, operational costs, and QoS.

To address this, the thesis employs a MOMDP framework. The MOMDP quantifies rewards for the three optimization objectives and represents virtualized services and resources (e.g., VDUs, VNFs, CNFs, NSs, NSis, and VIMs) along with their monitored metrics. This framework also enables the optimization answering RQ2, executed using deep-RL and MORL.

RQ4: What are the ways to distribute the control system of the cognitive MTD solution in a multi-tenant peer-to-peer environment? Multi-tenant Telco Cloud environments enable the coexistence of VNOs, each managing distinct clients, traffic flows, and services. This setup requires each VNO to deploy its own cognitive MTD solution. However, the deep-RL training process for the OptSFC solution depends on access to both traffic patterns and network monitoring data to optimize the MTD model. Consequently, this RQ investigates a collaborative system where multiple VNOs can enhance their MTD decision-making capabilities while rigorously preserving the confidentiality of each VNO's proprietary data.

This thesis proposes an FL approach with SMC-based aggregation to enable collaborative deep-RL training across VNOs. MTDFed reduces the convergence time for training optimal MTD strategies while preserving the confidentiality of each VNO, compared to training each VNO in isolation on its own data.

Beyond addressing the core research questions, this thesis makes several additional contributions: *(a)* the definition of a closed-loop orchestration methodology enabling the automation of proactive and reactive MTD enforcement, and *(b)* the HLA design, providing an extendable and composable framework that can increase its portfolio of MTD actions, all while using a standard-aware interface to interact with NFV 5G/B5G cloud-native networks.

6.3 Outlook of Future Research

While an MTD framework provides an additional layer of security to future cloud and telecommunication networks, it can also be a new point of attack. The ML model, for instance, is a sensitive attack surface that can be vulnerable to hijacking and poisoning, triggering useless MTD operations to waste resources or as a DoS attack [3]. Measures can be implemented to safeguard data integrity and authenticate the origin of data, particularly from network endpoints providing monitoring data. To mitigate these risks, measures such as data integrity verification and endpoint authentication must be implemented to ensure the reliability of monitoring data. Additionally, introducing noise to ML predictions can serve as a countermeasure against model inversion attacks [3].

As discussed in the evaluation of the MTDFed solution, the secure aggregation method in FL ensures model confidentiality but remains vulnerable to adversarial poisoning attacks from malicious participants, such as adversarial sample injection, label flipping, and min-max attacks. To address these vulnerabilities, MTDFed can be further strengthened by integrating additional security measures that enhance the robustness of FL models against malicious actors while preserving privacy. A promising approach is the incorporation of MTD strategies into the FL process itself, such as dynamically selecting different aggregation algorithms or shuffling SMC aggregation nodes, all while maintaining the confidentiality guaranteed by SMC [179].

Moreover, for technical, economic, and legal reasons, the decisions of the deep-RL agents should also be humanly explainable (*i.e.*, with statistics and logic). For instance, ENISA[32] emphasizes the explainability of AI and ML algorithms, especially DNNs, which are not explainable by nature. Integration of explainable AI into MTD remains an open research question. It is potentially possible to implement different explainable RL (XRL) methods present in the literature, such as the XRL via reward decomposition [180]. With this method, rewards can be classified according to semantically meaningful reward types, which fits well with the multi-objective nature of the MTD optimization problem.

Finally, another path for future research is the exploration of energy-efficient MTD strategies as another optimization objective. This constitutes a relevant research direction in light of environmental considerations related to sustainable energy consumption and the overarching goal of achieving a global net-zero footprint and the UN Sustainable Development Goals (SDGs) [31]. In this context, the integration of MTD techniques can consider the carbon emissions associated with network slices' activities, and the energy cost for MTD actions can be integrated into the optimization model: *e.g.*, by strategically placing VNFs within cloud nodes powered by green energy sources rather than fossil-fuel-powered nodes.

Bibliography

[1] P. Rost et al. Network slicing to enable scalability and flexibility in 5G mobile networks. *IEEE Communications Magazine*, 55, 2017.

[2] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina. Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.

[3] Pawani Porambage, Gurkan Gur, Diana Pamela Moya Osorio, Madhusanka Liyanage, Andrei Gurtov, and Mika Ylianttila. The roadmap to 6G security and privacy. *IEEE Open Journal of the Communications Society*, 2:1094–1122, 2021.

[4] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[5] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller, and Fabian Schneider. NFV and SDN — key technology enablers for 5G networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, 2017.

[6] Ian F Akyildiz, Ahan Kak, and Shuai Nie. 6G and beyond: The future of wireless communications systems. *IEEE Access*, 8:133995–134030, 2020.

[7] Joel Margolis, Tae Tom Oh, Suyash Jadhav, Young Ho Kim, and Jeong Neyo Kim. An in-depth analysis of the mirai botnet. In *2017 International Conference on Software Security and Assurance (ICSSA)*, pages 6–12, 2017.

[8] Pawani Porambage, Gurkan Gur, Diana Pamela Moya Osorio, Madhusanka Livanage, and Mika Ylianttila. 6G security challenges and potential solutions. In *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, pages 622–627, 2021.

[9] Rui Zhuang, Scott A. DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, page 31–40, 2014.

[10] Network Function Virtualization (NFV). Standard, European Telecommunications Standards Institute (ETSI), Nice, FR, March 2012. [Online; accessed 21-October-2022].

[11] Trung-Kien Le, Umer Salim, and Florian Kaltenberger. An overview of physical layer design for ultra-reliable low-latency communications in 3gpp releases 15, 16, and 17. *IEEE access*, 9:433–444, 2020.

[12] GSAcom. 5g-marketsnapshot may 2024. https://gsacom.com/paper/5g-marketsnapshot-may-2024/, 2024. Accessed: 2024-07-10.

[13] Haibo Zhou, Wenchao Xu, Jiacheng Chen, and Wei Wang. Evolutionary v2x technologies toward the internet of vehicles: Challenges and opportunities. *Proceedings of the IEEE*, 108(2):308–323, 2020.

[14] Shree Krishna Sharma, Isaac Woungang, Alagan Anpalagan, and Symeon Chatzinotas. Toward tactile internet in beyond 5g era: Recent advances, current issues, and future directions. *Ieee Access*, 8:56948–56991, 2020.

[15] International Telecommunication Union. Framework and overall objectives of the future development of imt for 2030 and beyond. Recommendation ITU-R M.[IMT-2030], ITU-R, 2023.

[16] Olaonipekun Oluwafemi Erunkulu, Adamu Murtala Zungeru, Caspar K. Lebekwe, Modisa Mosalaosi, and Joseph M. Chuma. 5g mobile communication applications: A survey and comparison of use cases. *IEEE Access*, 9:97251–97295, 2021.

[17] Anne Morris. Swisscom takes its time with 5g sa, open ran. https://www.fierce-network.com/wireless/swisscom-takes-its-time-5g-sa-open-ran, 2024. Accessed: 2024-07-10.

[18] ETSI. Osm Release ELEVEN release notes. https://osm-download.etsi.org/ftp/osm-11.0-eleven/OSM_Release_ELEVEN_Release_Notes.pdf. Accessed: 2022-02-11.

[19] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[20] Jorge Gallego-Madrid, Ramon Sanchez-Iborra, Pedro M Ruiz, and Antonio F Skarmeta. Machine learning-based zero-touch network and service management: A survey. *Digital Communications and Networks*, 8(2):105–123, 2022.

[21] European Telecommunications Standards Institute (ETSI). Zero-touch network and service management (zsm); closed-loop automation; part 1: enablers. Technical Specification ETSI GS ZSM 009-1 V1.1.1, ETSI, 2021. Available at: `https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/00901/01.01.01_60/gs_ZSM00901v010101p.pdf`.

[22] Geoffrey Chollon, Dhouha Ayed, Rodrigo Asensio Garriga, Alejandro Molina Zarca, Antonio Skarmeta, Maria Christopoulou, Wissem Soussi, Gürkan Gür, and Uwe Herzog. ETSI ZSM Driven Security Management in Future Networks. In *2022 IEEE Future Networks World Forum (FNWF)*, pages 334–339, 2022.

[23] Fabio Giust, Gianluca Verin, Kiril Antevski, Joey Chou, Yonggang Fang, Walter Featherstone, Francisco Fontes, Danny Frydman, Alice Li, Antonio Manzalini, et al. Mec deployments in 4g and evolution towards 5g. *ETSI White paper*, 24(2018):1–24, 2018.

[24] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, 6:47980–48009, 2018.

[25] Panagiotis Gkonis, Anastasios Giannopoulos, Panagiotis Trakadas, Xavi Masip-Bruin, and Francesco D'Andria. A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues. *Future Internet*, 15(12):383, 2023.

[26] Mithun Mukherjee, Mian Guo, Jaime Lloret, and Qi Zhang. Leveraging intelligent computation offloading with fog/edge computing for tactile internet: Advantages and limitations. *IEEE Network*, 34(5):322–329, 2020.

[27] Wissem Soussi, Gürkan Gür, and Burkhard Stiller. Moving target defense (mtd) for 6g edge-to-cloud continuum: A cognitive perspective. *IEEE Network*, pages 1–1, 2024.

[28] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307, 2004.

[29] Jin-Hee Cho, Dilli P. Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J. Moore, Dong Seong Kim, Hyuk Lim, and Frederica F. Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22(1):709–745, 2020.

[30] Wissem Soussi, Maria Christopoulou, George Xilouris, and Gurkan Gur. Moving target defense as a proactive defense element for beyond 5G. *IEEE Communications Standards Magazine*, 5(3):72–79, 2021.

[31] United Nations. Make the sdgs a reality. https://sdgs.un.org, 2025. Accessed: 2025-05-02.

[32] European Union Agency for Cybersecurity (ENISA). Securing machine learning algorithms. Technical report, 2023.

[33] Wissem Soussi, Gürkan Gür, and Burkhard Stiller. Democratizing container live migration for enhanced future networks - a survey. *ACM Comput. Surv.*, 57(4), December 2024.

[34] Heleen L Van Soest, Michel GJ den Elzen, and Detlef P van Vuuren. Net-zero emission targets for major emitting countries consistent with the Paris agreement. *Nature communications*, 12(1):2140, 2021.

[35] Soo-Jin Moon, Vyas Sekar, and Michael K. Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1595–1606, New York, NY, USA, 2015. Association for Computing Machinery.

[36] Rui Huang, Hongqi Zhang, Yi Liu, and Shie Zhou. RELOCATE: A Container Based Moving Target Defense Approach. *PoS*, CENet2017:008, 2017.

[37] Mohamed Azab and Mohamed Eltoweissy. Migrate: Towards a lightweight moving-target defense against cloud side-channels. *Proceedings - 2016 IEEE Symposium on Security and Privacy Workshops, SPW 2016*, pages 96–103, 8 2016.

[38] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 2015.

[39] Paul Voigt and Axel Von dem Bussche. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[40] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, and Muhammad Ali Babar. Understanding and addressing quality attributes of microservices architecture: A systematic literature review. *Information and Software Technology*, 131:106449, 2021.

[41] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[42] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine*, 55(3):38–43, 2017.

[43] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.

[44] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K. Leung. Dynamic service migration in mobile edge computing based on markov decision process. *IEEE/ACM Transactions on Networking*, 27(3):1272–1288, 2019.

[45] International Energy Agency (IEA). Data centres and data transmission networks. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks, 2024. Accessed: January 10, 2024.

[46] Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power management in grid computing with xen. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 407–416. Springer, 2006.

[47] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, volume 8, pages 337–350, 2008.

[48] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[49] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*, pages 254–265. Springer, 2009.

[50] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 101–110, 2009.

[51] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient live migration of virtual machines using shared storage. *ACM Sigplan Notices*, 48(7):41–50, 2013.

[52] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 2009.

[53] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees De Laat, Joe Mambretti, Inder Monga, Bas Van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the MAN/WAN. *Future Generation Computer Systems*, 22(8):901–907, 2006.

[54] Fei Ma, Feng Liu, and Zhen Liu. Live virtual machine migration based on improved pre-copy approach. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 230–233. IEEE, 2010.

[55] Shashank Sahni and Vasudeva Varma. A hybrid approach to live migration of virtual machines. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–5. IEEE, 2012.

[56] Thad Benjaponpitak, Meatasit Karakate, and Kunwadee Sripanidkulchai. Enabling live migration of containerized applications across clouds. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2529–2538. IEEE, 2020.

[57] Sai Venkat Naresh Kotikalapudi. Comparing live migration between linux containers and kernel virtual machine: investigation study in terms of parameters. Master's thesis, 2017.

[58] Kiranpreet Kaur, Fabrice Guillemin, and Francoise Sailhan. Container placement and migration strategies for cloud, fog, and edge data centers: A survey. *International Journal of Network Management*, 32:e2212, 11 2022.

[59] Adel Alshamrani, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, 21(2):1851–1877, 2019.

[60] Mohamed Esam Elsaid, Hazem M. Abbas, and Christoph Meinel. Virtual machines pre-copy live migration cost modeling and prediction: a survey. *Distributed and Parallel Databases*, 40:441–474, 9 2022.

[61] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, jul 2009.

[62] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

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

[64] Gavin A. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, University of Cambridge, Department of Engineering, Cambridge, UK, 1994.

[65] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 387–395. PMLR, 22–24 Jun 2014.

[66] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[69] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744, 2022.

[70] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[71] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[72] Kristof Van Moffaert, Madalina Drugan, and Ann Nowe. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL*, 04 2013.

[73] John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

[74] Volodymyr Mnih et al. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, New York, New York, USA, 2016. PMLR.

[75] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[76] Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

[77] Diederik M Roijers, Denis Steckelmacher, and Ann Nowé. Multi-objective reinforcement learning for the expected utility of the return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM*, volume 2018, 2018.

[78] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[79] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[80] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.

[81] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[82] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. OpenFlow random host mutation: Transparent moving target defense using software defined networking. In *HotSDN'12 - Proc. of the 1st ACM Int. Workshop on Hot Topics in Software Defined Networks*, pages 127–132, 2012.

[83] Dilli P. Sharma, Jin-Hee Cho, Terrence J. Moore, Frederica F. Nelson, Hyuk Lim, and Dong Seong Kim. Random host and service multiplexing for moving target defense in software-defined networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[84] Yue Bin Luo, Bao Sheng Wang, Xiao Feng Wang, Xiao Feng Hu, Gui Lin Cai, and Hao Sun. RPAH: Random port and address hopping for thwarting internal and external adversaries. In *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*, pages 263–270, 2015.

[85] Yue Bin Luo, Bao Sheng Wang, Xiao Feng Wang, Xiao Feng Hu, and Gui Lin Cai. Tpah: A universal and multi-platform deployable port and address hopping mechanism. *IET Conference Publications*, 2015, 2015.

[86] Abdullah Aydeger, Nico Saputro, and Kemal Akkaya. A moving target defense and network forensics framework for ISP networks using SDN and NFV. *Future Generation Computer Systems*, 94, 2019.

[87] Mariusz Rawski, Slawomir Kukliński, Piotr Sapiecha, Marek Pelka, Grzegorz Przytula, Przemysław Wojslaw, and Krzysztof Szczypiorski. MMTD: MANO-based moving target defense for corporate networks. In *2020 World Conference on Computing and Communication Technologies (WCCCT)*, 2020.

[88] Huangxin Wang, Fei Li, and Songqing Chen. Towards cost-effective moving target defense against ddos and covert channel attacks. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, MTD '16, page 15–25, New York, NY, USA, 2016. Association for Computing Machinery.

[89] Matheus Torquato, Paulo Maciel, and Marco Vieira. Analysis of vm migration scheduling as moving target defense against insider attacks. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 194–202, New York, NY, USA, 2021. Association for Computing Machinery.

[90] OpenVZ. Openvz. https://www.openvz.org/, 2024. Accessed on 2024-04-04.

[91] Adrian Reber. Forensic container checkpointing in kubernetes. Accessed: January 11, 2024.

[92] Peter Schuurman. Kubernetes 1.27: Statefulset start ordinal simplifies migration. Accessed: January 11, 2024.

[93] Maksym Planeta, Jan Bierbaum, Leo Sahaya Daphne Antony, Torsten Hoefler, and Hermann Härtig. MigrOS: Transparent Live-Migration support for containerised RDMA applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 47–63. USENIX Association, July 2021.

[94] Shunmugapriya Ramanathan, Abhishek Bhattacharyya, Koteswararao Kondepu, and Andrea Fumagalli. Enabling containerized central unit live migration in 5G radio access network: An experimental study. *Journal of Network and Computer Applications*, 221:103767, 2024.

[95] Niklas Eiling, Jonas Baude, Stefan Lankes, and Antonello Monti. Cricket: A virtualization layer for distributed execution of CUDA applications with checkpoint/restart support. *Concurrency and Computation: Practice and Experience*, 34:e6474, 6 2022.

[96] Hongliang Liang, Qiong Zhang, Mingyu Li, and Jianqiang Li. Toward migration of SGX-enabled containers. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2019.

[97] Eryk Schiller, Jesutofunmi Ajayi, Silas Weber, Torsten Braun, and Burkhard Stiller. Toward a live BBU container migration in wireless networks. *IEEE Open Journal of the Communications Society*, 3:301–321, 2022.

[98] Kiranpreet Kaur, Fabrice Guillemin, and Francoise Sailhan. Live migration of containerized microservices between remote kubernetes clusters. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2023.

[99] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Live service migration in mobile edge clouds. *IEEE Wireless Communications*, 25(1):140–147, 2018.

[100] Carlo Puliafito, Carlo Vallati, Enzo Mingozzi, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Container migration in the fog: A performance evaluation. *Sensors*, 19(7), 2019.

[101] Mohamed Azab, Bassem M. Mokhtar, Amr S. Abed, and Mohamed Eltoweissy. Smart moving target defense for linux container resiliency. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 122–130, 2016.

[102] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schioberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, page 169–179. Association for Computing Machinery, 2007.

[103] Ali J. Fahs and Guillaume Pierre. Proximity-aware traffic routing in distributed fog computing platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 478–487, 2019.

[104] Fikirte Teka, Chung-Horng Lung, and Samuel Ajila. Seamless live virtual machine migration with cloudlets and multipath TCP. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 607–616, 2015.

[105] Laura Lemmi, Carlo Puliafito, Antonio Virdis, and Enzo Mingozzi. Ensuring lossless workload migration at the edge with SRv6. In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2023.

[106] Carlo Puliafito, Antonio Virdis, and Enzo Mingozzi. The impact of container migration on fog services as perceived by mobile things. In *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 9–16, 2020.

[107] Andrej Binder, Tomas Boros, and Ivan Kotuliak. A SDN based method of TCP connection handover. In Ismail Khalil, Erich Neuhold, A Min Tjoa, Li Da Xu, and Ilsun You, editors, *Information and Communication Technology*, pages 13–19, Cham, 2015. Springer International Publishing.

[108] Vitor A. Cunha, Daniel Corujo, Joao P. Barraca, and Rui L. Aguiar. Using Linux TCP connection repair for mid-session endpoint handover: a security enhancement use-case. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 174–180, 2020.

[109] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the Internet. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC '07)*, pages 1–7, 2007.

[110] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[111] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986, February 2021.

[112] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-Peer communication across network address translators. In *2005 USENIX Annual Technical Conference (USENIX ATC 05)*, Anaheim, CA, April 2005. USENIX Association.

[113] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. A POMDP approach to the dynamic defense of large-scale cyber networks. *IEEE Transactions on Information Forensics and Security*, 13(10):2490–2505, oct 2018.

[114] Sailik Sengupta, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. General sum markov games for strategic detection of advanced persistent threats using moving target defense in cloud networks. In *Decision and Game Theory for Security*. Springer International Publishing, 2019.

[115] Xinzhong Chai, Yasen Wang, Chuanxu Yan, Yuan Zhao, Wenlong Chen, and Xiaolei 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)*, pages 375–379, 2020.

[116] Seunghyun Yoon, Jin-Hee Cho, Dong Seong Kim, Terrence J. Moore, Frederica Free-Nelson, and Hyuk Lim. DESOLATER: Deep Reinforcement Learning-Based Resource Allocation and Moving Target Defense Deployment Framework. *IEEE Access*, 9:70700–70714, 2021.

[117] Microsoft Defender Research Team. Cyberbattlesim, 2021. Created by Christian Seifert et al.

[118] Quan Jia, Kun Sun, and Angelos Stavrou. Motag: Moving target defense against internet denial of service attacks. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2013.

[119] Sam Smith, Matthew Southerby, S Setiniyaz, Robert Apsimon, and Graeme Burt. Multiobjective optimization and pareto front visualization techniques applied to normal conducting RF accelerating structures. *Physical Review Accelerators and Beams*, 25(6):062002, 2022.

[120] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[121] Enrique Tomás Martínez Beltrán, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Mitigating communications threats in decentralized federated learning through moving target defense. *Wireless Networks*, 30(9):7407–7421, 2024.

[122] Chao Feng, Alberto Huertas Celdrán, Michael Vuong, Gérôme Bovet, and Burkhard Stiller. Voyager: Mtd-based aggregation protocol for mitigating poisoning attacks on dfl. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9, 2024.

[123] Zan Zhou, Changqiao Xu, Shujie Yang, Xiaoyan Zhang, Hongjing Li, Sizhe Huang, and Gabriel-Miro Muntean. Safeguarding privacy and integrity of federated learning in heterogeneous cross-silo IoRT Environments: A moving target defense approach. *IEEE Network*, 38(3):25–32, 2024.

[124] Amiya Kumar Sahu, Suraj Sharma, Mohammad Tanveer, and Rohit Raja. Internet of things attack detection using hybrid deep learning model. *Computer Communications*, 176:146–154, 2021.

[125] Abebe Abeshu and Naveen Chilamkurti. Deep learning: The frontier for distributed attack detection in fog-to-things computing. *IEEE Communications Magazine*, 56(2):169–175, 2018.

[126] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[127] Kaiqiang Qi and Chenyang Yang. Popularity prediction with federated learning for proactive caching at wireless edge. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2020.

[128] Yiwen Han, Ding Li, Haotian Qi, Jianji Ren, and Xiaofei Wang. Federated learning-based computation offloading optimization in edge computing-supported internet of things. In *Proceedings of the ACM Turing Celebration Conference - China*, ACM TURC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[129] Fan Meng, Peng Chen, Lenan Wu, and Julian Cheng. Power allocation in multi-user cellular networks: Deep reinforcement learning approaches. *IEEE Transactions on Wireless Communications*, 19(10):6255–6267, 2020.

[130] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. Resource allocation based on deep reinforcement learning in IoT edge computing. *IEEE Journal on Selected Areas in Communications*, 38(6):1133–1146, 2020.

[131] Jungyeon Baek and Georges Kaddoum. Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. *IEEE Internet of Things Journal*, 8(2):1041–1056, 2020.

[132] Abdulmalik Alwarafy, Mohamed Abdallah, Bekir Sait Ciftler, Ala Al-Fuqaha, and Mounir Hamdi. Deep reinforcement learning for radio resource allocation and management in next generation heterogeneous wireless networks: A survey. *arXiv preprint arXiv:2106.00574*, 2021.

[133] Laizhong Cui, Xiaoxin Su, Zhongxing Ming, Ziteng Chen, Shu Yang, Yipeng Zhou, and Wei Xiao. Creat: Blockchain-assisted compression algorithm of federated learning for content caching in edge computing. *IEEE Internet of Things Journal*, 9(16):14151–14161, 2022.

[134] Shihao Shen, Yiwen Han, Xiaofei Wang, and Yan Wang. Computation offloading with multiple agents in edge-computing–supported iot. *ACM Trans. Sen. Netw.*, 16(1), 2019.

[135] ETSI Industry Specification Group (ISG) ZSM. Zero touch network & service management (ZSM). https://www.etsi.org/technologies/zero-touch-network-service-management. Accessed: 2022-02-11.

[136] Wissem Soussi, Maria Christopoulou, Gurkan Gur, and Burkhard Stiller. MERLINS – moving target defense enhanced with Deep-RL for NFV in-depth security. In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 65–71, 2023.

[137] Wissem Soussi, Cantali Gökcan, Gürkan Gür, and Burkhard Stiller. Deep reinforcement learning for radio resource allocation and management in next generation heterogeneous wireless networks: A survey. *submitted to ACM SIGCOMM proceedin*, 2025.

[138] Wissem Soussi, Maria Christopoulou, Themis Anagnostopoulos, Gurkan Gur, and Burkhard Stiller. Topofuzzer — a network topology fuzzer for moving target defense in the telco cloud. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–5, Miami, Florida, USA, 2023.

[139] Nicholas Mayone, Pascal Kunz, Beytullah Yigit, Wissem Soussi, Burkhard Stiller, and Gürkan Gür. Ipv6 connection shuffling for moving target defense (mtd) in sdn. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 373–378, 2024.

[140] Themis Anagnostopoulos. Katana network slice manager. Accessed: October 21, 2022.

[141] KubeVirt. KubeVirt [Official site]. https://kubevirt.io/. accessed: 19-Aug-2023.

[142] AWS. Amazon ec2 instance types. https://web.archive.org/web/20250102014806/https://aws.amazon.com/ec2/instance-types/.

[143] Datadog. Docker adoption. https://www.datadoghq.com/docker-adoption/.

[144] Wissem Soussi, Gürkan Gür, and Burkhard Stiller. Contmtd: Live migration optimization for containers in moving target defense. In *submitted to ACM SIGCOMM 2025*, 2025.

[145] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[146] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9, 1996.

[147] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[148] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[149] Pablo Neira Ayuso. Securing machine learning algorithms, 2012. Accessed: 06/02/2025.

[150] National vulnerability database (NVD) - vulnerability metrics. https://nvd.nist.gov/vuln-metrics/cvss. Accessed: 2022-02-11.

[151] National vulnerability database (NVD). https://nvd.nist.gov.

[152] Q3 Cloud Spending Up Over $11 Billion from 2021 Despite Major Headwinds. article, Synergy Research Group, 10 2022.

[153] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[154] Cheng-Yen Tang, Chien-Hung Liu, Woei-Kae Chen, and Shingchern D. You. Implementing action mask in proximal policy optimization (PPO) algorithm. *ICT Express*, 6(3):200–203, 2020.

[155] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[156] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in Neural Information Prsyocessing Systems 32*. 2019.

[157] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[158] Antonin Raffin et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[159] Florian Felten and Lucas N. Alegre. MORL-baselines: Multi-objective reinforcement learning algorithms implementations. https://github.com/LucasAlegre/morl-baselines, 2022.

[160] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2447–2461, 2020.

[161] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16937–16947. Curran Associates, Inc., 2020.

[162] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA, 2019. Internet Society.

[163] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1253–1269, New York, NY, USA, 2020. Association for Computing Machinery.

[164] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[165] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti

Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[166] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[167] Sukchan Lee. Open5gs. Accessed: October 21, 2022.

[168] Ali Gungör. Ueransim. Accessed: October 21, 2022.

[169] Cedric Westphal, Stefan Lederer, Christopher Mueller, Andrea Detti, Daniel Corujo, Jianping Wang, Marie-Jose Montpetit, Niall Murray, Christian Timmerer, Daniel Posch, Aytac Azgin, and Will (Shucheng) LIU. Adaptive Video Streaming over Information-Centric Networking (ICN). RFC 7933, August 2016.

[170] Jean Le Feuvre, Cyril Concolato, and Jean-Claude Moissinac. GPAC: Open source multimedia framework. In *Proc. of the 15th ACM International Conference on Multimedia*, MM '07, page 1009–1012, 2007.

[171] Vox. 2022, in 7 minutes Vox. https://www.vox.com/videos/2022/12/27/23523151/2022-rewind-year-review-video.

[172] Wissem Soussi, Maria Christopoulou, George Xilouris, Edgardo Montes de Oca, Vincent Lefebvre, Gürkan Gür, and Burkhard Stiller. Demo: Closed-loop security orchestration in the telco cloud for moving target defense. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–3, 2023.

[173] Amazon Web Services. Amazon EKS Service Level Agreement, 2024. Accessed: 2024-10-17.

[174] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, l3 cache Side-Channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, 2014. USENIX Association.

[175] James E Kelley Jr and Morgan R Walker. Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 160–173, 1959.

[176] Rene A Yamin and David J Harmelink. Comparison of linear scheduling model (lsm) and critical path method (cpm). *Journal of construction engineering and management*, 127(5):374–381, 2001.

[177] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at QUIC: An approach for rigorous evaluation of rapidly evolving transport protocols. In *IMC '17: Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 290–303, New York, NY, USA, 2017. Association for Computing Machinery.

[178] Charles Miller, Michael Pelosi, and Michael Brown. Domain fronting through microsoft azure and cloudflare: How to identify viable domain fronting proxies. In *Proceedings of the Def Con 31*, pages 1–11. Def Con, 2023.

[179] Chao Feng, Alberto Huertas Celdrán, Zien Zeng, Zi Ye, Jan von der Assen, Gérôme Bovet, and Burkhard Stiller. Leveraging mtd to mitigate poisoning attacks in decentralized fl with non-iid data. In *2024 IEEE International Conference on Big Data (BigData)*, pages 7745–7754, 2024.

[180] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*, 2019.

[181] Sumeyya Birtane, Mays Al-Naday, Francesco Paolucci, Rana Abu Bakar, Vincent Lefebvre, Virgilios Passas, Sarantis Kalafatidis, Antonios Lalas, Anastasios Drosou, Edgardo Montes de Oca, Ana Rosa Cavalli, Péter Vörös, Mohammed Alshawki, Burkhard Stiller, Wissem Soussi, Gokcan Cantali, and Gürkan Gür. Footprint-optimized orchestration and management of secure complex services over 6g continuum. In *2024 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 383–388, 2024.

[182] Anthony Mamaril, Rinchen Kolodziejczyk, Wissem Soussi, and Gürkan Gür. Containers on the move: An experimental analysis of container migration in kubernetes. In *ICC 2025 - IEEE International Conference on Communications*, 2025.

[183] Anthony Mamaril, Rinchen Kolodziejczyk, Wissem Soussi, and Gürkan Gür. Exploring live payload migrations for mtd in microservices architecture. In *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pages 1–5, 2024.

[184] Maria Christopoulou, Wissem Soussi, George Xilouris, Gürkan Gür, Edgardo Montes de Oca, Harilaos Koumaras, and Burkhard Stiller. Ai-enabled slice protection exploiting moving target defense in 6g networks. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), online, 8-11 June 2021*, 2021.

A Publications

This PhD thesis resulted in several scientific works published in selected venues. These works were all directly or indirectly related to the topic of this thesis. Further, master's and bachelor's theses were supervised and developed in the cybersecurity and network management context, whose individual outcomes in terms of proposed solutions and evaluations contributed toward the overall *MERLINS* approach and results.

A.1 Contribution of Own Publications Within Chapters

The PhD thesis objectives outlined at the macro level are typically decomposed into several specific goals achieved by publications across the thesis period. Thus, it is relevant to highlight where those publications appear as core or additional elements in each chapter of this PhD thesis. Also, several students' thesis (*i.e.*, Master Thesis (MSc), Master Project (MAP), Bachelor Thesis (BSc), and Bachelor Project (BAP)) contributed to this PhD thesis by implementing solutions motivated, designed, and supervised by the author of this PhD thesis. Table A.1 lists own contributions in the PhD thesis' chapters.

A.2 List of Publications

This section lists publications made by the author during the PhD thesis period. In addition to the previously cited publications that contributed directly to the thesis, the author's research and collaboration within the Communication Systems Group (CSG) resulted in several publications from 2021 to 2025. All of these publications are related to computer networks and cybersecurity.

Table A.1: List of Publications per Chapter

Chapter	**Related Publications**				**Student Thesis**			
	Journal	**Full Paper**	**Short Paper**	**Poster/Demo Paper**	**BSc**	**BAP**	**MSc**	**MAP**
1. Introduction	[30]	[136]	-	-	-	-	-	-
2. Theoretical Foundations	[30, 33]	[22, 181, 182]	[183]	-	[Anthony & Rinchen]	[Anthony & Rinchen]	[Zeno]	[Zeno, Michael]
3. State-of-the-Art	-	-	-	-	-	-	-	-
4. *MERLINS* Approach	[27, 30, 33]	[136, 139, 144]	[138]	[172, 184]	[Nicolas & Pascal]	-	-	-
5. Evaluations	[27]	[136, 139, 144]	[138]	[172, 184]	[Nicolas & Pascal]	-	-	-
6. Conclusions	-	-	-	-	-	-	-	-

Publication Title
[30] Moving Target Defense as a Proactive Defense Element for Beyond 5G
[184] AI-Enabled Slice Protection Exploiting Moving Target Defense in 6G Networks
[172] Demo: Closed-Loop Security Orchestration in the TelcoCloud for Moving Tar- get Defense
[138] TopoFuzzer - A Network Topology Fuzzer for Moving Target Defense in the TelcoCloud
[136] MERLINS–Moving Target Defense Enhanced with Deep-RL for NFV In-Depth Security
[27] Moving Target Defense (MTD) for 6G Edge-to-Cloud Continuum: A Cognitive Perspective
[33] Democratizing Container Live Migration for Enhanced Future Networks - A Survey
[144] ConMTD: Live Migration Optimization for Containers in Moving Target Defense
[22] ETSI ZSM Driven Security Management in Future Networks
[181] Footprint-Optimized Orchestration and Management of Secure Complex Services over 6G Continuum
[182] Containers on the Move: An Experimental Analysis of Container Migration in Kubernetes
[183] Exploring Live Payload Migrations for MTD in Microservices Architecture
[139] IPv6 Connection Shuffling for Moving Target Defense (MTD) in SDN

A.2.1 First Author Publications

The author of this thesis led the publications listed below, thus, having him as the main contributor (*i.e.*, First Author).

2025

- ***(under review)****[Full Paper]* **W. Soussi**, G. Cantali, G. Gür, B. Stiller: **ContMTD: Live Migration Optimization for Containers in Moving Target Defense**; 2025.

2024

- *[Journal]* **W. Soussi**, G. Gür, B. Stiller: **Democratizing Container Live Migration for Enhanced Future Networks - A Survey**; ACM Computing Surveys 57, 4, Article 97 (April 2025), 37 pages.

- *[Journal]* **W. Soussi**, G. Gür, B. Stiller: **Moving Target Defense (MTD) for 6G Edge-to-Cloud Continuum: A Cognitive Perspective**; IEEE Network, vol. 39, no. 1, pp. 149-156, Jan. 2025.

2023

- *[Full Paper]* **W. Soussi**, M. Christopoulou, G. Gür, B. Stiller: **MERLINS–Moving Target Defense Enhanced with Deep-RL for NFV In-Depth Security**; IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Dresden, Germany, 2023, pp. 65-71.

- *[PhD School]* **W. Soussi**, G. Gür, B. Stiller: **ML-Driven Moving Target Defense for Network Slice Protection**; 11th TMA PhD School at the Network Traffic Measurement and Analysis Conference (TMA), Naples, Italy, 26-29 June 2023.

- *[Short Paper]* **W. Soussi**, M. Christopoulou, T. Anagnostopoulos, G. Gür, B. Stiller: **Topo-Fuzzer - A Network Topology Fuzzer for Moving Target Defense in the TelcoCloud**; NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 2023, pp. 1-5.

- *[Demo Paper]* **W. Soussi**, M. Christopoulou, G. Xilouris, E.M.d Oca, V. Lefebvre, G. Gür, B. Stiller: **Demo: Closed-Loop Security Orchestration in the TelcoCloud for Moving Target Defense**; NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 2023, pp. 1-3.

2021

- *[Full Paper]* **W. Soussi**, M. Christopoulou, G. Gür, B. Stiller: **Moving Target Defense as a Proactive Defense Element for Beyond 5G**; IEEE Communications Standards Magazine, vol. 5, no. 3, pp. 72-79, September 2021.

A.2.2 Co-authorship Publications

Different successful collaborations were placed, having the author of this PhD thesis as one of the collaborators. All of these collaborations were related to the Network Security field and the topic of this thesis, resulting in the publications listed below.

2025

- *[Workshop Paper]* Y. Abdullah, M.B Alshawki, P. Ligeti, **W. Soussi**, B. Stiller: **Byzantine-Resilient Federated Learning: Evaluating MPC Approaches**; IEEE ICDCS Workshop 2025, FL4WEB Workshop at the 45th IEEE International Conference on Distributed Computing Systems, Glasgow, Scotland, UK, 20-23 July 2025.

- *[Full Paper]* A. Mamaril, R. Kolodziejczyk, **W. Soussi**, G. Gür: **Containers on the Move: An Experimental Analysis of Container Migration in Kubernetes**; ICC 2025 - IEEE International Conference on Communications, Montreal, Canada, 8-12 June 2025.

- *[Full Paper]* M. Osswald, T. Schönenberger, G. Cantali, **W. Soussi**, G. Gür: **Anomaly Detection in Microservices Architecture Using Graph Neural Networks**; Euromicro International Conference on Parallel, Distributed, and Network-based Processing (PDP 2025), Turin, Italy, 12-14 March 2025.

2024

- *[Full Paper]* S. Birtane, **W. Soussi**, G. Gür, B. Stiller, et al.: **Footprint-Optimized Orchestration and Management of Secure Complex Services over 6G Continuum**; 2024 IEEE Conference on Standards for Communications and Networking (CSCN), Belgrade, Serbia, 2024, pp. 383-388.

- *[Full Paper]* N. Mayone, P. Kunz, B. Yigit, **W. Soussi**, B. Stiller, G. Gür: **IPv6 Connection Shuffling for Moving Target Defense (MTD) in SDN**; 2024 IEEE International Conference on Cyber Security and Resilience (CSR), London, United Kingdom, 2024, pp. 373-378.

- *[Recent Results Paper]* A. Mamaril, R. Kolodziejczyk, **W. Soussi**, G. Gür: **Exploring Live Payload Migrations for MTD in Microservices Architecture**; 2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring), Singapore, Singapore, 2024, pp. 1-5.

2023

- *[Full Paper]* O. Kalinagac, **W. Soussi**, Y. Anser, C. Gaber, G. Gür: **Root Cause and Liability Analysis in the Microservices Architecture for Edge IoT Services**; ICC 2023 - IEEE International Conference on Communications, Rome, Italy, 2023, pp. 3277-3283.

- *[Short Paper]* Z. Heeb, O. Kalinagac, **W. Soussi**, G. Gür: **IoMiRCA: Root cause analysis in IoT-extended 5G microservice environments**; 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), ACM, New York, NY, USA, 106–108.

2022

- *[Demo Paper]* O. Kalinagac, **W. Soussi**, G. Gür: **Graph Based Liability Analysis for the Microservice Architecture**; 2022 18th International Conference on Network and Service Management (CNSM), Thessaloniki, Greece, 2022, pp. 364-366.

- *[Full Paper]* G. Chollon, R.A. Garriga, A.M. Zarca, A. Skarmeta, M. Christopoulou, **W. Soussi**, G. Gür, U. Herzog: **ETSI ZSM Driven Security Management in Future Networks**; 2022 IEEE Future Networks World Forum (FNWF), Montreal, QC, Canada, 2022, pp. 334-339.

- *[Short Paper]* Z. Heeb, O. Kalinagac, **W. Soussi**, G. Gür: **The Impact of Manufacturer Usage Description (MUD) on IoT Security**; 2022 1st International Conference on 6G Networking (6GNet), Paris, France, 2022, pp. 1-4.

2021

- *[Poster Paper]* M. Christopoulou, **W. Soussi**, G. Xilouris, G. Gür, E.M.d Oca, H. Koumaras: **AI-Enabled Slice Protection Exploiting Moving Target Defense in 6G Networks**; EuCNC 6G Summit Virtual Conference, 6G Vision Poster Session B, (Porto, Portugal) 8-11 June 2021.

Acronyms

A3C	Asynchronous Advantage Actor-Critic
ASP	Attack Success Probability
B5G	Beyond 5G
BR	Bayesian Ridge
CNF	Cloud-native Network Function
CRIU	Checkpoint/Restore in Userspace
CSP	Communication Service Provider
DDoS	Distributed Denial-of-Service
deep-RL	Deep Reinforcement Learning
DML	Decision Making Layer
DQN	Double Q-Network
EL	Enforcement Layer
ENISA	European Network and Information Security Agency
ERLLC/eURLLC	Enhanced Ultra-Reliable Low-Latency Communication
ETSI	European Telecommunications Standards Institute
EU	European Union
EUPG	Expected Utility Policy Gradient
FeMMB	Further-enhanced Mobile Broadband
FL	Federated Learning
FNR	False Negative Rate

FPR False Positive Rate

GDPR European General Data Protection Regulation

GUI Graphical User Interface

HFL Horizontal Federated Learning

HLA High-Level Architecture

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IoT Internet of Vehicles

LBFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno

LiMi Live Migration

MDP Markov Decision Process

MEC Multi-access Edge Computing

ML Machine Learning

MLP Multi-Layer Percetron

MOL Management and Orchestration Layer

MOMDP Multi-objective Markov Decision Process

MTD Moving Target Defense

NBI North-Bound Interface

NFV Network Functions Virtualization

NFV MANO NFV Management and Orchestration

NIC Network Interface Card

NIST National Institute of Standards and Technology

NN Neural Network

NS Network Service

NSi Network Slice

OSS/BSS	Operations Support System and Business Support System
POMDP	Partially-Observable Markov Decision Process
PPO	Proximal Policy Optimization
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RDMA	Remote Direct Memory Access
ReLU	Rectified Linear Unit
REST API	Representational State Transfer Application Programming Interface
RF	Random Forest
RI.AS.	Risk Assessment
RL	Reinforcement Learning
RQ	Research Question
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SotA	state-of-the-art
SVR	Support Vector Regression
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TI	Tactile Internet
TNR	True Negative Rate
TPR	True Positive Rate
UDP	User Datagram Protocol
UE	User Equipement
umMTC	ultra-massive Machine Type Communication

UPF	User Plane Function
VDU	Virtual Deployment Unit
VFL	Vertical Federated Learning
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNO	Virtual Network Operator
VNO	Virtual Network Operator
VoIP	Voice-over-IP
ZSM	Zero touch network & Service Manangement

Glossary

Artificial Intelligence A machine-based system that can, for a given set of human-defined objectives, make predictions, recommendations, or decisions influencing real or virtual environments.

Attack graph A graphical representation of potential attack paths and vulnerabilities in a system, used to analyze and mitigate security risks.

Beyond 5G (B5G) This indicates the telecommunication network technology following that of 5G networks. With the current global initiatives, B5G networks are also denoted as 6G or NextG networks.

Cloudification The process of migrating traditional IT infrastructure, applications, and services to cloud-based platforms to improve scalability, flexibility, and cost-efficiency.

Cognitive security A security approach that uses artificial intelligence and machine learning to detect, analyze, and respond to threats in real-time.

Communication Service Provider An organization that offers communication services such as voice, data, and video to customers, including telecom operators and internet service providers.

Decentralization The distribution of control, authority, or functions away from a central location or organization to multiple distributed entities.

End-to-end communication A complete communication path from the source to the destination, ensuring data is transmitted and received without interruption.

Formal model A mathematical or logical representation of a system or process, used to analyze and verify its behavior and properties.

Framework A structured set of guidelines, tools, and best practices designed to help develop and manage systems or projects effectively.

High Level Architecture A conceptual framework or design that outlines the overall structure and components of a system without detailing its implementation.

Information Security The protection of information and communication systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.

Infrastructure Communication Provider A telecommunication company that offers physical infrastructure, such as RANs, edges, and core networks and servers as a service.

Live Migration The process of moving a running application from one physical host to another without service interruption.

Machine Learning A branch of Artificial Intelligence (AI) that focuses on the development of systems capable of learning from data to perform a task without being explicitly programmed to perform that task. Learning refers to the process of optimizing model parameters through computational techniques such that the model's behaviour is optimized for the training task.

Methodology A systematic approach or set of procedures used to achieve specific goals in a systematic way.

Moving Target Defense The concept of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity, and increase the costs of their probing and attack efforts.

Near real-time A system or process that operates with minimal delay, providing results or responses almost immediately after input or events.

Proactive security A security approach that focuses on preventing threats and vulnerabilities before they occur through planning and preventive measures.

Risk assessment The process of identifying, analyzing, and evaluating potential risks to determine their impact and likelihood.

Solution A software or framework implementing the method designed to solve a specific problem or address a particular need.

Telco Cloud A cloud computing environment specifically designed for telecommunications providers to deliver network functions and services virtually.

Virtual network operator A company that provides network services to customers without owning the underlying network infrastructure, often leasing it from other providers.

List of Algorithms

List of Figures

List of Tables

Curriculum Vitae

Personal Details

Name	Wissem Soussi
Date of Birth	January 16, 1996

Education

February 2021 – September 2025	Doctoral Program at the University of Zurich UZH, Department of Informatics (IfI), Communication Systems Group (CSG), Switzerland
September 2019 – June 2020	Master of Science (MSc), Cybersecurity, University of Grenoble / Grenoble INP, France
March 2018 – June 2019	Master of Science (MSc), Computer Science, University of Grenoble, France
September 2015 – June 2018	Bachelor of Science (BSc), Computer Science, University of Montpellier, France

Professional Experience

September 2025 – Currently	Research Associate at the Information Security Group (ISE), Zurich University of Applied Sciences ZHAW, Switzerland
September 2020 – August 2025	Research and Teaching Assistant at the Information Security Group (ISE), Zurich University of Applied Sciences ZHAW, Switzerland
January 2020 – June 2020	Graduate Student Researcher at Computer Networks Group (Drakkar), Grenoble Informatics Laboratory (LIG), France
January 2019 – July 2019	Undergraduate Student Researcher at Computer Networks Group (Drakkar), Grenoble Informatics Laboratory (LIG), France