Politecnico di Milano
School of Industrial and Information Engineering
Master of Science Degree in Computer Science and Engineering

POLITECNICO
MILANO 1863

Optimal Real-Time Control of a Water Distribution System undergoing cyber-attacks: a Reinforcement Learning approach

Advisor:
Prof. Marcello Restelli

Co-Advisors:
Prof. Stefano Galelli
Dr. Andres Felipe Murillo Piedrahita

Thesis By:
Davide Salaorni
Student ID: 928212

Academic Year 2021/2022

In memory of my beloved grandma.
She would be proud.

Acknowledgments

This work is the result of nine months of research in collaboration with Singapore University of Technology and Design. Despite the pandemic situation which has forbidden my relocation in Singapore, I am really grateful for this experience, especially because I had the opportunity to come in contact with very expert people and with current research fields, that made me feel a real contributor for the improvement of actual real-world problems. I would like to thank all the people who made this work possible, either involved directly or indirectly.

I would like to express my sincere and deep gratitude to Prof. Stefano Galelli for the guidance, the constant presence and all the insights. I would also thank a lot Dr. Andres Felipe Murillo Piedrahita for all the support, the simulations run at dead of night and to make me feel an au pair collaborator. I have appreciated so much that they didn't left me to my own devices, in spite of the distance and the pandemic, giving me continuous feedback and total trust.

I would also like to offer my special thanks to Prof. Marcello Restelli for the great insights and the extreme availability, even though the fate—and the spam detector—has been a bit hostile.

Abstract

In this work, we demonstrate how the introduction of a reinforcement learning agent can be used to control a water distribution systems undergoing cyber-physical attacks. The need for limiting the impact of such digital threats stems from the more and more pronounced integration between physical and digital systems, grown in the last decade, and the birth of, so called, cyber-physical systems. In the field of hydraulic networks, where the handled asset is of critical importance for city dwellers, there is still lot of work to do to accomplish a sufficient level of reliability in the safeguard of digital infrastructures.

The approach we adopt builds on the integration of a deep reinforcement learning model, namely Deep Q-Network, into the Digital HydrAuLic SIMulator (DHALSIM), appointed to simulate the entire cyber-physical system—and thus combining the water network process with the emulation of industrial control systems and corresponding communication protocols. The integration is made feasible thanks to a new feature implemented in DHALSIM, the *stepwise simulation*, allowed by the addition of a new simulator wrapper, namely Epynet. Indeed, this extension permits to run the experiment in a step-by-step manner, giving the possibility, between each frame, to control the actuator variables of the water network. Finally, the reinforcement learning agent is smoothly integrated, since at each step it can analyze the state of the network and output a suitable control action.

The evaluation phase is run considering the system in normal operating conditions and undergoing cyber-attacks. In this work, we report results collected by some restricted experiments, which suggest a good behaviour in terms of Demand-Satisfaction Ratio with small imperfections with respect to the overflow risk. Indeed, it would be interesting to reproduce analyzed experiments for a longer time with more computational power and a different tuning of the hyperparameters. Our intention is to take the first steps in this almost unexplored aspect of hydraulic network and to sketch the guidelines for future extensive studies.

Sommario Esteso

Nell'ultimo decennio, con l'avvento della Quarta Rivoluzione Industriale e del nuovo paradigma lavorativo denominato Industria 4.0, l'interconnessione dei processi produttivi tramite l'utilizzo di sistemi di controllo digitali si è consolidata a tal punto da diventare una realtà per la maggior parte delle medie-grandi imprese. Questa ventata di rinnovamento ha portato numerevoli benefici per quanto riguarda il miglioramento della produttività e l'abbattimento dei costi di magazzino, ma contemporaneamente ha favorito l'insorgenza di vulnerabilità legate all'utilizzo di sistemi digitali e alla gestione di risorse tramite connessioni alla rete internet.

In questo scenario diventa dunque fondamentale preoccuparsi dei rischi che queste interconnessioni possono causare e, allo stesso tempo, cercare di ideare nuove strategie e soluzioni per mettere in sicurezza i canali digitali adibiti alla gestione di risorse industriali. Infatti, non è raro apprendere notizie relative ad attacchi informatici ai danni di specifiche aziende che agiscono in un particolare settore, o addirittura direttamente indirizzati contro la macchina statale. Le falle, in molti casi, derivano proprio da quei dispositivi introdotti con la rivoluzione industriale, che rientrano nel gruppo dei cosiddetti *sistemi di controllo industriale*. La ragione risiede nel fatto che molto spesso i protocolli industriali utilizzati dai macchinari di produzione non implementano sufficienti meccanismi di sicurezza, in quanto non ideati per essere inseriti in un'infrastruttura interconnessa. Inoltre, i tardivi aggiornamenti applicati ai dispositivi di controllo, dovuti alla mancanza di una finestra temporale adeguata, incrementano le debolezze del sistema, concedendo ulteriori vulnerabilità ad agenti malintenzionati.

Poiché l'interconnessione e la creazione di sistemi cyber-fisici si stanno diffondendo in svariati ambiti, dalla produzione energetica al settore manifatturiero, è necessario limitare i possibili rischi con misure di sicurezza adeguate, soprattutto in contesti in cui la risorsa fisica gestita dall'impianto risulta essere un bene primario per l'umanità e il pianeta. Un caso lampante è quello degli impianti idrici, in cui i sistemi di controllo possono migliorare sensibilmente l'efficienza della rete, andando anche a raccogliere informazioni sullo stato dell'infrastruttura e qualità dell'acqua, ma che possono diventare bersagli di attacchi informatici, con ingenti conseguenze su larga scala.

Per ovviare a questo problema, negli ultimi anni ricercatori e industrie stanno studiando meccanismi di sicurezza per rispondere a queste minacce digitali, anche attraverso l'utilizzo di approcci basati sul Machine Learning e il Deep Learning. Le soluzioni trovate finora si basano prevalentemente sul monitoraggio del sistema in tempo reale alla ricerca di anomalie, mediante l'analisi delle letture dei sensori e modelli che descrivono il processo fisico. In questi casi l'intelligenza artificiale entra in campo nel momento in cui il sistema di monitoraggio va allenato con dati verosimili al fine di ricreare un'esperienza il più possibile conforme a quella reale. Parallelamente, gli studiosi hanno cercato di ricreare fedelmente in versione digitale sistemi cyber-fisici, che possano modellizzare oltre al processo fisico di competenza, anche l'infrastruttura di controllo e le conseguenti minacce informatiche a cui potrebbe essere sottoposta.

In questo elaborato di tesi si utilizza un approccio basato sul Reinforcement Learning per allenare un sistema idrico a resistere ad attacchi informatici, incrementando la resilienza della rete a perturbazioni esterne. Il tutto viene simulato attraverso un modello digitale del sistema cyber-fisico, in grado di emulare sia il processo fisico che l'infrastruttura di controllo. Essendo un approccio totalmente innovativo, in quanto cerca di applicare il problema di controllo ottimo in tempo reale—traducibile in un problema di schedulazione degli attuatori—ad uno scenario in cui si verificano attacchi informatici che perturbano lo stato del sistema, l'obiettivo dell'elaborato non è quello di fornire una soluzione algoritmica più accurata possibile, quanto quello di muovere i primi passi verso un nuovo tipo di analisi che potrebbe portare ad interessanti sviluppi e soluzioni future.

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network.

CPS Cyber Physical System.

DDA Demand-Driven Analysis.

DE Differential Evolution.

DHALSIM Digital HydrAuLic SIMulator.

DOS Denial Of Service.

DQL Deep Q-Learning.

DQN Deep Q-Network.

DSR Demand-Satisfaction Ratio.

ICS Industrial Control System.

MDP Markov Decision Process.

MITM Man-In-The-Middle.

PDA Pressure-Driven Analysis.

PLC Programmable Logic Controller.

RL Reinforcement Learning.

SCADA Supervisory Control And Data Acquisition.

WDS Water Distribution System.

Chapter 1

Introduction

In the last decade, the management of physical infrastructures has rapidly evolved. Nowadays, when we talk about physical processes, we do not consider only the physical infrastructure, but, more or less unconsciously, we take also into account all the communication networks, sensors, actuators and computers that monitor and control them. For this reason, to describe this kind of systems, the term *cyber-physical system* (CPS) has been introduced. Thus, a cyber-physical system is a complex system, composed by the traditional physical infrastructure connected to a digital layer, capable of analyzing system information. Along with the physical asset handled by the bare process, another kind of asset has also been emphasized: the digital datum. Indeed, data are able to shape structures, behaviours and conditions of a physical process and allow to analyze it from a mathematical point of view. In a cyber-physical system, data can be retrieved by sensors installed in the system, which can *read* the state of the physical infrastructure and send information to computers, capable of performing data analysis. Then, after the computations, a decision is taken: data are sent back to actuators in order to *write* the current state of the system, if necessary, or keep it as is. Actuators act on the physical system as raw inputs, that is physical actions, like the click of a button or the rotation of a knob.

Sensors, actuators and computational devices populate the group of *industrial control systems* (ICS). The introduction of these components aims to increase performance and efficiency of physical systems and should guarantee security, safety and reliability of processes. All these aspects are part of a modern paradigm that builds on the integration of ICS equipment into production plans and that is generally called *Industry 4.0*—after the presentation of W.Wahlster at Hannover Fair in 2011 [1]. The paradigm envisions some key themes, such as the interconnection between machines, devices, sensors and people, allowed by the *Internet of Things*; the availability of a huge amount of data, which can be analyzed and correlated to ease the decision-making process; the automation of cyber-physical systems, able to make decision on their own and perform tasks autonomously.

Defined as the *Fourth Industrial Revolution*, Industry 4.0 leads to large benefits for the industrial sector. The collection of data and the broad interconnection favor the optimization of the production, providing the synchronization of processes, the reduction of wasted materials and a better scheduling of activities. Moreover, it decreases production costs, since the automation accelerates processes and reduces costs of inventory [2]. In the last few years, in Italy, Industry 4.0 has been bringing huge changes in industrial processes, and companies now want to push into this direction, investing on Cloud Manufacturing, Advanced Automation and Human-Machine Interfaces (HMI). In addition, is promoted also the Industrial Smart Working, which allows workers to work from home, with a huge increment of employees' efficiency and satisfaction. In 2021, the Italian marketplace of Industry 4.0 is worth over 4.5 billions of euros, with an increment of investments between +12-15% with respect to the previous year. With the opportunities offered by the Italian National Plan, in 2020, Industry 4.0 services have reached the value of 275 billions of euros, with an enhance of +8% with respect to 2019. Considering also the particular scenario of COVID-19 pandemic, these data seem incredible and certainly they would not be real without the opening to digital channels and new production approaches [3].

Besides the many advantages, Industry 4.0 suffers from a big problem caused just by its own nature. Indeed, the push of the integration with digital components and the increment of interconnections among devices open up to new vulnerabilities related to the digital infrastructure, that indirectly endanger the physical layer. The boost over modernization of factories should be followed by an appropriate deployment of secure-by-design solutions. However, this is not always the case, as proved by the over 60,000 exposed SCADA devices across the Internet found by A. Mirian et al. in 2015 [4] with simple ZMAP queries. More recently, in 2020, the research conducted by Positive Technologies Security [5] highlighted several threats: 91% of the analyzed organizations can be penetrated by external attackers, which, in 100% of cases, once inside the internal network, can get user credentials and full control over the infrastructure and in 65% of cases can steal sensitive information about partners and company employees. A huge number of vulnerabilities are caused by the ICS equipment. Indeed, according to *Claroty Biannual ICS Risk & Vulnerability Report* [6], a 25% increment in ICS vulnerabilities disclosures with respect to 2019 has been observed, mainly affecting critical infrastructures such as energy sectors, manufacturing, water and wastewater treatment plants. The principal cause of those vulnerabilities are the out-of-date ICS components, since, in order to update the industrial control equipment, it is required a specific maintenance window, which is available only few hours per week—and sometimes even per month. Positive Technologies, during penetration tests, have been able to get ICS equipment connection schema of analyzed companies and connect directly with an internal device communicating with the industrial network.

However, the bright side of the aforementioned statistics is that the increase of disclosed vulnerabilities itself is leading to research focused on detecting and remedying to security flaws. Academia and industry are spending more and more effort to study security measures for ICS. Since working on real world plans is particularly difficult, due to restrictions dictated by industrial secrets and the risk of damaging the infrastructure, researches rely on the development of CPS testbeds to conduct analysis and perform experiment on a plausible network. An example is the WADI testbed, developed to lead researches on secure water distribution systems and test the impact on an attack on the cyber-physical system [7]. Along with testbed solutions, has been developed also digital framework, called *Digital Twins*, able to trustfully reproduce the CPS environment leveraging the implementation of virtual versions of the CPS. Getting back to the field of hydraulic networks, a recent example is the Digital HydrAuLic SIMulator (DHALSIM) [8], which allows to simulate the physical process of a water distribution system and, at the same time, the communication network and protocols between ICS components of the cyber-physical system. Moreover, the design implementation is conceived to replicate several attacks against the digital layer to analyze how the reaction of the system and the consequences on the physical plant.

In this work, we take into account the field of water distribution systems and hydraulic security measures. To do so, we exploit the functionalities of DHALSIM to study the behaviour of a water distribution system undergoing specific cyber-attacks. The research is not meant to be a mere analysis of data collected with reasoned experiments, but it is more focused on the implementation of new mechanisms aimed to enhance the resilience of the system. To reach this goal, we develop a solution based on machine learning techniques, specifically on a reinforcement learning algorithm, which, by means of a trial-and-error approach, is capable to train the behaviour of the cyber-physical system in order to autonomously react in situations of external threats. In other words, we plan to solve an optimal, and real-time, control problem, conceived to find the optimal control action in each step of the simulation. To do so, we provide DHALSIM with a new feature which allows to experiment in a white box environment, where water network information can be collected at each simulation step and control variables can be modified within the same intervals. This feature also fosters the implementation of a real-time control agent, chosen among reinforcement learning approaches, able to act during the development of the experiment and capable to increase its knowledge about the environment with the collection of state samples. The purpose is to make the agent able to distinguish between normal operating conditions of the systems and scenarios in which the water network is affected by cyber-attacks. The study aims to take the first steps towards the integration of control agent with environments undergoing digital threats, since there are many examples of researches assessing pump scheduling problems in normal operating conditions, but very few considering situations of security risks. In addition, we hope that this work could be inspiring for future in-depth studies, bringing to light new relevant approaches, enhancing the security level of ICS equipment in critical environment as water distribution systems.

1.1 Orginal Contributions

We sum up the principal contributions of this work as follow:

1. We integrate the Digital HydrAuLic SIMulator (DHALSIM) with a new Python wrapper, namely Epynet, which allows us to create a new feature for DHALSIM, the *stepwise simulation*, to analyze each experiment frame-by-frame. Moreover, this development also provides the possibility to perform control operations of the actuators during the experiment—something that would not otherwise be possible. The addition also enhanced the performances of the framework in term of time complexity with respect to the wrapper already present, called WNTR.

2. We implement and integrate a reinforcement learning algorithm with DHALSIM, making the framework able to assess the optimal real-time control problem. The algorithm that has been implemented is the famous Deep Q-Network (DQN), structured to fit the cyber-physical system of a water distribution network and organized in a portable way, in order to adapt to future changes in the framework.

3. We tested the aforementioned reinforcement learning algorithm with the hydraulic network both in normal operating conditions and under attack conditions. We studied the obtained results and evaluated the benefits that an AI agent can bring into a cyber-physical system.

1.2 Thesis Organization

In the seven chapters of this thesis, the contents are organized as follows. In Chapter 2, we explain the motivation behind this work, presenting the state-of-the-art to set the starting point of our discussion and summarizing goals and challenges related to the research. In Chapter 3, we provide the basic knowledge to understand this work: first, we explain the behaviour of water distribution networks and industrial control systems; then, we present the framework and *DHALSIM*; finally, we give some notions about Reinforcement Learning and Deep Q-Network. Chapter 4 is devoted to explaining how we merge together the concepts presented in the previous chapter. We show the system model on which we work, how we extend DHALSIM with a new important feature, how we conceive the attacks, and present the formalization of the control problem in the analyzed environment. In Chapter 5, we provide details regarding the experimental setup, specifically explaining how we integrate *Epynet* and DQN algorithm into DHALSIM. Chapter 6 is earmarked to the analysis of results and to the comparison of the different studied scenarios. Finally, in Chapter 7 we sum up the results in the conclusions, describing limitations of the work and providing additional ideas for future works in the field.

Chapter 2

Motivation

Over the past few years, it has become clearer and clearer that industrial control systems can enhance monitoring, automation and management of tasks related to different fields, even increasing their performance and efficiency. At the same time, in parallel with the many advantages, have started to be known also drawbacks and risks, which can get significant with the handling of critical assets, regarding the production of goods or the safety of human beings. The main threat comes from the weakness of industrial control systems against cyber-attacks [9] [10] [11], which can truly compromise the behavior and reliability of an entire cyber-physical system. In the case of this thesis, the at-risk subject is a water distribution system, and, indirectly, also the group of users relying on that network to be supplied with water.

Also in hydraulic system literature, we can find articles assessing risks [12] [13] and evaluating countermeasures [14] and have been even created challenges to compare performances of different attack detecting frameworks, as the BATADAL competition [15]. Design a detecting algorithm is definitely not a trivial task: defenders have to foresee proactively all the actions that an attacker would attempt to compromise the system, making assumptions that must hold in real-world scenarios.

In this work, we focus on a different aspect of the defensive side: rather than build another detection framework, we try to investigate whether it is possible to make the cyber-physical system resilient to cyber-attacks, training the water distribution network to distinguish between its normal operating conditions against compromised ones. This is not meant to replace detection algorithms; instead, the idea is to complement the information provided by detection algorithms, so as to provide the system with more information about its current state and helping to choose robust reactions.

However, so far, not much as been done concerning this aspect, thus we have to set the fundamentals of this research topic relying on a framework, a so called *Digital*

Twin, able enough to reproduce the environment of a water distribution network (WDS) and the interactions within, on top of which exploiting reinforcement learning concepts to control the system undergoing cyber-attacks.

2.1 State of the art

Here, we discuss important work in the the area of cyber-attacks, introducing also machine learning techniques applied to cyber-physical systems (CPS). Often these concepts will met each other, for example in anomaly detection frameworks, where machine learning is exploited in the classification task of known signatures. However, there are no studies on how to tackle the control problem of a water distribution system undergoing cyber attacks.

2.1.1 Digital Twins of WDS

Replicating a small network of a real process in lab could be a arguably easy task and it could generate a physical model interesting to study, however quite limited by physical restrictions, like dimensions or feasible use cases. That's why, sometimes, we need a Digital Twin: a way more complex model, simulated by a computer, which can explore drastic scenarios, without consequences on the real-world, and even in a faster way. Also in the hydraulic field, researchers have thought to create Digital Twins to simulate bigger water distribution network and emulate cyber-physical systems. The true challenge in simulating a CPS is the merging of its three fundamental components: the physical system, the ICS equipment and the industrial communication network.

Related to WDS, we saw a first implementation of a Digital Twin with EpanetCPA [16], a framework with the objective to characterize effects of cyber-physical attacks (CPAs) on the hydraulic behavior of a water distribution network. Taormina et al. [16] exploited the EPANET simulator to represent the physical system, improved in the second release with pressure-driven feature [17], and Matlab to simulate cyber-physical components. However, the absence of the layer emulating the communication network prevents the tool from being a completely accurate Digital Twin. A year later, a new attempt has been made by Nikolopoulos et al. with RISKNOUGHT [18], which improved the cyber layer description of EpanetCPA, but with protocol and network data simulation only restricted to ACK signals. Both EpanetCPA and RISKNOUGHT, becuase of their design, are not able to produce coherent network traffic data and emulate realistic attacks on the communication layer, rather only their impact on the physical system.

The first Digital Twin deserving this definition is the Digital HydrAuLic SIMulator (DHALSIM) [8]. In this framework, Murillo et al. have been able to integrate the WNTR hydraulic simulator [19], used to reproduce the physical system, with MiniCPS [20], an industrial network emulator. WNTR and MiniCPS are integrated in a co-simulation environment. DHALSIM can reproduce the hydraulic processes of a WDS as well as the full stack emulation of common industrial control protocols, allowing the possibility to replicate cyber-attacks both from communication layer and physical layer perspective.

In this work, we start from this version of DHALSIM to implement an external agent able to tackle the control problem in presence of cyber-attacks, after integrating the framework with another more appropriate Python wrapper of EPANET simulator, called Epynet [21].

2.1.2 Anomaly Detection in WDS

A good starting point to understand the state-of-the-art concerning detection algorithms is surely the BATtle of the Attack Detection ALgorithms (*BATADAL*) [15]. During this competition, seven different frameworks have been presented and evaluated in terms of time-to-detection and classification accuracy. Here, it is already evident the advantage that machine learning and neural networks can bring: e.g., Abokifa et al. [22] exploited a multilayer perceptron to detect unusual patterns that differ from normal operating conditions, obtaining one of the first positions; Brentan et al. [23] adopted recurrent neural network to predict tank water levels from pump flow, upstream pressure and hour of the day; Chandy et al. [24] made use of a convolutional variational autoencoder that computes the probability to be in presence of anomalies.

More recently, Addeen et al. [25] summarize in an accurate survey common cyber-physical attacks and their relative detection mechanisms for water distribution systems. From presented works, we can draw the improved methodology of aforementioned Abokifa et al. [26] consisting in four modules, with the task of checking control rules of actuators, monitoring variables bounds through a statistical method, training an Artificial Neural Network model (ANN) to predict anomalies and detecting those last among the whole dataset with the Principal Analysis Component (PCA). A further contribution comes from Taormina and Galelli [27] with an algorithm designed for detecting and even localizing cyber-attacks against water distribution systems, which exploits a deep learning neural network architecture trained on data pertaining to normal operating conditions. Newer studies, like the one conducted by Kadosh et al. [28], bring back the attention on the physical understanding of the WDS topology, demonstrating that also a trivial one-class classification algorithm like a Support Vector Data Descriptor (SVDD) can compete with more complex architectures if provided with tailored features.

Despite these many advances, there is still work to do: the number of studied cyber-attacks is limited and novel ones are necessary to enrich the possible scenarios, existing detection methods are not optimal and most of them cannot intercept all the attacks, and the security of network and communication layer has to be improved to assure reliable components of WDS.

2.1.3 Control problem in WDS

In WDS field, the control problem is often seen as a pump scheduling problem, where the main goal consists in meeting the water demand while minimizing the amount of energy consumed by the pumps—and, thus, the utilization costs. Common methods exploited in literature include deterministic optimization algorithm, as Linear, Dynamic and Nonlinear Programming, and metaheuristics, like differential evolution (DE), particle swarm optimization (PSO) and other genetic algorithms [29].

Hajgato et al. [30], driven by too high computational burden implied by aforementioned methods, tried to tackle the real-time optimization problem through Reinforcement Learning techniques, in particular by a Dueling Deep-Q Network, which works online. Rather than the status, as happens in pump scheduling problems, the authors focused on predicting the pump speeds, sampling the state of the WDS in discrete intervals of time during a simulation and evaluating the best action which maximizes the modeled objective function.

In this work, we take a further step: we start conceptualizing the control problem as authors in [30], but, instead of working with normal operating conditions, we introduce the cyber-physical layer with relatives protocols and attacks, thanks to DHALSIM frameowork. Thus, we conceive the problem as a real-time pump scheduling of a WDS undergoing cyber-attacks.

2.2 Goals and Challenges

The main goal of this work is to bridge the gap present in hydraulic network literature regarding the absence of a method that allows to a water distribution system to autonomously react to cyber-attacks, avoiding the disrupting of the normal operations and improving the resilience of the cyber-physical system.

Indeed, if on one hand there are already some algorithms employed to detect several kind of attacks, as shown in Section 2.1.2, and, on the other, we can enlist articles assessing the pump scheduling problem with an offline approach and only in one case with a RL algorithm, as explained in Section2.1.3, there is still the need to bridge the gap in between, tackling the optimal control problem of network actuators also in a situation of cyber-risk. In this work, we want to set the basis of this new approach, but to reach this goal we need to make some intermediate steps.

A first objective, achieved in this thesis, is the integration with DHALSIM of a new Python wrapper of EPANET [31], to simulate the physical system of the water distribution network. The wrapper, a Python library conceived by employees of *Vitens*, the largest water drinking company in The Netherlands, is called *EPYNET* [21]. This integration is not only a preliminary step towards the realization of the aforementioned goal, but also an important addition to DHALSIM, which gains a new relevant feature: the *stepwise simulation*. This feature, beyond the possibility to perform a frame-by-frame analysis, permits also the changing of some actuator parameters in the middle of the experiment, identically to a real-time controlled system. Thus, DHALSIM acquires a new peculiarity, which make it suitable for more deepened experiments, that, for example, could require the injection of control inputs in the network, stressing a precise frame of the simulation. Besides the acquisition of a new functionality, *Epynet* allows DHALSIM to get better performances in term of time complexity with respect to WNTR, thanks to its lightweight structure.

As previously said, implementing the stepwise simulation opens up to a real-time approach. Here is our second objective: the integration with an online control agent. Indeed, the capability to actively interact with the network offers several sparks related to machine learning field, especially regarding reinforcement learning. The principal challenge here, is to find a suitable algorithm which can smoothly fit with the environment of a water distribution network and its communication layer. Also, it seems even more interesting to study how the system react in case of cyber-security threats. Thus, we can sum up the second goal of this work in the modeling a reinforcement learning agent able to tackle the optimal control real-time problem of a cyber-physical system undergoing cyber-attacks.

Finally, the last objective of the thesis is to evaluate the effectiveness of the obtained control agent on a threatened water distribution system. The idea is to compare the response of the network undergoing cyber-attacks with normal operating conditions, to see if the learning process can make the CPS more robust and resilient. This can be done by studying the trend of tank levels and pump status nearby attacks, defined a suitable objective function depending on the network expected behaviour. This goal has been achieved only in part, since we don't have the resources to perform a complete and exhaustive analysis, both from computational and timing point of view. Moreover, beyond the agent's performances, which can be considered of questionable relevance, we aim to sketch the path to a new approach for the WDS optimal real-time control problem, enriched by the presence of external concealed cyber-attacks.

Chapter 3

Background

In this chapter, we give background information about the different domains of knowledge on which this work builds. We start by defining a water distribution system, both as a physical process and as part of a cyber-physical system. Then, we analyze the components of a CPS and their interactions, up to the detailed explanation of the Digital HydrAuLic SIMulator (DHALSIM). In the end, we present the Reinforcement Learning approach and the algorithm we exploited to tackle the control problem.

3.1 Water Distribution System

To describe the key processes occurring within a water distribution system we refer to Heasted Methods' et al. textbook [32]. A water distribution system is a physical structure with the basic purpose of delivering water from the source to the customers—which can be simply homeowners, restaurants, factories and so on. Customers determine the amount of water utilization, that can vary over time, both seasonally and daily, and over space. Thus, is important to have a good knowledge of the network topology to identify how water is distributed across the system for an accurate modeling.

Typically, a modern water network is made up of two classes of components: *nodes* and *links*. In the first group we can find:

- *reservoirs*, providing water to the system;
- *tanks*, employed to store excess water within the system and release it at times of high usage;
- *junctions*, used to remove or add water from/to the system.

Instead, between the links we can list:

- *pipes*, conveying water from one node to another;
- *pumps*, appointed to increase the hydraulic level to overcome elevation differences and friction losses;
- *valves*, exploited to control flow and pressure of the system depending on predetermined rules.

All the previously enlisted components are essential to fulfill the task of water networks, which is really important, since the handled asset is of vital relevance for the humankind. Concretely, a water network has to assure the maintenance of a certain level of pressure inside the system, to allow the water to flow without stagnation or flooding from the source to the customer. The criticality of water distribution networks has enlightened the need of preventing and controlling in advance risky scenarios and pushed researchers to emulate such situations with credible simulations.

With the term simulation, we refer to the process of exploiting a mathematical representation to reproduce the real system: that's the so called *model.* The model, together with the digital representation of aforementioned components, allows us to shape the *simulator*. The importance of a simulator is rather evident: with a simulator we can prevent system responses under particular conditions without disrupting the real network; we can face problems in advance and evaluates risks, costs and times without investing in real-world projects. Today we can achieve this task thanks to digital simulators, a clear example of which is EPANET [31]. EPANET is a modern simulator, provided also with a graphical user interface (GUI), capable of emulating the behavior of a water distribution system, regardless of its dimension and topology, both from a quantitative and qualitative point of view. Indeed, Epanet can calculate, for example, the water level in a tank after a period of one month, evaluating at the same time its water quality.

Besides these several advantages, there are some limitations, especially related to the way in which, nowadays, a water network is integrated with technologies, becoming an out-and-out cyber-physical system (CPS). Indeed, this new configuration brings researchers to consider not only the physical system, but also the surrounding cyber-infrastructure, creating the need to enrich simulators like EPANET with functionalities that can emulate the digital layer, in particular ICS equipment and industrial communication network.

3.2 Industrial Control Systems

ICS are network infrastructure designed to guarantee that a physical process operates at all times based on a set of defined operational parameters, as explained by Humayed et al. [33]. Within the context of WDS, ICS monitor the hydraulic conditions (e.g., tank water levels) and apply a predefined control logic to hydraulic actuators, like pumps and valves, to maintain across the whole system the desired levels of pressure. Depending on the specific design and deployed technology, the ICS may also monitor water quality parameters (e.g. pH or chlorine concentration) and control additional components, such as booster pumps. Galloway et al. coin for these communication networks used in ICS the name of Industrial Control Networks [34].

In Figures 3.1 and 3.2, we illustrate a simple WDS and its corresponding ICS. In this system, the physical network is made up of a tank (T1) and a pump (P1), which controls tank water level. Moreover, we have a structure composed by two PLCs (PLC1 and PLC2), one SCADA server and the network, that is the ICS. Let's now see how the two entities work together. To start, PLC1 uses a sensor to measure the water level of T1; then, it sends this reading to PLC2, which uses the acquired knowledge to apply the corresponding control rule (e.g. to decide whether P1 needs to be turned on or off). This cycle is known as a *scan cycle* and it is executed periodically in order to maintain the tank level within the desired operational parameters. In addition, the PLCs report to the SCADA server the values of different variables, such as T1 level, P1 status, P1 flow or pressure at the junctions. The information exchanged through these scan cycles travels through the network using industrial control protocols, which will be described next. In addition, note that each PLC is located in a different substation. This means that each PLC is located within a Local Area Network (LAN) and both networks are connected using a Wide Area Network (WAN), represented in Figure 3.2 by r0. Considering that WDS are typically distributed across vast spatial domains, locating the PLCs in different substations is therefore a compelled network configuration.

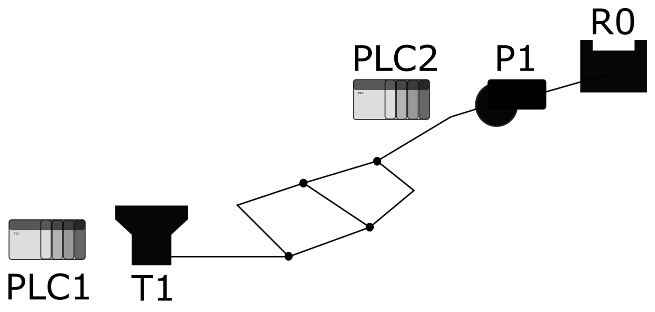

Figure 3.1: Simple water distribution system. Focus on physical layer, where two PLCs control the water level of Tank T1.

Network communications are logically divided into layers. Each layer has

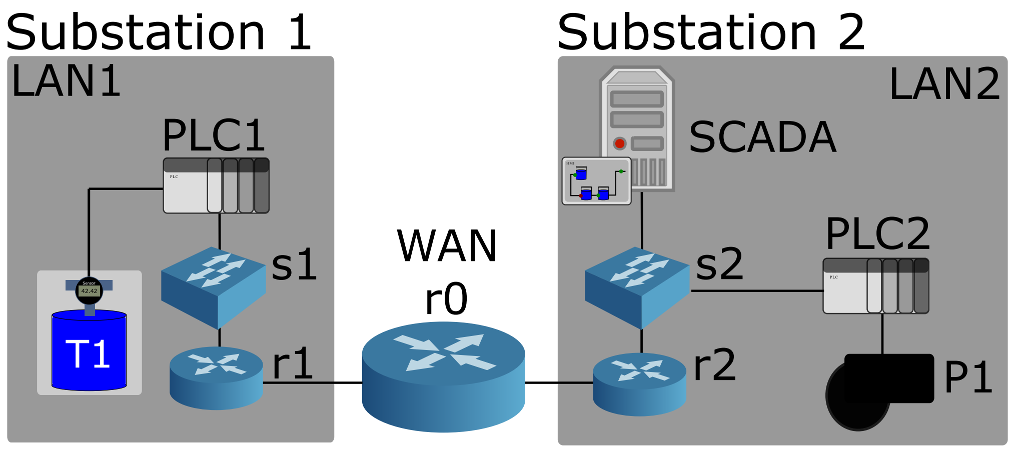

Figure 3.2: Cyber physical layer of the previous water distribution network. In the cyber layer, due to the geographical distance between the elements, the two PLCs are located in its own substation. A set of routers and switches connects the PLCs.

certain functionalities that can be offered by a specific protocol [35]. In this way, protocols are designed to operate at a specific layer; for example, IEEE 802.11b/g/n (commonly known as "WiFi) is a network link layer protocol. This also means that stack of protocols needs to be defined to provide network applications. The term *stack* is used because protocols at lower layers offer services to the upper layer and the uppermost one is directly offering the final application. In the internet case of web applications, the HTTP protocol is the protocol used by browsers to offer the final content to the users. But at the same time, the HTTP protocol is supported by the combination of TCP/IP protocols. Finally, in a wireless local area network (WLAN), these TCP/IP protocols can be supported by IEEE 802.11.

Figure 3.3 shows an example of the stack of protocols. In the uppermost slot we have CIP, an industrial protocol used for communications between PLCs, which can exchange messages containing sensor readings or actuator status. CIP is supported by another protocol called "Ethernet over IP" (ENIP) and together are placed on top of TCP/IP. Notice that although the upper layer protocols are industrial protocols, the bottom protocols are the same TCP/IP used in traditional internet applications.

The stack of protocols represented in Figure 3.3 is used to monitor and control a system like the one shown in Figure 3.2. This monitoring-controlling process is performed in something called *scan cycles*. In the example, PLC2 requests PLC1 the level of T1. After receiving it, PLC2 looks up the control rules configured for P1 and then applies the appropriate decision (on whether to turn off or on the pump). Nevertheless, the process happening at network level is more complex. Indeed, at

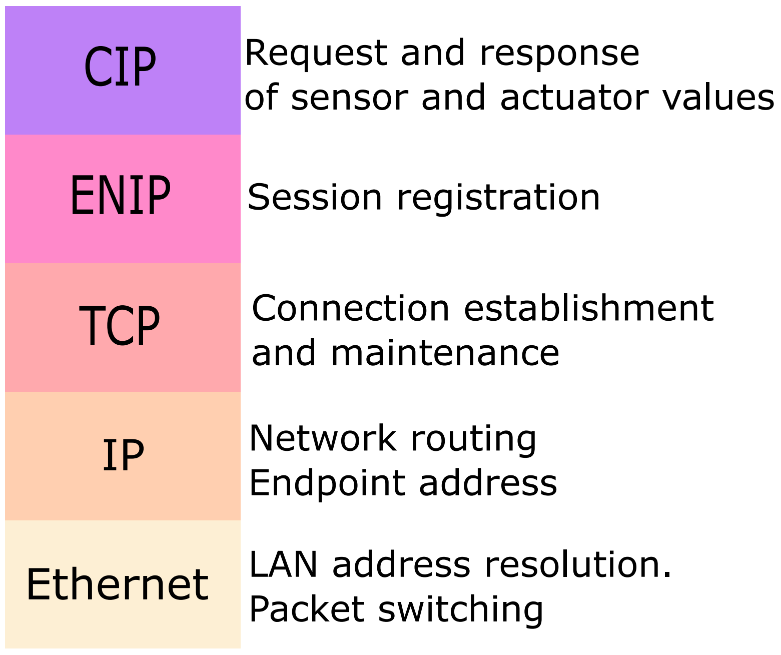
CIP
Request and response of sensor and actuator values
ENIP
Session registration
TCP
Connection establishment and maintenance
IP
Network routing Endpoint address
Ethernet
LAN address resolution. Packet switching

Figure 3.3: Protocols stack.

network level, to exchange a CIP message have to be performed the following steps: 1) local address resolution (ARP), 2) TCP connection establishment, 3) ENIP session registration, 4) CIP request and response.

1. the first stage (*local address resolution*) has to be executed only at the first cycle. At the beginning of a scan cycle, PLC2 does not have a route to send messages to PLC1, but PLC2 has configured router r2 as its gateway. This means that PLC2 knows the IP address of r2. To access its gateway, PLC2 sends a broadcast message to s2 asking for the local (MAC) address of r2. Broadcast messages are sent through all the interfaces of the switch. This causes s2 to flood the substation 2 network with messages asking for the MAC address of r2. This is done using a protocol known as Address Resolution Protocol (ARP). After the PLC2 obtains the MAC address of r2, the following stages can take place. Note that this MAC address is stored in the PLC2 cache and used to establish future connections.

2. The second stage (*TCP connection establishment*) starts when PLC2 sends a *TCP SYN (TCP-SYN)* message to the IP address of PLC1. This message is sent through r2, using the MAC address resolved in the previous stage. The router looks up in its routing table for a route to access the PLC1 IP address. r2 knows that PLC1 is reachable through r1 and forwards this message to him,

through r0. Upon reception of this message, r1 sends the message to s1. The message finally arrives to PLC1. Upon reception of the *TCP-SYN* message, PLC1 replies with an *TCP SYN Acknowledgement (TCP-SYN-ACK)* to PLC2. This last message uses the same routes and path used before. Finally, PLC2 replies with an *Acknowledgement (ACK)* message and, when PLC1 receives it, the TCP connection is established. This mechanism is known as 'Three Way Handshake' [35].

3. The third stage (*ENIP session registration*) starts once a TCP connection has been established. This connection is used by PLC2 and PLC1 to register an ENIP session, which allows to signal both PLCs that an exchange of CIP messages is about to take place. In addition, an ENIP session can be used to exchange a set of sensor readings or actuators status (commonly called *tags*) between the PLCs. To register an ENIP session, the PLCs exchange a *ENIP Register Session Request* and *ENIP Register Session Response* message.

4. Finally, with the ENIP session registered, the last stage can take place. In this stage, PLC2 sends a CIP message requesting the value of the tag T1 (tank 1), and PLC1 replies to this message with a CIP response message and the T1 value. Upon reception, PLC2 closes the TCP connection. Importantly, the second to fourth stages are repeated at each scan cycle.

As shown above, it is necessary to follow multiple procedures, sequences of messages and group of nodes to exchange information between PLC2 and PLC1. Many factors could impact these components and therefore affect the entire communication protocol, ultimately compromising hydraulic processes. Reproducing these interactions in simulation models is nowadays very important in any cyber-security analysis, such those required for modern WDS.

This has been the main reason behind the development of the Digital Hydraulic SIMulator. In fact, DHALSIM leverages MiniCPS to accurately represent the aforementioned network information. This information enables DHALSIM to be used as a platform to run cyber security experiments with high fidelity. In these experiments, DHALSIM can launch two types of anomalies: *attacks* and *events*. Attacks are actions initiated by an attacker with the intention of impacting in a harmful way the network and physical system. Attacks in DHALSIM are divided in two main types: device attacks, launched by the PLCs, and network attacks, launched by additional network nodes. Network events are situations that might impact the network and physical system in a negative way, but are not launched by attackers. They could be seen as circumstances that arise from faulty or unexpected conditions in the system. An example of a network event is a loss of connectivity in a network link, that could lead to all the packets being dropped in that link.

3.3 Digital Hydraulic SIMulator

This section provides an overview of DHALSIM, its architecture and design, implementation details, key functionalities and limitations. Moreover, we show some details about how to configure DHALSIM experiments with device attacks, network attacks or network events.

3.3.1 Software architecture

In Figure 3.4 we can examine the complex architecture of DHALSIM. The principal components are the physical simulation, provided by EPANET simulator [31], and the network emulation tools, i.e. MiniCPS [20] and Mininet [36].

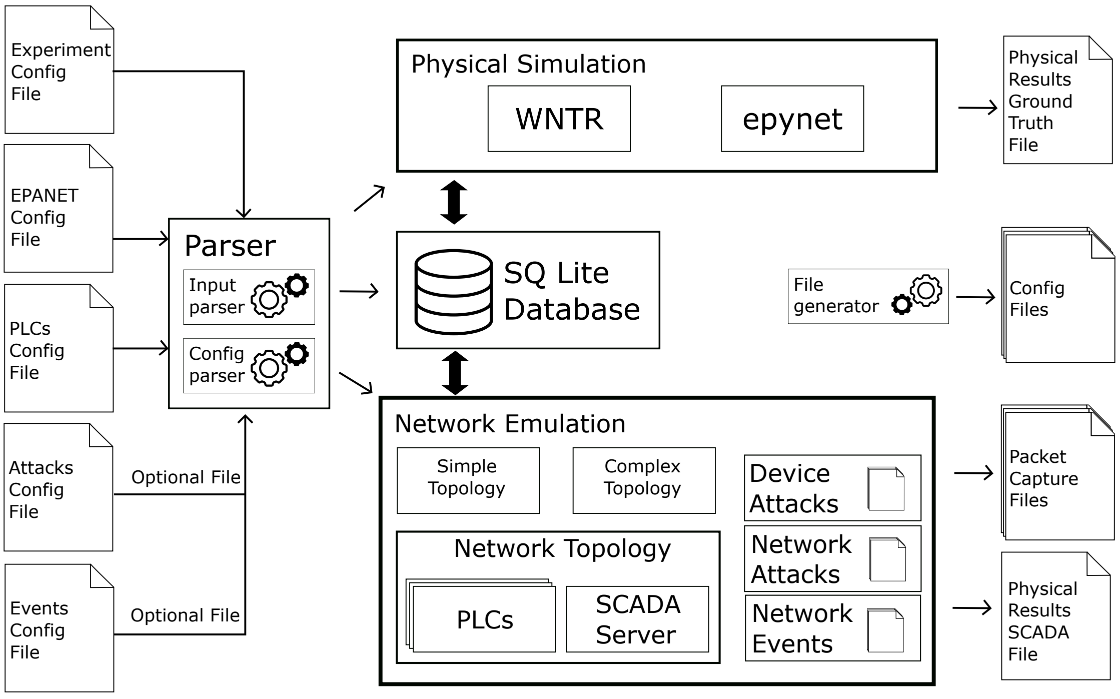

Figure 3.4: DHALSIM Architecture. The architecture is composed of a parser, physical emulation, network emulation, an SQLite Database, and a File generator. The parser converts EPANET .inp files into DHALSIM topologies.

Mininet is a platform to easily create virtual networks and nodes, which run inside a single machine host. Each Mininet node has its own virtual network interfaces and can run any software installed in the host machine.

MiniCPS provides an implementation of two popular ICS communication protocols, ENIP/CIP and Modbus (here, we use ENIP/CIP, since it's the most common protocol for PLC-PLC communication). By means of MiniCPS, Mininet nodes can communicate using these industrial protocols. In this sense, DHALSIM creates a virtual network, where nodes representing PLCs and SCADA communicate using ICS protocols. Additionally, a different process runs the EPANET simulation and many others can be launched to create network attacks or other types of events (e.g., dropping a percentage of packets at a network link during a defined number of iterations, a Denial of Service attack at a network link).

A DHALSIM experiment runs in the following way: first, the configuration files are created—more related details will be presented below. Second, the Parser reads these configuration files, initializes a Mininet network and launches a process to run the physical simulation. The Mininet nodes, which represent PLCs and SCADA, reproduce the behavior of ICS equipment, while the physical simulation runs in a step-by-step basis and uses a master clock to synchronize the concurrent processes running in the experiment. During the simulation, different events or attacks can be launched. Finally, after being run for the configured duration, all the processes receive a signal that starts their shutdown routines, at the end of which the output files are stored into a folder.

3.3.2 Design

Five main components mark DHALSIM out: a Parser, a Physical Simulation engine, a Network Emulation engine, a SQLite Database, and a File generator. This subsection explains their technical details and how they interact in a DHALSIM experiment.

The Parser

The DHALSIM Parser is a module that reads configuration files to launch a DHALSIM experiment. All the configuration files use YAML (*YAML Ain't Markup Language*)[1]. The following configuration files are processed by the parser:

- Experiment Config File: the experiment configuration file defines the global configuration options for a DHALSIM experiment. This file specifies the EPANET .inp file, the number of hydraulic time step iterations the experiment will run for, the path to additional configuration files and the type of hydraulic simulator being used—initially only WNTR, but then also *Epynet*, introduced by thist thesis work. In addition, custom demand patterns and initial tank levels can be configured . Finally, this file also defines the type of network topology used, i.e., *Simple* or *Complex*. Details of these topologies are explained below.

[1]YAML is a is a human-readable data-serialization language. https://yaml.org/

- EPANET Config File: this is an EPANET .inp ‘standard’ file. DHALSIM automatically parses .inp files in order to build an appropriate Mininet topology and uses the control rules defined in the *[CONTROLS]* section to create the PLC control logic.

- PLCs Config File: this is a configuration file that indicates how many PLCs the WDS has and which PLC handles which sensor or actuator. Sensors in DHALSIM are tank levels, junction pressures, and pumps/valves flows. A sensor is a value read from a PLC and used to apply a control rule and all sensor information are polled by the SCADA server. Actuators in DHALSIM are pumps and valves. The PLCs implement the control logic defined in the .inp *[CONTROLS]* section to change the status of the actuators. Also the status of actuators is polled by the SCADA server.

These three configuration files are sufficient to run a DHALSIM experiment without attacks or events. If attacks or network events are going to be launched during a simulation, the following optional files must be provided:

- Attacks Config file: attacks are configured in this optional file. Two types of attacks are configured in DHALSIM: device attacks and network attacks, both providing *triggers* that can be used to easily set conditions that launch the attack. Triggers can be the simulation iteration or values of tags in the physical system. For example, a time trigger could activate a man in the middle attack when the simulation reaches the configured number of iterations. Device attacks are attacks running in PLC processes that can change the way a PLC applies a control logic. Network attacks launch an additional Mininet node running a script that exploits a network vulnerability and affects the network and its physical behavior. These network attacks are activated by triggers. Attack scripts that implement Denial of Service attacks and Man in the Middle attacks are provided in DHALSIM.

- Events Config file: events are configured in this optional file. Currently, only network events are supported. Network events are events that affect the way a network link behaves. An example of an event would be one that causes a percentage of packets in a network link to be dropped.

By means of these files, the Parser decides which Mininet network topology should be created, counts how many PLCs are present in the network, configures the network parameters of those PLCs, adds the SCADA server and launches a physical simulation.

The Physical Simulation

The physical simulation module runs an EPANET simulation. Before the completion of our work, only one EPANET wrapper was offered with DHALSIM: WNTR [19], a Python package wrapper for EPANET [31]. DHALSIM runs a WNTR simulation in way that requires a slight modification of the way control rules are applied in a simulation. This is required to let MiniCPS PLCs to control the actuator status of the simulation. In WNTR, a water network is represented through a network model object. Also, network model objects create control instances for each control rule configured in an .inp file. At each simulation step, DHALSIM constantly eliminates and creates new control instances in order to dynamically update the actuator status in the simulation. This is done because, at the moment, WNTR does not offer a dynamic way of updating the actuators status. Besides this, WNTR does not provide a way of running step-by-step simulations. Such simulations are a necessary condition for MiniCPS to affect the system state and to enable experiments where attacks affect the physical state of the system. As a workaround, DHALSIM configures the WNTR simulation duration to be equal to an hydraulic time step and runs a number of WNTR simulations equal to the number of iterations configured in the Experiment Config File. Naturally, these limitations have an impact on the computational requirements of DHALSIM experiments.

A simulation step into DHALSIM runs in the following way: First, the physical simulation (called the plant) reads the actuator status stored into the SQLite Database. Second, the plant updates the water network object to reflect the new actuator status. Third, the plant runs the next simulation iteration. Fourth, the plant stores into the SQLite Database the new values of the system state. The system state are the values of tank levels, pump/valves flows, and junctions pressures. In addition, these values are stored into a data structure that is stored into the *ground truth file* at the end of the simulation. Fifth, these values are read by the PLCs in the Mininet network and the values are exchanged to implement the control logic. The latter is implemented by the PLCs by taking control decisions and storing the new actuator status back into the SQLite database. Then, the whole cycle repeats. Once the simulation has run the configured number of iterations, the simulation process finishes. This triggers all other processes to finish, concluding the experiment.

The SQLite Database

The SQLite Database is the communication point between the Physical Simulation and the Network Emulation. This approach has been chosen to synchronize the concurrent processes because it does not generate additional communication traffic that could introduce artifacts into a DHALSIM experiment. In addition, using a centralised approach there is the advantage of having special registries in the database to synchronize events in an experiment. Such registries are the simulation master clock, used to trigger some attacks or events, and *sync* flags, used by PLCs and the physical simulation to synchronize their state and control actions. The SQLite Database is a stateless database, meaning that only the current values of the simulation are stored. Previous values are recorded separately by the SCADA server and physical simulation, respectively.

The Network Emulation

The network emulation uses MiniCPS to launch a Mininet network in which Mininet nodes run Python scripts representing the PLC and SCADA behaviour. A PLC behaviour is composed of a *pre-loop* and *main loop*. The pre loop is used by the PLC to initialize its variables and configure attacks that are going to be run in the PLC. Additionally, the pre loop launches a *tcpdump* capture process (TCP dump is a Linux networking tool used to capture a copy of the packets in a network interface [2]). Each PLC process runs a *tcpdump* subprocess at the beginning of the simulation. This provides network information during a DHALSIM experiment. Generating these *tcpdump* capture files is possible due to the complete network stack being implemented by Mininet and MiniCPS and is one of the key features of DHALSIM.

The main loop performs the following operations:

- Update local cache: Each PLC has a local cache where it stores the variables necessary for its behaviour. These variables are configured in the PLCs Config file and in the *[CONTROLS]* section of the .inp file described in Section 3.3.2. There are two ways to obtain these variables: independent variables and dependent variables. If a PLC is directly connected to a variable, such variable is called an independent variable. An independent variable is a variable that is defined inside a PLC section of the configuration file. For these variables, the PLC performs a *get* operation into the SQLite Database. This reflects a PLC that is physically connected to a sensor. Dependant variables are variables not under the control of a specific PLC (defined into another PLC section in the Config file), but still necessary to apply a control rule. For these variables, the PLC will perform a *receive* operation. The receive operation triggers the CIP/ENIP network process to receive a tag through the network described in Section **??**.

- Apply a control rule. If a PLC controls an actuator (as defined in the PLC Config File) and that actuator has a control rule associated to it, the PLC

[2]TCP dump is a command-line packet analyzer: https://www.tcpdump.org/

uses the sensor values to apply the control rule. DHALSIM automatically parses the control rules defined in the *[CONTROLS]* section of an .inp file. Nevertheless, the new actuator status is stored in the local cache, instead of the SQLite Database.

- Apply device attack actions. Device attacks run on PLCs and could affect the way a PLC operates a particular actuator. For example, a device attack could order a PLC to maintain a pump open during a certain number of simulation iterations. If this is the case, the PLC overwrites the actuator status stored in the local cache.
- Store actuator values into the SQLite Database. The PLC stores into the SQLite Database all the actuator status under its control.
- Send the variables under its control to other PLCs and the SCADA server. The values stored in the PLC cache are made available in the network for other PLCs or the SCADA server to request. This process runs in a separate sub thread of each PLC process.

The other type of Mininet node is the SCADA server. A SCADA server is a special node in Mininet that does not implement any control logic, but periodically polls the PLCs in the network for the system state. Then, it stores those values into a data structure that is written into the *scada values* file at the end of a DHALSIM experiment. The inclusion of a SCADA server is useful in cyber security experiments, because some attacks could include concealing techniques to mask the impact of a cyber-physical attack [16]. Finally, if network attacks or events are configured, additional Mininet nodes are launched. In such case, each node would run a script for the configured network attacks or events.

All nodes are connected using a Mininet network. Recall that this network can be configured in two ways, Simple or Complex topology. In a simple topology, all nodes are in the same Local Area Network, meaning that no routing is necessary. In a complex topology, each PLC and SCADA server is in its own local area network and network routing is used to interconnect them. Such topology should be adopted when working on WDS spanning across large spatial domains.

3.3.3 Implementation

DHALSIM is a Python Open Source Software with MIT License and available as a Github repository at `https://github.com/afmurillo/WadiTwin`. DHALSIM has been developed and tested using Ubuntu 20.04 and requires Mininet, MiniCPS and WNTR. An automatic installation script is provided in the repository. DHALSIM uses Python 2.7 and Python 3.6 to run some of its code. The need for Python 2.7

stems from Mininet and MiniCPS, which do not yet offer full support for Python 3.6. With the thesis work, DHALSIM acquires a new simulator wrapper, Epynet [21], in addition to WNTR, to perform a stepwise simulation, and a RL control agent which exploits MushroomRL library [37].

Launching an experiment with DHALSIM ends up to get results split in two types of files, equiped with timestamps to allow cross-validation: physical results files and network files. The physical results files are two .csv tables containing the results of the physical simulation (EPANET): ground truth and SCADA values. The ground truth collects the real values generated by the physical simulation, whereas the SCADA values contains the values that the SCADA server polled form the PLCs in the network. Notice that if an attacker launches a concealment attack like those shown in [16], the values from SCADA and ground truth may differ. The network files are the *tcpdump* capture files, which have the extension .pcap. The capture files have all the network messages seen by a Mininet node during the experiment. This provides researchers with all the network packets that a DHALSIM experiment generates during its execution.

3.4 Reinforcement Learning

in the context of Artificial Intelligence, Reinforcement Learning (RL) is a branch of Machine Learning, which is the *field of study that gives computers the ability to learn without being explicitly programmed*—reporting the definition given in 1959 by A. Samuel [38]. Indeed, a machine learning model imitates the way that humans learn, leveraging the experience retrieved from collected data.In particular, Reinforcement Learning is a computational approach that, through a process of trial and error of interactions with the environment, teaches to a specific agent a behavioral policy to face a given problem. Following the definition of Richard S. Sutton and Andrew G. Barto [39], Reinforcement Learning is

> *a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex, uncertain environment.*

Indeed, we have to see the agent as an infant that takes the first steps in the surrounding world, being himself a part of that same world. As shown in Figure 3.5, the trial and error approach appears very clear: each action taken in the unexplored environment can produce good or bad results, which are represented by the returned reward. The agent's goal is to maximize the total reward on the long term, thus being farsighted in the chosen behavior.

In fact, the great strength of RL is the ability to learn as a human and beyond a human. The method of trial and error, driving the learning process of the agent,

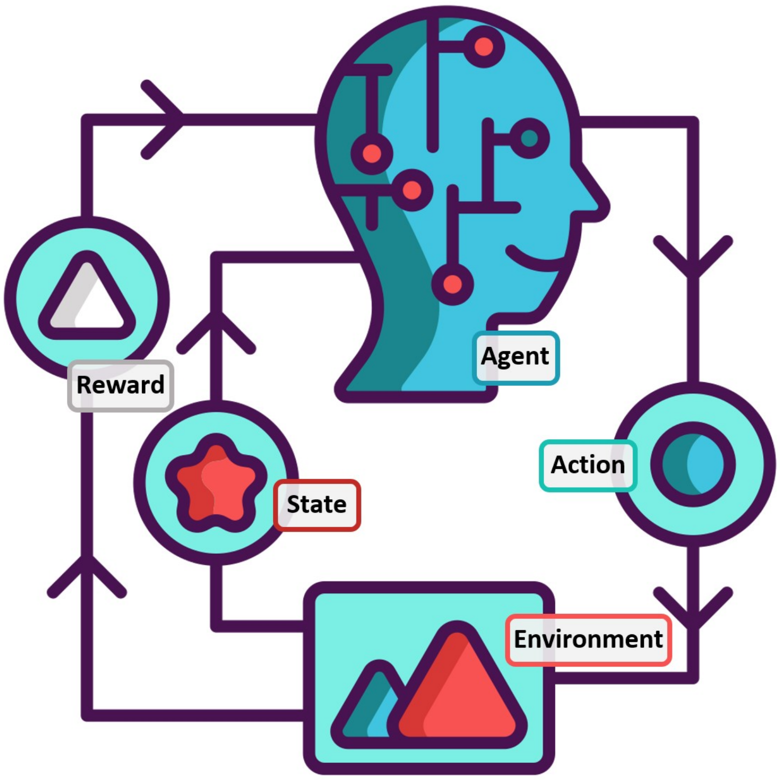

Figure 3.5: RL interaction schema. Here we can see the interactions between the agent and the environment. In the endless loop, the agent moves himself inside the environment and, after each action, he comes out to be in a new state, with more knowledge about the previous one, reinforced by the reward.

can open to a huge range of possibilities, including some still unexplored by human beings. That's because RL aims to start the training from a condition of zero prior knowledge, with the *tabula rasa* approach, getting rid of that bias which always influences human choices. Silver et al. have shown how RL can impressively overcome human intellect and experience, first with *AlphaGo* [40] and then with *AlphaZero* [41], two programs become main experts in the games of Go, Chess and Shogi.

After these incredible results, seems reasonable to wonder if a RL agent would bring the same outcomes in a CPS environment, which can be affected by several different stimuli and, in the worst case scenario, heavily compromised by them. In our case, as previously explained, the environment is a water distribution system and the RL agent has to face a problem of resilience, keeping the network robust against cyber-attacks. An interesting topic could be understanding if the algorithm can bring some novelty in the way we deal with this problem, or if it simply would confirm the approach we have adopted until today. However, the main benefit should come from the automation of the optimal real-time control problem, which could potentially replace the action of a human inspector and react immediately after the detection of a cyber-attack, reducing the negative impact on the CPS.

Among the several algorithms present in reinforcement learning literature, we choose to adopt the Deep Q-Network agent, which smoothly fits with the WDS environment, as we explain in Section **??**. Hereinafter, we will exhibit a general presentation of the algorithm.

3.4.1 Theoretical Concepts

In this subsection we spend few words about generic notions of RL. We will take back them in subsection 3.4.2, together with more theoretical concepts about Deep Reinforcement Learning, to better understand the nature of the DQN algorithm.

Markov Decision Process

Markov Decision Processes (MDPs) are mathematically idealized frameworks that have been adopted by reinforcement learning to reduce problems to mathematical structures for which precise theoretical assertions can be made. Finite MDPs allow to represent the problem as a framing structure, which is useful to straightforwardly describe the interaction between the *agent* and the *environment*.

Formally, a MDP is defined as a tuple $< S, A, P, R, \gamma, \mu >$, such that:

- S is the set of observable states of the environment. If the problem allows to completely explore the environment, the MDP is said *fully observable* and S contains all the possible states. Otherwise we have to deal with a *partially observable* MDP.

- A is the set of actions that the agent performs to interact with the environment.

- $P : S \times A \times S' \to [0, 1]$ is a *state transition probability matrix*, which, given two states $s, s' \in S$ and action $a \in A$, represents the probability to reach s' from state s by performing action a.

- $R : S \times A \to \mathbb{R}$ is a *reward function* which indicates the returned reward achieved by the agent by performing an action $a \in A$ from a state $s \in S$.

- $\gamma \in [0, 1]$ is the *discount factor* used to weight the reward across the time: more it is closer to 0, more the agent focuses on the immediate reward with a myopic attitude; more it grows to 1, more the agent will be farsighted.

- $\gamma : S \rightarrow [0, 1]$ is a probability distribution over S that specifies the probability of start an episode in a state $s \in S$.

The combination of these tuples gives rise to sequences or *trajectories* that are structured like:

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, S_3, ...$$

The values of subscripts in the sequence represent the different interval of time along which the episode, namely the whole progression of a sequence, from its start to its end, takes place. From Figure 3.6 we can clearly extract a generic trajectory following the interaction between agent and environment.

Finally, in MDPs the environment must respect the *Markov Property*, which means that rewards and transaction functions everywhere across the environment depend only on current state and action—one-step dynamic—and are completely unrelated with past trajectories taken of the agent.

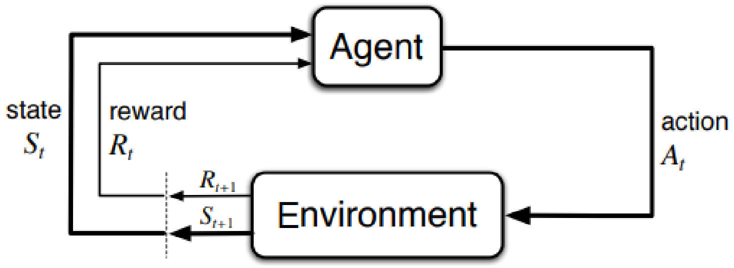

Figure 3.6: Markov Decision Process. The figure represent the interaction between agent and environment in a MDP.

Policy

A *policy* is a probability distribution π over actions in a given state $s \in S$, such that $\pi : S \times A \rightarrow [0, 1]$. In other words, the policy represents the agent's behavior, depending only on his current state—since MDP policies are stationary (not time-dependent)—so the likelihood of choosing certain actions against others.

A problem related to the choice of a policy is the *exploration-exploitation dilemma*. An agent following a policy may end up to sweep always the same trajectories, jeopardising to be stuck in a local optimum. Indeed, (s)he will *exploit* always the same policy, being myopic against other solutions of the problem. For this reason, it's necessary the *exploration* phase, in which the agent can adopt a more random behavior, realising the possibilities he has and searching optimal choices among them.

This problem is known as *exploration-exploitation trade-off* and presents different solutions, like *ϵ-greedy* policies, which select the best action with probability $1 - \epsilon$ or a random one the rest of time, or *softmax action selection*, that improves the previous technique by ranking and weighting the suboptimal actions, so that they don't have anymore the same outcome probability when a random action is selected.

Value Functions

Given a policy π, it is possible to define the *utility* of a state, so a value that allows the agent to understand which policy to choose against the others. This evaluation is expressed in term of expected return, namely the expected discounted sum of future rewards that the agent can collect starting from a given state $s \in S$ and following policy π. The expected discounted return is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{3.1}$$

where $\gamma \in [0, 1]$ is a parameter called *discount rate.*

The function $V^\pi : S \to \mathbb{R}$, mainly used in *policy evaluation* problems, that computes the utility of a state is called *state-value function* and is defined as:

$$\begin{align} V^\pi(s) &= \mathbb{E}_\pi[G_t|s_t = s] \tag{3.2}\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|s_t = s] \tag{3.3}\\ &= \sum_{a\in A} \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V^\pi(s')] \tag{3.4}\end{align}$$

Instead, for *policy control* problems, it's easier to consider the value of the action in each state and this is possible with an *action-value function* $Q^\pi : S \times A \to \mathbb{R}$, defined as:

$$\begin{align} Q^\pi(s, a) &= \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{3.5}\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|s_t = s, a_t = a] \tag{3.6}\\ &= \sum_{s',r} P(s', r|s, a)(r + \gamma \sum_{a'\in A} \pi(a'|s')Q^\pi(s', a')) \tag{3.7}\end{align}$$

Equations 3.4 and 3.7 are known as *Bellman Expectation Equations*, which decompose state-value function and action-value function into the immediate reward plus the discounted value of the successor state or state-action couple respectively. With these equations it is possible to evaluate and estimate the utility retrieved by following a given policy, as a measure of policy goodness.

In finite MDPs, we can precisely find a policy which is the best among the others, namely the *optimal policy* π^*. Thus, since the evaluation metric of policies is the utility given by value functions, this policy will be selected by the Bellman equations that maximize that utility. Moreover, we will call the functions to find π^* *optimal value functions*, which are defined respectively as:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q_{\pi^*}(s,a) && (3.8) \\ &= \max_a \sum_{s',r} P(s',r|s,a)[r + \gamma V^*(s')] && (3.9) \end{aligned}$$

$$\begin{aligned} Q^*(s,a) &= \max_{a \in A(s)} Q_{\pi}(s,a) && (3.10) \\ &= \sum_{s',r} P(s',r|s,a)[r + \gamma \max_{a'} Q^*(s',a')] && (3.11) \end{aligned}$$

Equations 3.9 and 3.11 are called *Bellaman optimality equations*. The first one is derived from the *optimal state-value function* and expresses the fact that, under an optimal policy, the value of a state corresponds to the expected return achieved through the best action chosen from that state. The second one represents the *optimal action-value function* and supports the idea that if the optimal value $Q^*(s',a')$ of state s' at the next step is known for all possible actions a', then we can achieve the optimal policy by selecting the action a' which maximises the expected discounted reward—specified inside square brackets.

3.4.2 Deep Q-Learning

Before starting to introduce the adopted algorithm, we need to make an elucidation about the name: in this work we will use the terms Deep Q-Learning and Deep Q-Network interchangeably, following the convention adopted by the Python library MushroomRL [37], which prefers the second one. However, with a more formal approach, we should say that the correct algorithm name is Deep Q-Learning and that Deep Q-Network denotes only the deep neural network model exploited by it.

Origin and Overview

The born of DQL is due to DeepMind researches in 2013, when they implement the algorithm to let an agent learn how to play Atari games through reinforcement learning [42]. The work was a success, because they found out that the new algorithm outperformed all previous approaches and also, in some cases, a human expert in Atari games.

The DQL is conceived from a variant of Q-Learning, a tabular RL algorithm, which trains a deep learning model, using stochastic gradient descent to update the network weights. In addition, an experience replay buffer has also been inserted, which collects previous transactions and randomly samples among those, to smooth the training distribution with respect to past behaviors. In the shortlist of RL algorithms, DQN take place inside Model-free, Value Based and Off-Policy cluster, as can be seen in Figure 3.7. The table shows RL taxonomy, splitting the wide range of algorithms in two major sets, MDPs and bandits—basically, MDPs with a single state. The second dichotomy is between *model-based* and *model-free.* In this last group belong the most famous RL algorithm, like SARSA, explained also in [39], or Q-Learning [43]—inside the *value-based* branch—and TRPO [44] or A2C [45]—among *policy-based.* Moreover, frameworks as DDPG [46] has an hybrid apporach.

DQN is considered *model-free* because it doesn't need, during the learning or acting phase, a machine learning model to predict the response of the environment and evaluate possible future situations before they are actually experienced – the so called *planning.* Instead, it acts in a trail-and-error manner, without caring in advance of the environment reply.

Secondly, the attribute *value-based*, in contrast with *policy-based*, indicates that, in the learning process, DQN stores only value functions, not an explicit policy, which can be directly derived from the value function that selects more promising actions. Moreover, value-based algorithms are usually more sample efficient, even if they suffer of poor convergence with respect to policy-based ones which tend to directly learn the policy and are more stable.

Indeed, many value-based algorithms similar to Q-learning are part of *off-policy* algorithms. This means that they update Q-values using those of the next state s' and the greedy action a', assuming a greedy policy was followed even if the current policy is not the greedy one. In other words, the current policy is always improved referring to Q-values of the greedy policy. Conversely, *on-policy* algorithms update their Q-values using the Q-values of the next state s' and the action of the current policy a'', so the policy used for updating and the one used for acting are the same, unlike in off-policy methods.

Value-based off-policy algorithms result to be more sample efficient, since they can get the most out of every sample, but at the same time more unstable, because they learn from value functions – and not directly from the policy – and don't update Q-values on the current policy, but exploit another one that could be very dissimilar from the previous.

As DQN, many RL algorithms leverage the estimation of the action-value function, iterating the update of the relative Bellman equation through consecutive steps, such as $Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a')|s,a]$, since it has been proved

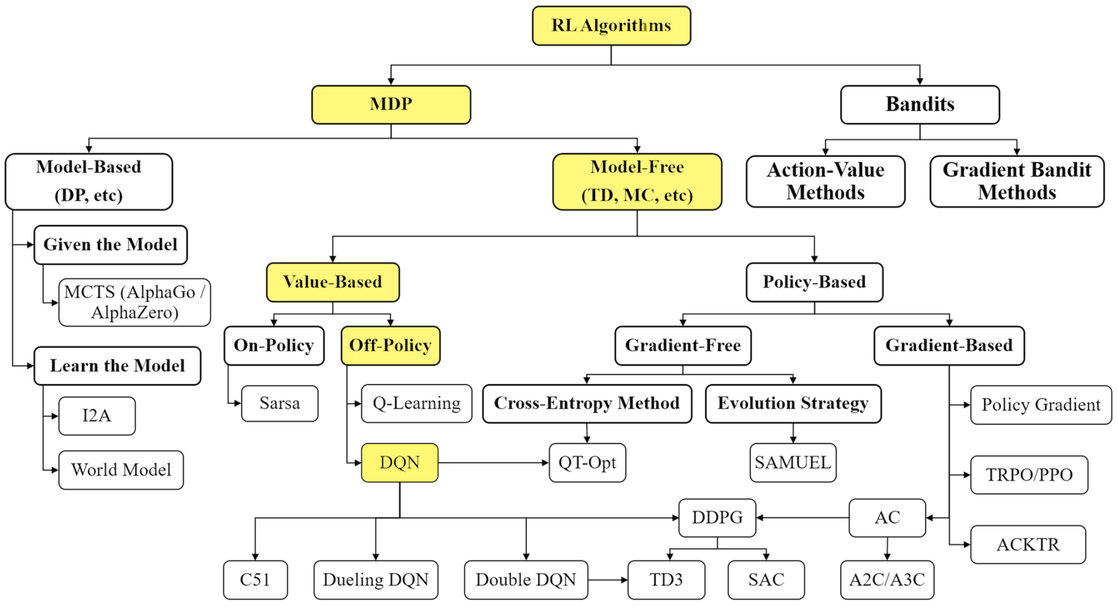

Figure 3.7: Taxonomy of Reinforcement Learning, source [47]. This is a schematic view of concepts and algorithms present in RL field. Depending on problem structures, from here we can infer how to lead our modeling.

that such *value iteration* method converge to Q^*, as $i \to \infty$.

However, this method is unusable in practice, because many environments present a huge state space. In fact, *value iteration* allows to estimate the action-value function separately for each different trajectory, but, since the number of sequences in large state space problems is enormous, the time and data required to gain a discrete knowledge of the environment become critical. Moreover, this method doesn't allow any sort of *generalization* between different sequences, which is a big limitation. For this reason, it is more common to use a function approximator to estimate the action-value function, $Q(s, a; \theta) \approx Q^*(s, a)$, which can be a linear or non-linear function, such as neural network.

Thinking about our DQL, with a neural network function approximator with weights θ, called Q-network or deep Q-network (DQN) if it has multiple hidden layers, we are provided with a model that can be trained by minimising a sequence of loss functions $L_i(\theta_i)$ changing at each iteration. Having $y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s, a'; \theta_{i-1}) | s, a]$ as the target of the loss function for each iteration i, where $\mathcal{E}$ represent the environment and the weights from previous step θ_{i-1} remain fixed during the optimisation of $L_i(\theta_i)$, and $\rho(s, a)$ as probability distribution, called *behaviour distribution*, over states and actions, we can formalize the loss function as:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]. \tag{3.12}$$

Even if the approach could seem similar to supervised learning, here targets depend on the network weights, instead of being fixed from the beginning of the learning.

Computing the derivative of the loss function with respect to the weights we get the subsequent gradient:

$$\begin{aligned}\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}[(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) + \\ -Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)].\end{aligned} \tag{3.13}$$

The computation of the full expectations in the previous gradient could be computationally expensive, thus it is often optimized using the stochastic gradient descent [48]. The approach can be traced back to the *Q-learning* algorithm if we update the weights after each step and replacing the expectations with single samples taken from the behaviour distribution ρ.

The algorithm

Here we want to analyze the DQL more in depth, delving into code-level details. As we can see from the pseudocode (see Algorithm 1), DQN uses a structure called *replay memory*, where are stored samples of the past agent's experience, in tuples like $e_t = (s_t, a_t, r_t, s_{t+1})$. This technique, namely *experience replay*, works by randomly drawing samples from the replay memory buffer to perform a Q-learning update to neural network weights. After this phase, the agent selects and executes an action chosen in accordance with the ϵ-greedy policy. Then it will retrieve a new state space from the environment and will generate another sample for the replay memory.

The update of neural network weights is computed thanks to equation 3.12, representing the loss function as a squared error between the target Q and the predicted Q. Indeed, in DQN we use two identical neural networks with the same architecture, one for the target Q and the other for the predicted Q. This method allows to improve the stability of the algorithm, considering the target approximator as a ground truth and updating at each iteration the other model. Also the target approximator has to be updated at a certain point and its update frequency is considered as a hyperparameter by the algorithm.

Algorithm 1: Deep Q-Learning with Experience Replay, pseudocode from [42]

```
Initialize replay memory $D$ to capacity $N$;
Initialize action-value function $Q$ with random weights;
for episode=1, M do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$;
    for t=1, T do
        With probability $\epsilon$ select a random action $a_t$, otherwise select
          $a_t = \max_a Q^*(\phi(s_t), a; \theta)$;
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$;
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi_(s_{t+1})$;
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$;
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$;
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
          equation 3.13
    end for
end for
```

DQN has many advantages with respect to standard Q-learning. First, it has a greater data efficiency, since it can potentially maximize the exploitation of experience samples reusing them in different updates. Second, using experience replay, it overcomes the problem of learning from consecutive samples that are strongly correlated, causing inefficiency. Third, the behaviour distribution is averaged over previous states, reducing divergence and oscillations in parameters and smoothing the learning. However, a limitation could be the structure itself of the replay buffer, since it is limited to N samples that are overwritten by newer once. There exist more sophisticated approaches that adopt a prioritized memory, giving more emphasis to some transitions against the others, since they are believed more relevant [49].

Chapter 4

Approach

Here, we present the concrete structure of the problem. We start from the hydraulic network we took into account, talking about system interactions, formalizing the optimal control problem and describing how we make it fit with the observed environment. Finally, we analyze the cyber-attacks adopted to performed the experiments reported in Chapter 5.

4.1 System Model

The system taken into account is, first of all, composed by a physical water network, called *Anytown*. Anytown mirrors the features of a small-medium size city and it is very well-known in literature. We chose Anytown for its humble dimensions, since a bigger topology, like *C-Town* or *KY3* would increase the time complexity of our problem. However, Anytown is large enough to test different type of cyber-attacks, proving to be as interesting as a bigger water distribution system.

As shown in Figure 4.1, Anytown is made up of a reservoir (R0), providing water to the system, and two tanks, T41 and T42, which have the task of maintaining a certain level of pressure across the network. The tanks release stored water in high usage periods, while the two pumps, P78 and P79, control hydraulic level across the system. There are also twenty-two junctions connecting a discrete number of pipes. Each component has two kind of properties: static and dynamic. Static properties are enlisted in the .inp file, provided as input to the simulator and formatted following the EPANET-toolkit instructions [50], and represent structural characteristics that do not change over time, like minimum and maximum level of tanks, base-demand of junctions or initial settings. Conversely, dynamic properties are variables computed at runtime by the simulator and can be read-only, like pressures of tanks and junctions, or changeable from external inputs, as status of pumps.

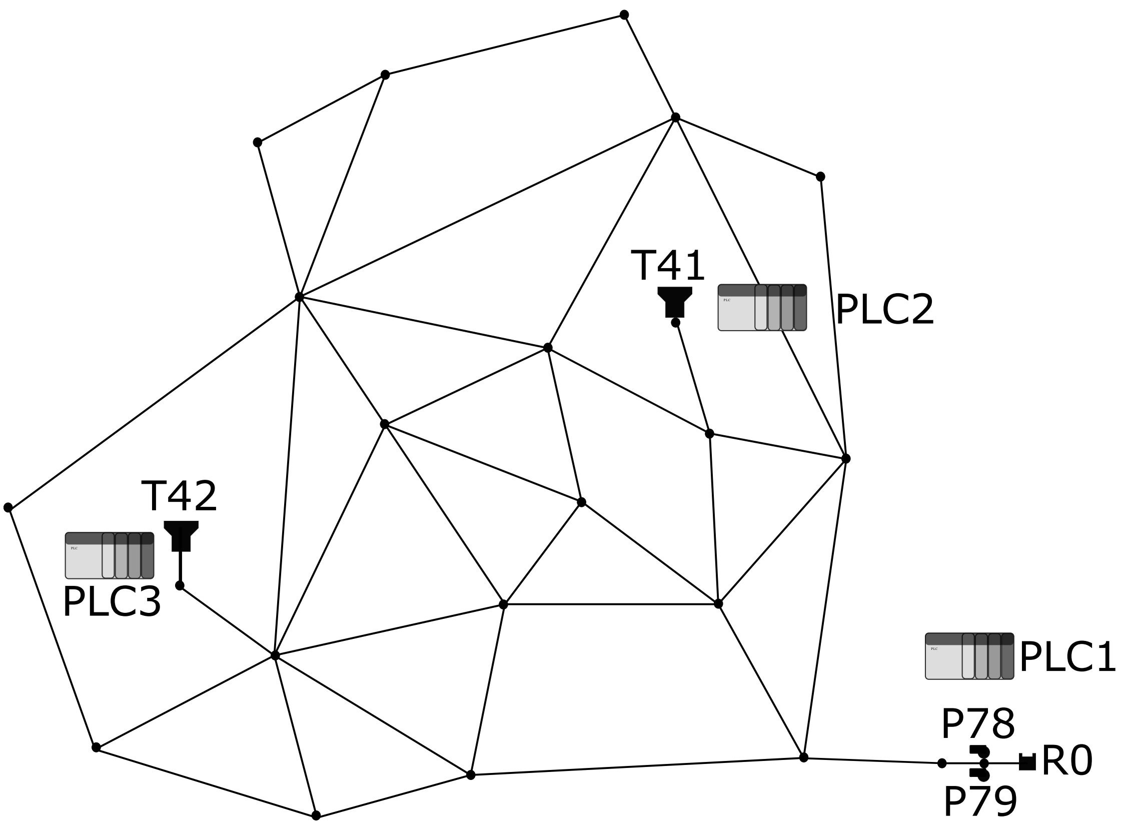

Figure 4.1: Anytown water distribution system. In the snapshot is depicted the physical layer and the ICS equipment composed by PLC1, an actuator appointed to control pump status, and PLC2 and PLC3, sensors designated to retrieve tanks and junctions values. In the physical layer we can clearly distinguish the reservoir, pumps and tanks, highlighted by respective symbols and labels. However, even if not labeled, we can observe also junctions and pipes, which are respectively black dots and segments in between.

On top of the physical layer, takes place the cyber-layer, comprising ICS equipment and the industrial control network. As shown in Figure 4.1 the ICS equipment is composed by three PLCs, namely PLC1, PLC2 and PLC3, interacting with the physical layer through sensors or actuators. Moreover, a centralized SCADA server, not represented in Figure 4.1, receives and stores information coming from sensors and sends control data to the actuators. However, PLCs, if adequately configured, can also exchange information among themselves, without passing throw the SCADA server.

In our configuration, PLC2 and PLC3 act as sensors, reading node values from the physical layer and sending them to the SCADA. Each of them belongs to a different substation, as explained in Section 4.4, and collects readings from targets defined in the PLCs Config File, described in subsection 3.3.2. On the other hand, PLC1 acts as an actuator, receiving information from the SCADA server and performing the control action on both pumps.

Concretely, the readings and actuators variables are assigned as indicated in Table 4.1. All the enlisted variables are dynamic properties of the system: the ones read by sensors are read-only variables, instead the others managed by the actuator clearly not. Moreover, the tanks belong one to each sensor and also variables are assigned not at random, but considering PLC2 responsible for the right side of the network and PLC3 of the left side, following the principle of local proximity.

PLC	Type	Main	Secondary

PLC1	Actuator	P78(status)
		P79(status)	

PLC2	Sensor	T41(level)
		J20(level)
			demand and supply of following junctions:
			J21, J14, J2, J13, J1, J3, J6, J15, J5, J19

PLC3	Sensor	T42(level)	demand and supply of following junctions: J4, J18, J12, J7, J8, J16, J17, J11, J10, J9, J22

Table 4.1: PLCs variables. In the table are enlisted the PLCs deployed for Anytown, specifying their type and variables whose PLCs manage the value. The distinction between main and secondary is the following: main variables are necessarily collected to populate state and action spaces of the control agent; secondary variables are useful for the computation of the custom objective function, but not for the MDP definition. We explain this in details in Section **??**

Along with network the components, the system presents also some important parameters, essential to outline the trend of the simulation. Indeed, to reproduce an experiment it is necessary to set reasonable demand patterns, which in EPANET are conceived as lists of multipliers that define the water request across the system for the entire duration of the analysis. Demand patterns can be considered the main input of the environment, since they drive the scenario in which we want to test the network.

In addition, EPANET requires also to specify the demand model option between two choices: Demand-Driven analysis (DDA), which assumes that nodal flows are always satisfied at all demand nodes regardless of the available pressures at those nodes, or Pressure-Driven analysis (PDA), which takes into account the pressure at demand nodes. The difference between the two modalities, as shown by authors of [51], is met in presence of pressure deficit, where the the DDA risks to register negative pressure values, unrealistic in a real-world scenario. For this reason, in our experiment we always prefer to use PDA.

Finally, other important parameters are the time options. By means of these, we can, for example, decide the duration of the experiment, the time interval for sampling the state of the network, the frequency of demand pattern updates and many others, less relevant for this work.

4.2 Epynet

As explained in Section 2.2, the first objective of this work is the enrichment of the Digital HydrAulic SIMulator (DHALSIM) with a new Python library, created to wrap the EPANET simulator [31]. This object-oriented wrapper, called *Epynet* and developed by Vitens, the largest water drinking company in The Netherlands, is an open-source library, available on GitHub [21].

Epynet is presented as a simple and clean Python library, easy to investigate and well-organized. It looks like a mere implementation of the API functions made available by EPANET's developers and it is conceived by Vitens to improve and optimize the water treatment and distribution process. Actually, the project seems to be quite inactive and not constantly updated as the WNTR library, but it has been a good starting point for our work, proving to be a suitable wrapper after some modifications.

The attraction of Epynet with respect to WNTR, the already integrated EPANET wrapper, is the possibility to run experiment in a step-by-step manner, feature not available in WNTR, which works like a black-box tool. This *stepwise simulation*, enriched by us with the possibility to apply controls action at runtime, is important both from a cyber-security and a reinforcement learning point of view. From a cyber-security point of view because, it allows to study frame-by-frame the internal state of the WDS undergoing cyber-attacks, analyzing the correlation of disrupting events with the consequences on the physical layer. The study could be lead also with WNTR, checking the results at the end of the experiment, but without the possibility to infer the network properties at runtime. From an RL perspective, we have the opportunity to know the real-time system state opens up to the use of an online algorithm assessing the control problem. Indeed, the stepwise feature allows the control of overwritable dynamic properties, like the pump status, giving a great incentive to the conception of new interactive experiments, in which the researcher can stress the system with specific crafted stimuli given at runtime.

Although the introduction of a feature like the stepwise simulation is essential to achieve the next goals presented in Section 2.2, thus being a mandatory stage for the completion of this work, the integration of Epynet gives also other advantages, in term of performances. Indeed, being a lightweight wrapper with respect to WNTR, which provides more advanced features—not significant for this work, like the computation of pipe leaks caused by disrupting events—it allows to considerably reduce time complexity, especially in experiments analyzing huge WDS topologies, with no relevant differences regarding the accuracy of results.

4.3 Control Problem

In this section we explain how we conceptualize the heart of this work, which inspires the title of the thesis and constitutes the core of second and third goals explained in Section 2.2: the control problem.

The control problem is the task of determining control and state trajectories for a dynamic system which can change over time, with several applications in different fields, like aerospace, robotics, economics, finance and so on. Actually, the interesting part of this topic regards, more than the generic version, the optimal control problem, namely the process to find a law to control a system such that an objective function is somehow optimized. The objective function can be a reward function that has to be maximized or, conversely, a cost function that needs to be minimized. For example, a possible task of optimal control could be finding the best route that minimizes the driving time between two cities, considering constraints, such as limited amount of fuel, and laws between variables, like the fuel consumption proportional to the speed of the car.

The first consideration to do, studying a control problem, is understanding the configuration of the system that has to be controlled. Indeed, as shown in Figure 4.2, there exist two different formalization: we can have an open loop system or a closed loop one. An open loop control system is a system in which the output does not affect the control action of the system, so there is no feedback loop between output and input. On the other hand, a closed loop control system presents one or more feedback loops between its output and input, allowing to reach the required output by evaluating the provided input by means of an error signal that highlights the diversity between output and input. The advantages brought by an open-loop control system are the simplicity, stability and speed of response. Nevertheless, closed loop control systems result more reliable, accurate and not affected by system disturbance.

4.3.1 WDS Optimal Real-Time Control Problem

In this work, we aim to study the application of the optimal control problem to a water distribution system, initially evaluated in normal operating conditions and then with the addition of cyber-attacks. As explained in Section 2.1.3, in literature the WDS optimal control problem is often conceived as a pump scheduling problem and it is assessed by means of deterministic optimization algorithm or metaheuristics. These solutions typically consider the WDS as a open loop control

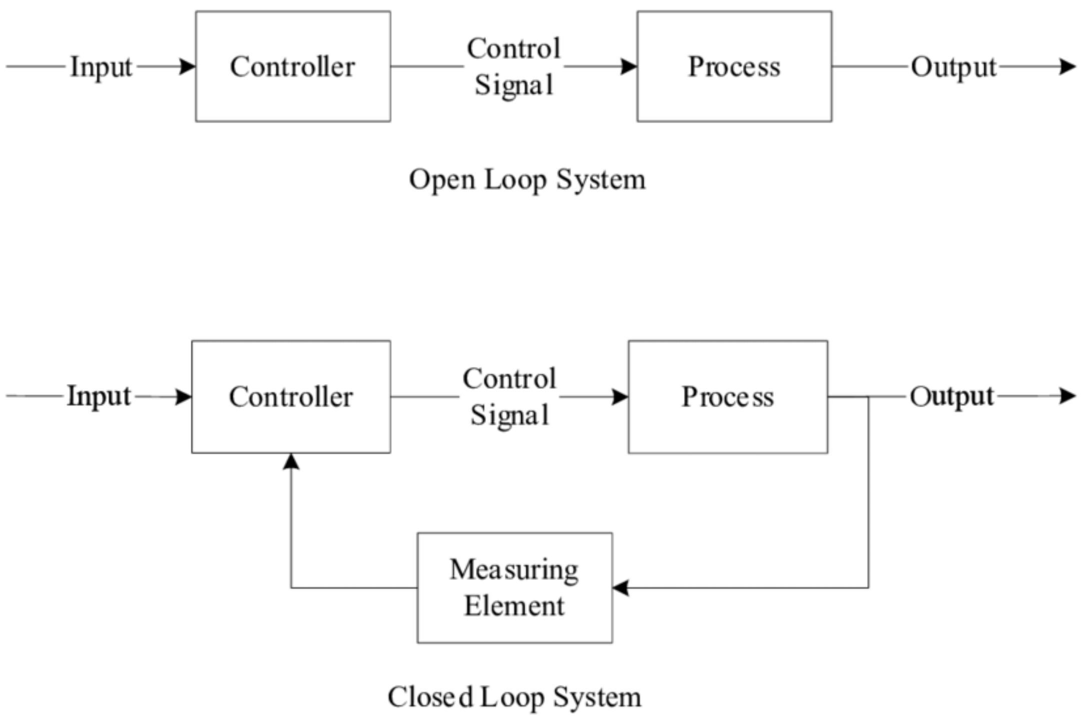

Figure 4.2: Control problem schema. In the upper side is represented the schema of an open loop system, deprived of any feedback loops. Below, is illustrated a closed loop control system, which differs from the previous one for the presence of the feedback loop.

system [52], providing different inputs to the network in terms of pumps schedules and evaluating which one produces an output as similar as possible to the desired one.

However, the adoption of a reinforcement learning algorithm to perform the optimal control problem entails a closed loop structure. In our case, we tackle the problem with a continuous interaction between the agent and the environment where the input of the closed loop is directly dependent on the output of the previous cycle. We think that, in WDS environment, a closed loop approach could be better than an open loop one especially in presence of cyber-attacks, where the input of the controller can drastically change—we explain details in Section 4.4.

Speaking more concretely, in a WDS, following the closed loop system depicted in Figure 4.2, the input signal is the system state, namely, using MDP notation, a state $s \in S$, where S in the set containing all the possible states of the environment. The controller is the RL agent, in this case a DQN, which, given the current state, chooses the best control signal, so the best action $a \in A$, to apply to the physical process. The component of DHALSIM appointed to provide outputs is the EPANET simulator, enveloped by Epynet wrapper, and the yielded output is nothing else than a new state $s' \in S$. Translating the theoretical concepts in more practical ones, as shown in Figure 4.3, at the beginning the agent is fed with the following variables, belonging to the state space:

- a *time* variable, which gives a temporal information;
- *node level* variables, providing the pressure registered inside specific nodes;
- *trend* variables, regarding the tendency of demanded water across the network;
- *attack related* variables, which can give information about ongoing attacks, as a detection algorithm.

The agent, after elaborating the received variables, outputs a control action aimed to select the status of pumps. These actions can vary in number depending on the amount of controlled pumps, but in the simpler version, considering only one pump, the feasible actions are just two:

- turn on the pump;
- turn off the pump.

In presence of multiple pumps the actions are 2^n where n is the number of pumps, one for each possible combination of pumps status.

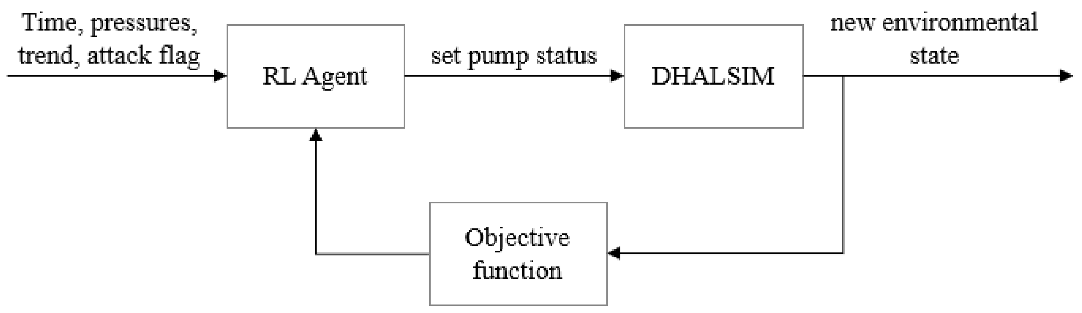

Figure 4.3: Closed loop control of WDS. The picture describes the general concept behind the closed loop control problem of considered water distribution system.

The selected action is given in input to the simulator which perform a step of the simulation, collects the results, consisting in a new state of the environment, and sends it back to the controller, which has to choose a new action for the next step, and so on until a terminal state is reached. In addition to the new environmental state, the comparison between the computed output and the expected one is represented by a loss function calculated by the agent and useful to understand if the prediction process has been accurate enough. To do so, we use a reward function, computed on the new configuration of the environment, that helps to update the weights of the neural network. The choice of the control action depends, for the majority of times, on those weights which calibrate themselves on values of the observed state. The reward function used in this work is described in Section 5.1.3.

The optimal criterion that leads to the actuation of the control actions is generally oriented to keep the pressure across WDS within specific bounds, to avoid tanks overflowing, generated by an high level of water in presence of a poor request, and network dehydration, caused by a small level of water during periods of high demand. A measure that can guarantee the satisfaction of the optimal criterion is the Demand-Satisfaction Ratio (DSR), which computes the ratio between the amount of supplied water and the quantity of requested one, outputting a value included between 0 and 1. The DSR represents the ability of the WDS of answering to the needs of consumers. Another optimal criterion usually considered is the minimization of pumps energy consumption. Indeed, the pump scheduling problem could be a way to optimize the use of pumps in the network, avoiding high costs and energy wastefulness. However, in this work we do not consider the minimization of energy consumption as a goal to achieve, since in presence of cyber-attacks we are more interested in preserving the reliability of the system, rather than restricting the utilization of pumps.

The benefit brought by the exploitation of a reinforcement learning technique is the possibility to teach the system how to adapt to environmental variations, in real-time. The optimal criterion is craved at each step of the simulation and the agent tries to reach it in the choice of every single control action. Instead, with an open loop configuration control actions are decided at the beginning of the simulation and the optimal condition is measurable only at the end of the entire process. In addition, this configuration could lead to many problems in presence of drastic changes within the system, since it does not have a feedback loop bringing useful information on past experience.

4.4 Attacker Model

Here, we focus on the part of the thesis related to cyber-security threats, showing where the attacks can be deployed and which vulnerabilities we consider in the experiments. As mentioned in Section 3.3, cyber-attacks are provided by DHALSIM itself, which creates a specific process handling the attacks declared in the Attack Config file—presented in subsection 3.3.2.

4.4.1 Cyber-layer of Anytown

As shown in Section 4.1, the cyber-layer of Anytown is made up of three PLCs and one SCADA server interacting by means of a stack of communication protocols, illustrated in Figure 3.3. Since the WDS is a spatially distributed system, it's reasonable that the cyber-layer is not located within a single Local Area Network (LAN), unable to cover too large areas. Thus, as depicted in Figure 4.4, there are four different LANs, called *substations*, one for each PLCs and the fourth for the SCADA server, connected together with a single Wide Area Network (WAN).

From a cyber-security perspective, the implications of this structure are that an attacker can compromise only one ICS component for each substation, so (s)he has to spend more resources to tear down two PLCs at the same time. Indeed, if all the PLCs were located in the same substation, it would be sufficient only one attack to compromise all of them. Moreover, the expedient of the substations allows the researchers to study single parts of the network struck by attacks or to implement double attacks hitting two substations at the same time, analyzing the different impacts on the WDS.

DHALSIM, as explained is subsection 3.3.2, provides two kind of attacks, namely device attacks and network attacks. The first ones are oriented to strike PLCs behaviour, changing the way they apply the control logic, like a Denial of Service attack. Thinking to Figure 4.4, a DOS attack can strike for example substation 2, preventing PLC1 to apply the control action on the pumps. In the same way can also be disrupted PLC2 or PLC3, impeding the correct collection of readings of the network state. In both cases, the scan cycle is compromised because a necessary control action cannot be applied or relevant network information remain concealed. On the other hand, network attacks work in another way and aim to tamper the connection between two or more ICS components. A common threat is the Man-In-The-Middle (MITM) attack, which can jeopardize the data integrity without directly disrupting the PLCs. Moreover, this kind of attack could be more difficult to detect with respect to device attacks, since the cyber-layer does nt stop working and the ICS equipement is not struck down.

4.4.2 Attacks

Among the cyber-attacks provided by DHALSIM, only some of them results reasonable to be experimented in the order to assess the optimal control problem. Indeed, in this work, we tested only specific configuration of network attacks that can fit with the configuration of the experiment. In fact, striking down a PLC with a DOS attack would compromise the unfolding of the simulation itself since the synchronization messages with the control agent are totally lost. At the same time, certain network attacks that tamper the communication towards actuators result

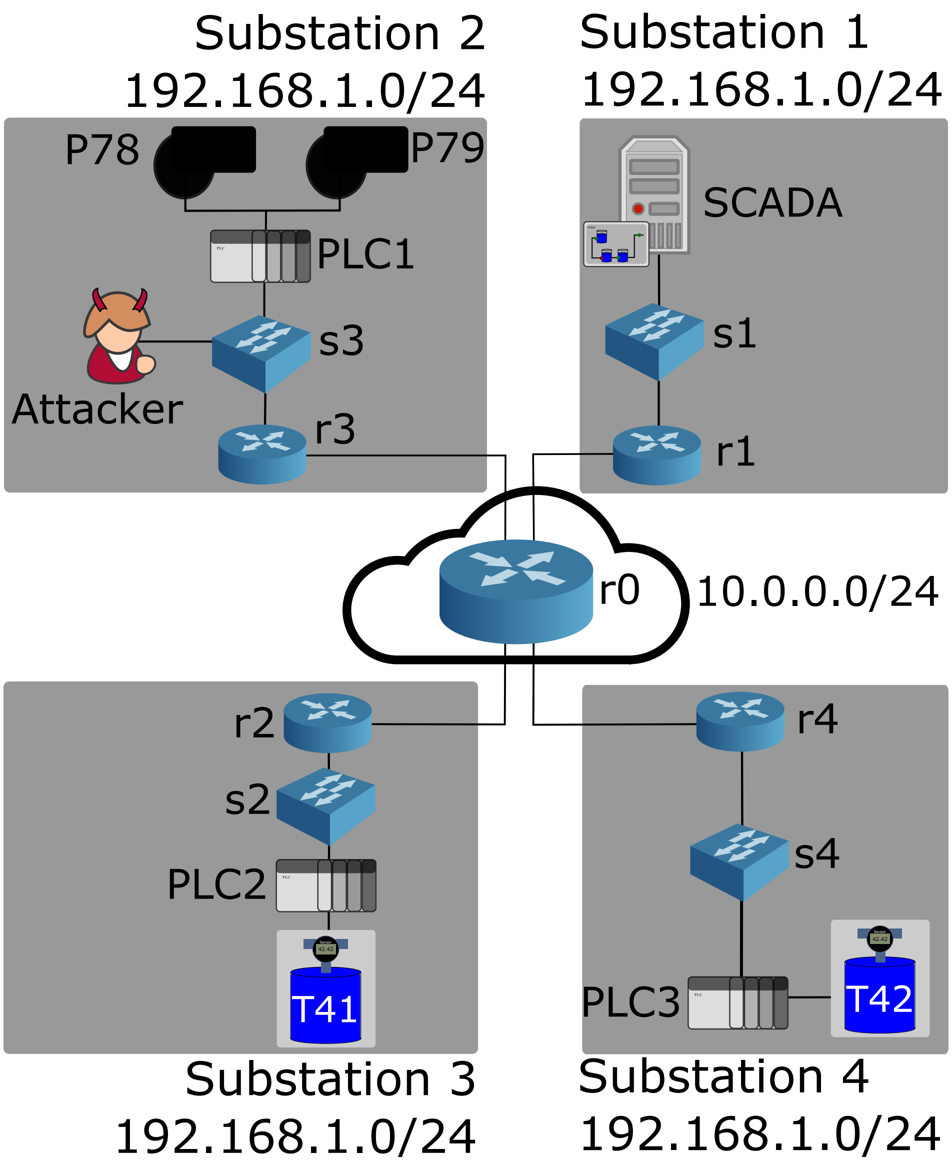

Figure 4.4: Antown cyber-layer. In the figure is depicted the structure of the digital layer of Anytown, with the four substation containing a PLC or the SCADA server. Every substation represents a single LAN and all the four LANs are connected together by the WAN router. The attacker drawn in substation 2 is trying to compromising the connection towards PLC1, used to apply the control action.

useless in this environment, because they prevent the researcher to analyze the impact of the control agent.

Actually, a MITM attack is a quite complex attack that needs some preliminary steps. Indeed, the attacker has to fake himself to be a component of the network to be trusted by other devices. In Figure 4.5, we can see that the attacker first of all sniffs packets routed to PLC1 from other addresses. Then, with an ARP spoofing attack, he puts himself in the middle of the communication between PLC1 and the relative switch and he start to control the traffic in both directions. In this case, he clearly wants to fool PLC1 by sending outdated or modified data, changing the ending of the control action to his liking. Moreover, this kind of attack is not so easy to detect. Indeed, realizing that an attacker is performing a MITM on some channels of the network, it does not always suffice to control the output control actions, since he could have thought to make them plausible. The proper way to discover the attack implies the utilization of a packet sniffer that can discover some suspicious ARP request from a unknown IP address. Another way to prevent MITM attacks consider to establish a safer communication with protocols that can impede this kind of threats, like SSL protocol.

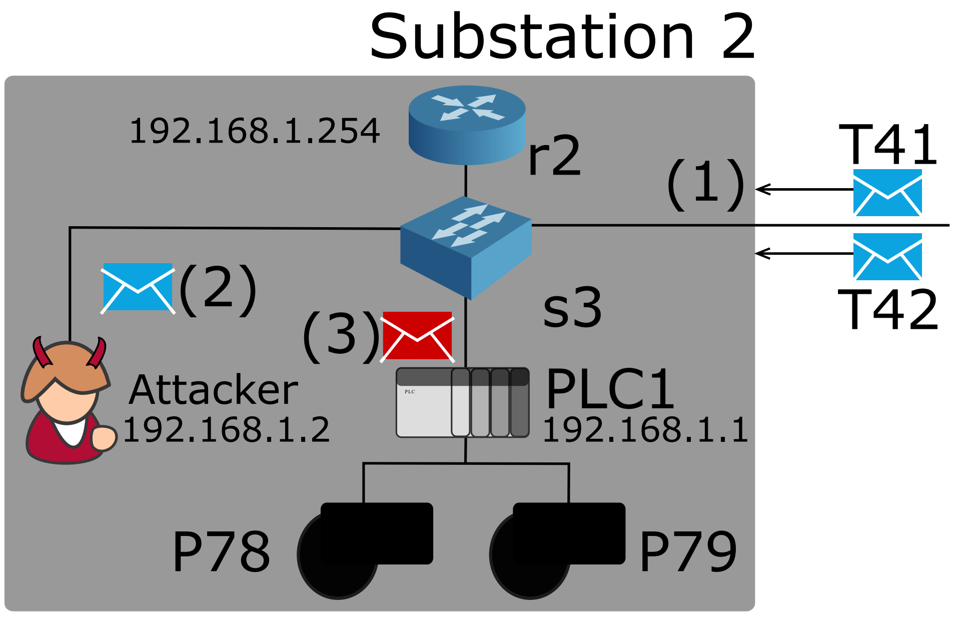

Figure 4.5: MITM attack. Here we can see in details the Man-In-The-Middle attack. In step (1) the attacker sniffs the packets coming from the sensors and provided to PLC1, the actuator, stealing session information. Then, he performs an ARP Spoofing attack to put himself in the middle of the communication between PLCs (2). Thus, he crafts packets containing sensors readings as his liking, with the purpose to fool the actuator, and he sends the packets to PLC1 (3). The unaware actuator applies the control logic suggested by received packets, compromising unintentionally the system.

In this work we focus on two specific case of MITM attack, which aim to threat the integrity of sensors reading sent to the control agent. This means that the attacker can try to place himself between PLC2 and the SCADA server, corrupting information regarding the right side of the network, or between PLC3, supervising the left side, and the SCADA server, or even both together. Our intention is trying to fool the control agent, located on the SCADA server, with tampered data that should be used to train the neural network. In this way, we try to test the resilience of the network in front of fake information with two different settings: one in which the attack is concealed to the agent and one where the agent knows that an attack is ongoing, as if (s)he got a perfect anomaly detector.

Chapter 5

Experimental setup

In this chapter we describe the concrete application and the implementation of the theoretical concepts explained so far. First, we talk again about the control problem, by delving into the explanation of interactions between agent and environment. Here, we can grasp the shape of the environment seen by the agent, how the agent controls the physical system and what are the reasoning behind each control decision. Secondly, we present the integration of Epynet and of the control agent with DHALSIM. Then, we describe the input data provided to the framework and the corresponding obtained output. In the end, we show the configuration of the experiments and the parameters used for the simulations.

5.1 Agent and Environment

Here, we explain in details the interactions between the agent and the environment of the cyber-physical system. Actually, the environment seen by the agent is not really the whole cyber-physical system, but only its physical processes, the water distribution network, because it is not aware of the ICS equipment and the traffic across the communication network. Indeed, since the agent is deployed on the SCADA server, we can see the it as a part of the cyber-physical system itself, because it implements the reasoning of the controller.

As explained in the background chapter 3.4.2, the agent is the Deep Q-Network (DQN), a RL algorithm appointed to solve the optimal real-time control problem tackled by this thesis, described in Section 4.3. In other words, DQN is the mind behind each control action, thus the brain and the knowledge keeper of the whole infrastructure. On the other hand, the environment is the Water Distribution System (WDS) presented in Section 4.1, called Anytown, which is unaware of the agent presence and evolves in relation to the stimuli given by actuators, namely the two pumps.

As shown in Figure 5.1, the interactions between agent and environment have a looped structure and are driven by the ICS components. Specifically, PLC2 and PLC3 represent the sensors of the system and are in charge of reading the environment state and send it to the SCADA server. Concretely, readings contain values relative to node properties enlisted in Table 4.1. The agent runs in the SCADA server and plays the role of controller of the system. Processing the readings, it elaborates suitable actions for the actuators to optimize an objective function. For example, in normal operating conditions, the objective function could aim to maintain the pressure across the WDS within specific bounds, to avoid situations of dehydration or overflowing. After getting the control action from the SCADA, PLC1 updates the status of pumps, which govern the evolution of the hydraulic network. The loop is closed by the simulation of the WDS until the next hydraulic step, when the experiment is frozen in a frame and readings are collected. Of course, in a real-world scenario the WDS does not stop to wait the read of sensors and the execution of the remaining control loop, but this is a reasonable approximation of the true process and a reliable test-bed to analyze our experiments.

5.1.1 Observation Space

The observation space is the set of variables needed by the agent to grasp the state of the environment. Indeed, observation space—the capital S of a Markov Decision Process (MDP)—encloses the current configuration of the WDS and it is the only information required to calculate the next control action. For the optimal real-time control problem that we are facing, among the collectable variables of the system, the ones that seem more reasonable to observe are:

- the timestamp of the current reading, split in *time*, intended as time of the day, and *day*, the day of the week, useful to know which part of the day is observed, since in a WDS the water demand changes across the different hours of the day;
- the level of the tanks, namely *T41_level* and *T42_level*, since the evolution of the system state depends on the water stored in the tanks;
- the pressure at the downstream junction of the pumps, i.e. *J20_level*, the first junction registering inputs provided by pumps;
- the moving average of the current demand pattern, *demand_SMA*, computed within a fixed rolling window to grasp the trend of water demand across the network;

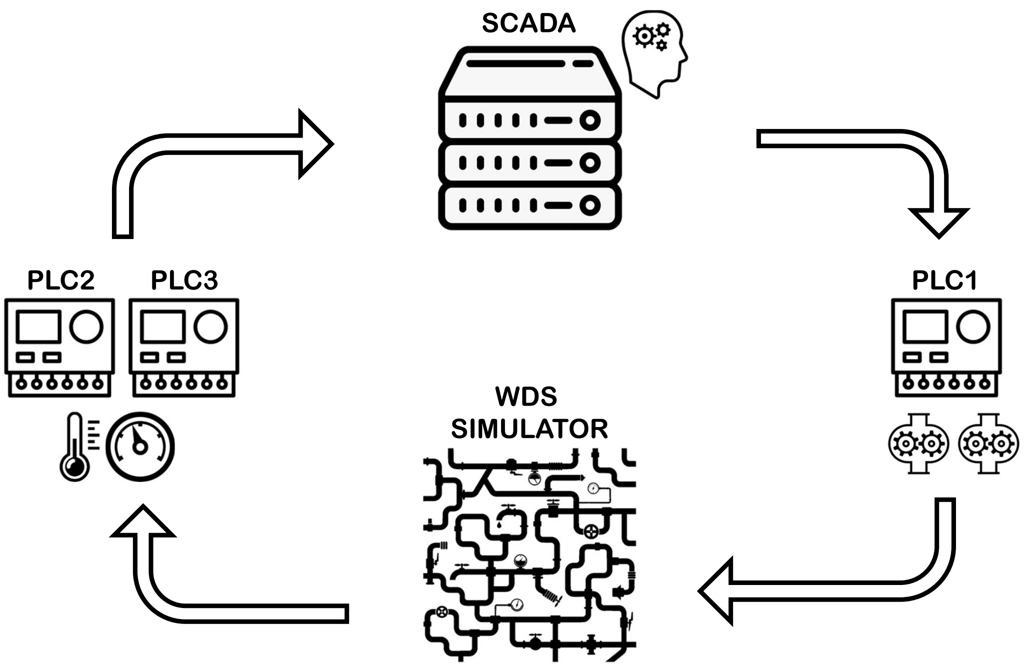

Figure 5.1: Control cycle. The figure represents the loop that drives interactions between agent and environment. PLC2 and PLC3 send readings of the current state to the agent, placed on the SCADA server. The agent processes network observations and outputs a control action, that is sent to PLC1, to apply the control logic on actuators. Then, the system evolves depending on the input given by pumps and the loop restarts from the beginning.

- the attack flag, the *under_ attack* variable, which simulates the output of a perfect detector and specifies if in the current step there is an ongoing attack.

5.1.2 Action Space

The action space is the set of the control actions $a \in A$ computed by the agent after the investigation of the observation space. Those actions indicate the measure to apply to actuators in order to satisfy the objective function and entail the change of pumps status. In our case, since Anytown get only two actuators, namely pumps P78 and P79, the possible actions that the agent can choose are 4 and are indicated as follows:

- action [0]: both pumps closed;
- action [1]: P78 open, P79 closed;

- action [2]: P78 closed, P79 open;
- action [3]: both pumps open.

Changing the status of the pumps is important because there is not always the need to have both pumps active. Indeed, if the water level in the tanks is close to its maximum threshold there could be the risk of overflow. Moreover, in some cases it could be required to consider the energy consumption caused by active pumps. At the same time, keeping the pumps always closed is not a good solution since it could lead to a pressure drop.

5.1.3 Reward Function

To learn from experience the agent needs a sort of incentive, to confirm that it is going in the right or wrong direction. This incentive in the RL field is called *reward*. Thus, the reward function is the equation that outputs the reward achieved by the agent in every single step. To learn, the agent has to maximize the obtained reward, performing those actions that allow him to increase the output of the reward function. For a researcher, is important to model a suitable reward function in order to train a robust agent. This means that a reward encouraging the agent to act as expected is not always a good idea, since it could lack of experience derived from exploration. Indeed, exploring several observation spaces and building a slower, but wiser, knowledge is better than rushing the planned solution and could lead to unexpected and revolutionary ways to solve the problem. In our case, the reward function is designed considering three principal concepts enlisted hereafter:

1. The first is the satisfaction of the water demand across the network, which represents the ability of the system to supply enough water to, for example, city dwellers. To evaluate this measure we use the Demand-Satisfaction Ratio (DSR), defined as:

$$DSR(t) := \frac{delivered\_water(t)}{requested\_water(t)} = \frac{\sum_{i \in J} J_i^D(t)}{\sum_{i \in J} J_i^B(t)},$$

   where J is the set of junctions in the system, J_i^D the delivered water by junction i and J_i^B the requested water—B stands for basedemand—by the same junction i. Notice that this measure is computed in a precise moment of the simulation and represent the DSR only in a single instant. Indeed, from now on we refer to it with the name *step-DSR* to distinguish it from a metric presented in Section 5.3, used to estimate the goodness of the results at the end of the experiment, and called just *DSR*. The step-DSR, and actually even the DSR, appear always as a value contained in the interval [0,1], because in case the delivered water overcomes the requested one the result is cropped to 1.

2. The second is the number of status updates done by each pump. Actually, this component of the reward function is conceived as a penalty variable since we want to discourage the continuous change of pump status. The reason is related to the fact that in a real-world scenario pumps are huge machines that cannot be switched on or off in a second, but requires some time to change their status. Moreover, without this penalty can happen the alternation of actions 1 and 2, which are equivalent in term of effects on the environment—since pumps are located in the same place—but that bring a useless waste of energy. Finally, this term trains the model to choose more accurately the control action, since if the selected action is wrong, it is not totally free to roll back to previous settings for the next step. The number of updates with respect to the previous state is computed as follows:

$$pump\_updates := d_H(bin(a_t), bin(a_{t-1}))),$$

thus as the Hamming distance between the binary representation of two consecutive actions. For example, if $a_{t-1} = action[1] = bin(1) = 01$ and $a_t = action[2] = bin(2) = s10$, then $D_H(01, 10) = 2$, so the number of updates is 2. That also explain what said before about the energy wastefulness got by alternating action 1 and 2, since the required updates are two, one to turn off pump P78 and the other to switch on P79.

3. the third concept is related to the prevention of overflow problems. The intrinsic nature of step-DSR, that tends to maximize the satisfaction of water demand, allows to easily avoid pressure-drop issues, but at the same time, increases the risk of tanks overflow. This comes from the fact that, without putting an overflow penalty, the agent is trained to maximize the step-DSR with as few updates as possible, so it basically learns to keep both pumps active all the time. This surely brings wastes of energy and risks of tank overflow. For this reason, we believe that it is reasonable to insert the overflow penalty to discourage the aforementioned behaviour. The overflow term is defined as follows:

$$overflow\_penalty := \begin{cases} \frac{tank\_level - threshold}{max\_level - threshold} & \text{if tank\_level} > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

where $threshold = max\_level \cdot risk\_percentage$, *max_ level* is the maximum level of water in the considered tank and *risk_percentage* is a parameter that indicates when the agent starts to receive penalties—for example from the 95% of the maximum level. The idea is that the agent begins to get penalties when one of the tanks overcome the threshold level and it continue to receive bigger and bigger penalties as the pressure reaches the relative maximum level.

The complete reward function, with the three terms presented above, looks like this:

$$reward := step\text{-}DSR - \frac{pump\_updates}{2} - overflow\_penalty$$

Notice that the *pump_ updates* value is divided by 2. This is done to align the maximum value reachable by all the terms and also to not give less priority to *overflow_ penalty* with respect to *pump_ updates*. Indeed, without that division the agent could learn to not update pumps—for example with a transaction from action 3 to action 0—in case of overflow, since it would receive a lower reward. However, in our opinion is more important to avoid an overflow event rather than consume a bit of more energy by changing pumps status.

5.2 Implementation

Here, we describe the key features implemented during the development of the thesis. We start presenting the implementation of Epynet [21], the Python wrapper of EPANET, initially as a standalone environment and then integrated with DHALSIM. Secondly, we speak about the implementation of the control agent in Epynet until the total integration with DHALSIM. Indeed, before switching to DHALSIM, we have setup a simple environment built on top of Epynet, which allows to run some experiment considering only the physical process. This lightweight framework is essential to test the results of simulations with Epynet wrapper, which has to be consistent with those obtained with WNTR and EPANET. Moreover, it is an important workbench for the implementation of the DQN, run in normal operating conditions, since attacks are not implemented in this version.

5.2.1 Integration of Epynet

As a standalone wrapper, Epynet does not present a large number of features, as it is designed to run simple experiment through the basic API provided by Epanet's developers within the EPANET-toolkit [50]. The strength of Epynet is the possibility to perform the hydraulic analysis in a step-by-step manner, with a combination of methods simple to understand and easily adjustable. Thus, from a code-wise point of view, it is way less twisted than WNTR, which is a more structured library that appears as a black box framework, thus more complex to modify at our liking.

The main and more significant change brought with our work is the introduction of the *interactive stepwise simulation*, a cornerstone of this entire work. This feature allows to run experiments stepping into the simulation a bit at a time, sampling the state of the network with a frequency specified by the *hydraulic step*—a time parameter written in the .inp file. After the generation of each sample, with this functionality we can infer the measures of all links and nodes and we are able to edit some current dynamic properties in the actuators, for example the status of pumps. That is the reason why we define *interactive* the stepwise simulation.

Some minor modification are the additions of methods to set time options and demand patterns, without the need to manually modify the .inp file, a method to export simulation analysis in a well-organized .csv file, and the implementation of Epanet API not exploited by the wrapper. Moreover, an important step is the update of the Epanet library imported by Epynet, which is still remained at version 2.1. Indeed, this release does not allow to perform Pressure-Driven Analysis (PDA), but only the less realistic Demand-Driven Analysis (DDA). This feature is implemented in Epanet from version 2.2, so with our intervention Epynet has been updated to the most recent release and can assess PDA, as required by DHALSIM.

After these changes, the integration with DHALSIM is really straightforward. Indeed, since it already presents methods to handle WNTR simulator which are very similar, in term of functionalities, to Epynet's ones, it is required only to split the cases when the calling function needs to invoke a specific wrapper rather than the other. To choose the desired one, we have only to set the simulator type in the Config File, as explained in subsection 3.3.2, in correspondence of *simulator* key. The main difference, regarding the portion of code related to Epynet, can be seen in the method that runs the simulation. Here, with WNTR is called a single black box function that basically complete the simulation by itself, without giving the possibility to interact with the environment. Differently, in code lines related to Epynet the different steps of the analysis are pretty clear and transparent, and this structure let to the programmer the opportunity to change the behaviour of the experiment.

5.2.2 Integration of DQN

The code-level implementation of DQN relies on a Python library called *MushroomRL* [37], which provides a structured interface to import into projects the most famous RL algorithms. In this work, the choice of DQN depends on a couple of considerations. The first is that we need to have an agent capable to deal with a discrete action space, since we basically have only 4 boolean actions, and a continuous observation space. The second is related to the online nature of DQN, which perfectly fits with planned experiments, and its intrinsic sample efficiency. Moreover, starting with a very well-known algorithm permits to have lot of background literature behind, which can provide better strategies to build and run the agent. However, this doesn't mean that DQN is the best way to tackle this kind of problem and, surely, as we explain in Chapter 7.1, it would be interesting to study if also other agents can be adopted for this task.

DQN is integrated both with the standalone version of Epynet and with DHALSIM. The first integration is less relevant to the purpose of this work, since it is a mere workbench to test the behaviour of the agent and the effects on the environment, like a proof of concept to check the feasibility of the second experiment. However, even if with the standalone Epynet the process is quite straightforward, the integration with DHALSIM requires way more work due to the distributed nature of the framework.

Indeed, the complexity is due to synchronization and inter-process communication, which is handled by underlying libraries, like Mininet [36] and MiniCPS [20], not yet fully supported in Python 3. As shown in Figure 5.2, the algorithm is placed on the root process that starts the experiment. The tricky part is that DQN is not the one which leads the simulation, deciding when to start the computation, the amount of episodes to run and the exit condition, but he is more *passive* and is provided with data by autonomous running processes. This also require to schedule training and test in advance, so that the agent knows what kind of data will be received. The exact flow of DHALSIM experiments is the following:

1. When an experiment starts, the framework creates immediately the control agent instance;

2. The parser checks the schedule of the simulation, contained within the Config File and declared as specified in subsection 5.4.2, and creates an intermediate YAML file to store episode key parameters that will be read during its simulation. Notice that an entire experiment is composed by several episodes, usually representing the WDS analysis corresponding to simulations of a week;

3. The root process creates for each episode all the CPS environment, which is entirely deleted at the end of the single episode;

4. The CPS environment sends the first information to the agent instance and the computation of the episode starts.

When we talk about CPS environment, we are approximating a very complex structure composed by several autonomous processes which reproduce nodes of the CPS network. The principal components of this structure are the four kinds of processes used to simulate respectively the physical system, the different PLCs, the SCADA server and the attackers. Following the schema in Figure 5.2, we can see that even if the agent is conceived to run on the SCADA server, in the implementation is not placed inside the SCADA process, for two reasons: the first is that the SCADA process is destroyed and generated at each episode, while the agent has to run for all the experiments and it would be hard to configure following the process behaviour; the second is due to the Python version of SCADA process, which is the 2.7, when the agent requires Pyhton 3. Thus, the communication between the two processes relies on an SQLite3 database, both to solve synchronization issues and to exchange data. The database solution is copied by the method used to handle the communication also within the CPS environment, since PLCs, WDS and SCADA processes—all inheriting from MiniCPS and implemented in Pyhton 2.7—exchange part of the information through a SQLite database. The remaining part of the communication is managed by MiniCPS interfaces, which emulate a physical LAN and assigns to each process an IP address.

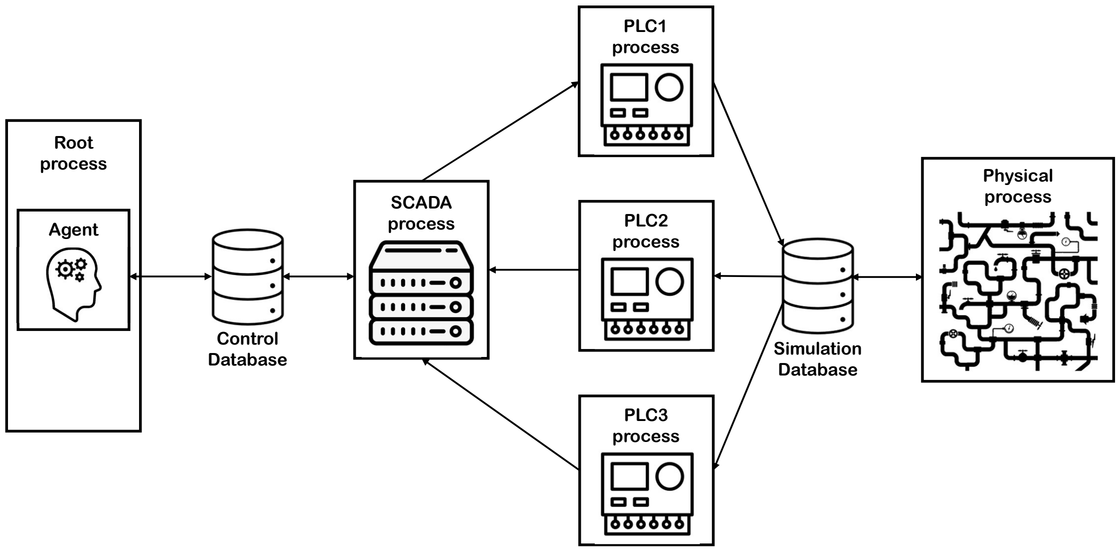

Figure 5.2: DHALSIM Processes. Here is depicted a superficial structure of processes running within DHALSIM to simulate the CPS. As can be noticed, the agent runs on the root process and exchanges data with the SCADA server by means of a SQLite database. The SCADA communicates with PLCs, by collecting information from sensors PLCs and providing actions to the actuator. The PLCs store and read data into another SQLite database that is connected to the WDS process, which retrieve and write data on it.

The agent process, to retrieve information about the system, behaves in different ways. Implementation-wise, most of the variables inside the observation space are collected through the communication with the SCADA server, by means of the so called *control database*. Some of these are the ones provided by PLCs and specified within *main* column in Table 4.1. Others, like time variables are read from an apposite table updated by the SCADA process, as well as the *done* signal and the *sync* flag. Conversely, information related to demand trend and ongoing attacks are taken from the intermediate YAML file. Moreover, data used for reward computation are collected still from the *control database* and are the ones indicated within column *secondary* in Table 4.1. Finally, after the computation of the training step, the control action chosen by the agent is stored inside a different table of the same *control database* and read by the SCADA process.

5.3 Input and Output Data

As it always happens with Machine Learning algorithms, also with this framework some initial data are required to modify the behaviour of the environment, reproducing different scenarios. This is essential to allow the agent to achieve a robust knowledge and to increase its generalization ability, preventing the risk of overfitting specific situations. At the same time, algorithms are usually designed to provide well-organized output files to foster an accurate analysis.

In this environment, we consider as input data the different weekly demand patterns, which control the trend of requested water across the WDS. In this way, we can simulate against several scenarios, both with low or high water demand. The patterns are created with a Jupyter Notebook that relies to a generic utilization table, varying in relation to the time of the day. For example, during nights we expect to have a lower request of water, since city dwellers are typically asleep, instead during the day we expect to see higher demands. The patterns are adjusted with a noise, in order to get different trends which respect a common generic behaviour.

The *training set* contains a wide range of patterns to let the agent see several scenarios, almost 315, which are randomly selected at the start of each episode. The set is equally populated by lower and higher demand patterns and is configured in order to span all feasible situations. Instead, the *test set* is made up of only 4 demand patterns, which have been accurately generated to uniformly test across the whole range of possibilities. The test demand patterns are shown in Figure 5.3 and are divided in high, middle and low demand.

As the input information, also the output ones have to be accurately selected to get suitable data to perform the experiment analysis. In addition to the .csv files related to scada values and ground truth, described in subsection 3.3.3, the control agent running on DHALSIM outputs also other two important files, both serialized with *pickle*: the saved model and the results file. The first file allows to store already trained model, saving its parameters, the policy, neural network

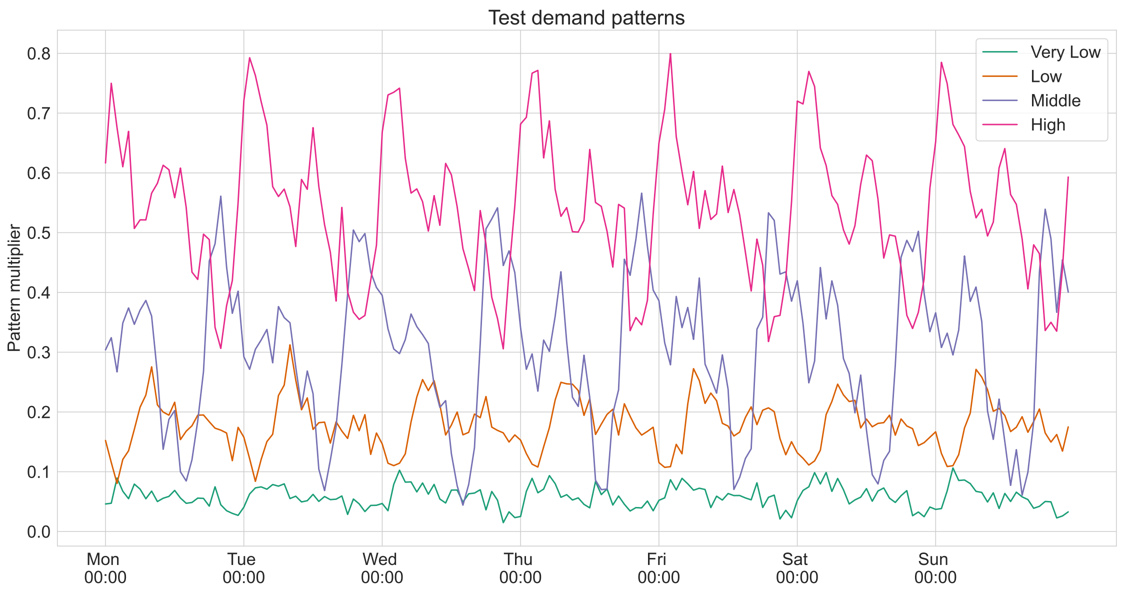

Figure 5.3: Test Demand Patterns.

weights and replay buffer. This is very important to not lose the experience got in a particular training session and it allows to load an already trained agent for further experiments. The second output file contains the results achieved with test simulations. Indeed, for each episode, it stores an evaluation metric, two datasets, one containing DQN samples and one including obtained Q-values, and the amount of total pump updates of that specific episode. As anticipated before, the evaluation metric is the total *Demand-Satisfaction Ratio* (DSR) of the WDS, which highlights if the system has been able to satisfy the request of water along the entire episode. The mathematical formulation of DSR is defined as:

$$DSR := \frac{\sum\limits_{i \in J} \frac{\sum_{t \in T} J_i^D(t)}{\sum_{t \in T} J_i^B(t)}}{|T|},$$

where T is the set containing the frames of the episode, J is the set of junctions and J^D and J^B represent the delivered and requested water respectively. The DSR is bounded between 0 and 1 and it highlights a better result proceeding towards the unit.

5.4 Experiment Configuration

In this work, we investigate the optimal real-time control problem with two different configurations. The first aims to understand if the agent can learn how to behave in presence of external modifications, namely the cyber-attacks, without having information about the forgery. In this setting, the agent is trained in normal operating conditions. Then, the evaluation is done both in normal operating conditions and also undergoing specific attacks. The purpose is to understand whether the policy found by the agent in a not compromised scenario can be resilient also in case of alterations of observation variables. In fact, the attacks, as explained in subsection 5.4.3, concern the modification of the tank water levels, which can be increased, simulating an overflow, or decreased, as in presence of pressure drops. In this first setting, we focus both on retrieving good results in normal operating conditions and on studying the agent's responses under concealed data corruption. However, we do not expect to see a particularly resilient policy in this second scenario, since the agent is provided with fake data and it is totally unaware of the attacks.

The second configuration, instead, aims exactly to understand what the agent can learn if perfectly informed about data corruption. Hence, we add to the observation space a boolean value, stored in the *under_attack* variable, which simulates the suggestion of an ideal anomaly detector running in parallel with the agent. If the attack is present in the current state of the system, the variable is set to 1, otherwise it remains 0. Moreover, in this setting, the agent is trained with episodes that include the cyber-attacks and with others in normal operating conditions, to reproduce a scenario as real as possible and let the agent to acquire a more generalized knowledge. The attack events injected in the training episodes can differ in term of number of attacks in a week (between 1 to 4), corrupted data (T41 or T42), entity of the corruption (overflow or pressure drops) and time of the event. Instead, they do not change in term of duration of the attack (always 18 hours). Finally, the test simulations are performed with the same exact attack events used in the first configuration, but, differently from that, are not analyzed in normal operating conditions.

5.4.1 Simulator Parameters

The simulator parameters are mainly related to time options for the analysis of the water distribution system. Those parameters are the common variables that can be set in EPANET GUI and that are already provided in the .inp file. Hereafter, we enlist those parameters in Table 5.1. Notice that the parameters are the same for both the configurations, since the simulator settings are independent from external data corruption.

Parameter	**Value**
Episode duration	1008 iterations (one week)
Hydraulic Timestep	600 sec (10 min)
Demand Model	PDD
Tank Level Bounds	(0, 10.668)

Table 5.1: Epynet Parameters.

5.4.2 Agent parameters

Here we report the agent settings used across both configurations. In this case, only the MDP information changes, since, in the second scenario, with the informed agent, the observation space is slightly different from the previous one. Indeed, as we can see in Table 5.2, the Informed Agent has also the information provided by the perfect anomaly detector. Besides MDP information, DQN needs also other

Configuration	**Observation Space**	**Action Space**
Uninformed Agent	Time, Day, T41_level, T42_level, J20_level, demand_SMA	[0], [1], [2], [3]
Informed Agent	Time, Day, T41_level, T42_level, J20_level, demand_SMA, under_attack	[0], [1], [2], [3]

Table 5.2: MDP Parameters.

parameters that are presented in Table 5.3 and that do not change between the two configurations. These parameters, usually called *hyperparameters*, need to be carefully tuned to achieve the agent's optimal configuration. In this work, we do not focus excessively on hyperparameters tuning, since our objective is not to get the most accurate agent, but still we tried to set different values and we ended up choosing the ones reported in Table 5.3.

5.4.3 Attack Schedule

Here we present the schedule of attacks used in the evaluation phase. As said in the introduction of this section, the attacks are the same for the Uninformed and Informed Agent. As shown in Table 5.4, in each test episode there are four Man-In-The-Middle attacks, lasting 18 hours each. Thinking to a week structure, first two attacks happen on Tuesday at 10 am and on Thursday at 6 pm, instead the third and the fourth are performed simultaneously at 2 pm on Saturday. The schedule

Parameter	**Value**
Training length	50 episodes
Approximator	ANN with 2 hidden layers of 8 neurons each
Policy	ϵ-greedy with decay
Replay Buffer Min Size	500 samples
Replay Buffer Max Size	40,000 samples
Batch size	32 samples
Fit frequency	4 step
Target update frequency	50 steps
Optimizer class	Adam
Optimizer learning rate	0.00025
Loss Function	Smooth L1-Loss

Table 5.3: DQN parameters.

is conceived to make the attacks sufficiently disrupting, having a large window of action, and separating consecutive attacks by several hours, letting the system to restore its normal operating conditions. With Informed Agent configuration, the

Parameter	**Attack 1**	**Attack 2**	**Attack 3**	**Attack 4**
Attack type	MITM	MITM	MITM	MITM
Start iteration	204	540	732	732
End iteration	312	648	840	840
Target sensor	PLC2	PLC3	PLC2	PLC3
Corrupted tag	T41	T42	T41	T42
Injected value	0.1	10	0.1	0.1

Table 5.4: Test-Attacks Schedule.

attacks injected in the train dataset have the same structure of those conceived for test episodes, but are created randomly at runtime. Notice that in test attacks, but also in train attacks, injected values tend to be always at the extreme bounds of the variables so as to reproduce the limit conditions of overflow and pressure drop.

Chapter 6

Evaluation and results

In this chapter, we analyze results obtained running the experiments with both the aforementioned configuration, namely with Uninformed and Informed Agent. To exhaustively explore the simulations, we include relevant plots, providing accurate explanations of the observed data. Then, we report some statistics to highlight also DHALSIM improvements obtained by means of the integration of Epynet. In the end, we describe software and hardware components that have allowed this research.

6.1 Uninformed Agent

As explained in Section 5.4, the Uninformed Agent is the first of the two configurations explored in this work. As the denomination suggests, this setting considers an agent with no clues regarding alterations of data perpetrated by external attacker. Moreover, the training is performed only in normal operating conditions, so the agent is totally unaware of this kind of experience.

In this section we explore the results obtained in normal operating conditions and undergoing cyber-attacks. We want to verify if the control agent is able to solve the pump scheduling problem in an uncorrupted environment and to understand whether it can adapt also to a situation of injected modifications of data. To do so, we draw some tables containing relevant information to grasp the goodness of the agent's behaviour and we plot interesting charts to analyze the evolution of the experiment through time.

6.1.1 Normal Operating Conditions

A system in normal operating conditions is a system that can evolve and transform following its regular behaviour, without any external perturbation. Thus, we can say that this setting represents almost an ideal situation, and it is important to be studied to achieve a better understanding of the principles that govern the evolution of the environment.

In Table 6.1 we can notice the first results related to the Demand-Satisfaction Ratio (DSR), which has a good outcome in all the four test demand patterns. In the table, are also reported the number of updates made by both pumps and considered in the objective function, as explained in subsection 5.1.3, and the risk of overflow, graded with a label among *high*, *moderate* or *low*. The evaluation of the overflow risk can be done easily by observing tanks behaviour in Figures 6.1, 6.2, 6.3 and 6.4.

Demand Pattern	**DSR**	**Pump updates**	**Overflow Risk**
Very Low	0.864	33	High
Low	0.864	29	High
Middle	0.859	33	Moderate
High	0.854	25	Low

Table 6.1: Uninformed Agent: results in normal operating conditions.

Analyzing the plots in Figures 6.1 and 6.2, related to the first two test patterns with respectively *very low* and *low* demand, we can notice that the agent struggles to avoid overflow, since the water request is too small and the tanks are never emptied. From the charts, we can also infer that the pumps should be turned off for longer periods, to let the emptying of the tanks. However, a wiser look to the third plot suggests that the agent continues with this behaviour even if he's having some problems, since the cumulative reward is constantly decreasing because of the overflow penalty. The reason is that the agent is afraid to take a riskier action, updating pumps status and getting a worse immediate reward, and it proceeds with this myopic behaviour. This way of acting is probably due to a lack of training, that has not allowed enough exploration to see also this kind of situations. Actually, we can expect this kind of happening in our experiments, since, as we point out in Section 7.1, all the simulations have been performed with too few samples and episodes, due to the lack of computational resources.

Instead, very different are the situations in Figures 6.3 and 6.4, where the agent gains a constantly growing reward and is able to maximize the DSR without having too much problems with tanks overflow. This behaviour could have been deducted also from the fact that we are in presence of higher water demands, which foster the emptying of tanks, with a consequent lower risk of overflow. The pumps still tend to stay open quite across the entire week, but this time it is a positive fact since they are necessary to avoid pressure-drop issues.

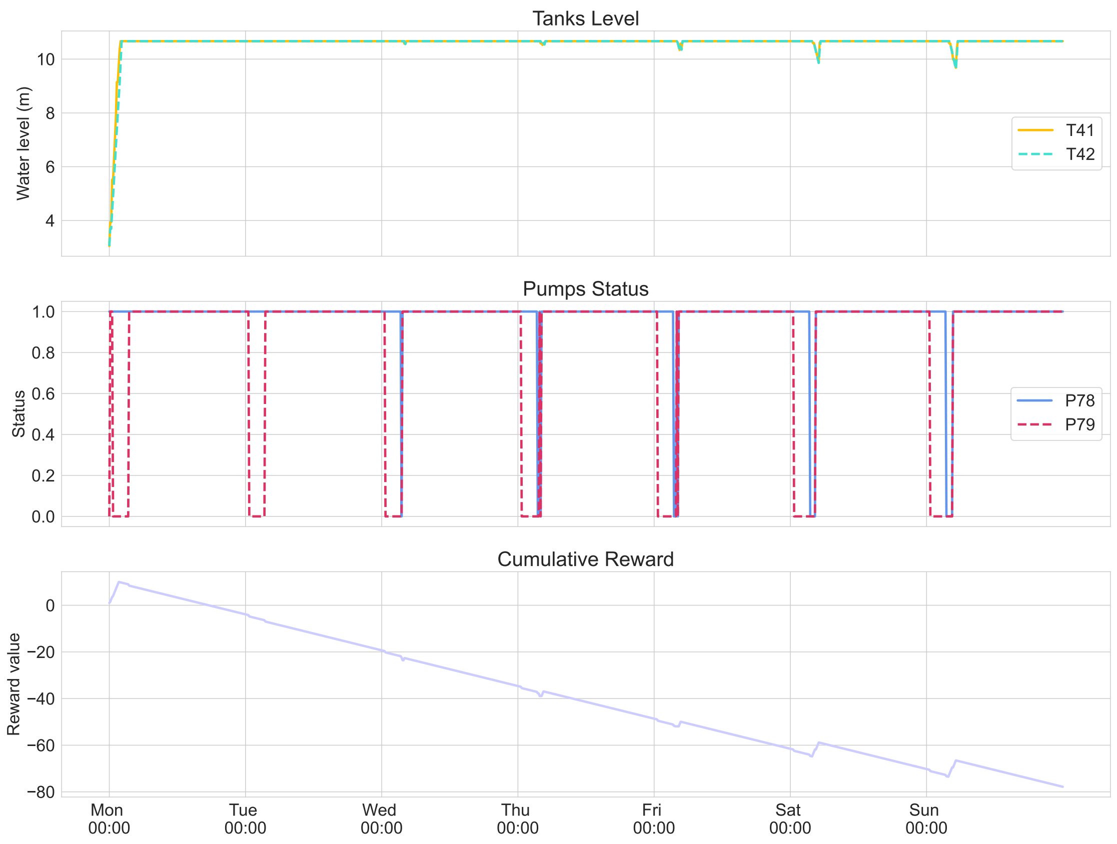

Figure 6.1: Uninformed Agent in normal operating conditions: very low demand.

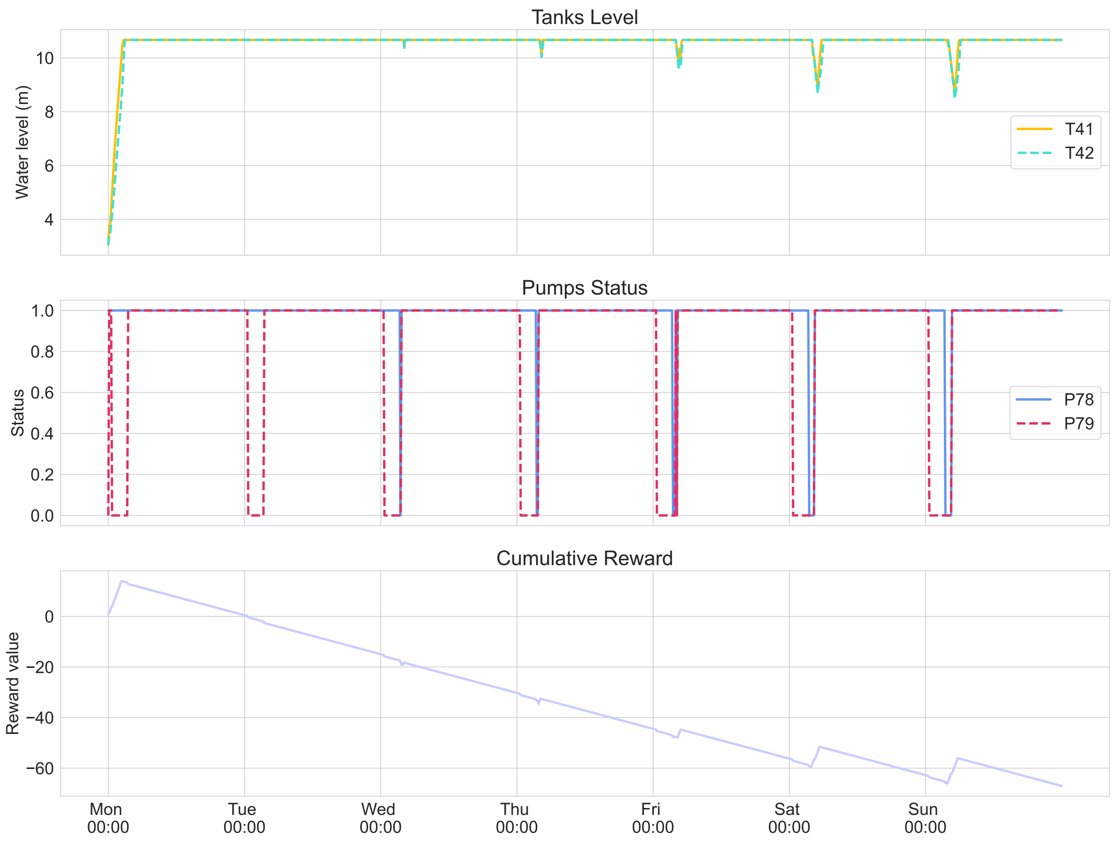

Figure 6.2: Uninformed Agent in normal operating conditions: low demand.

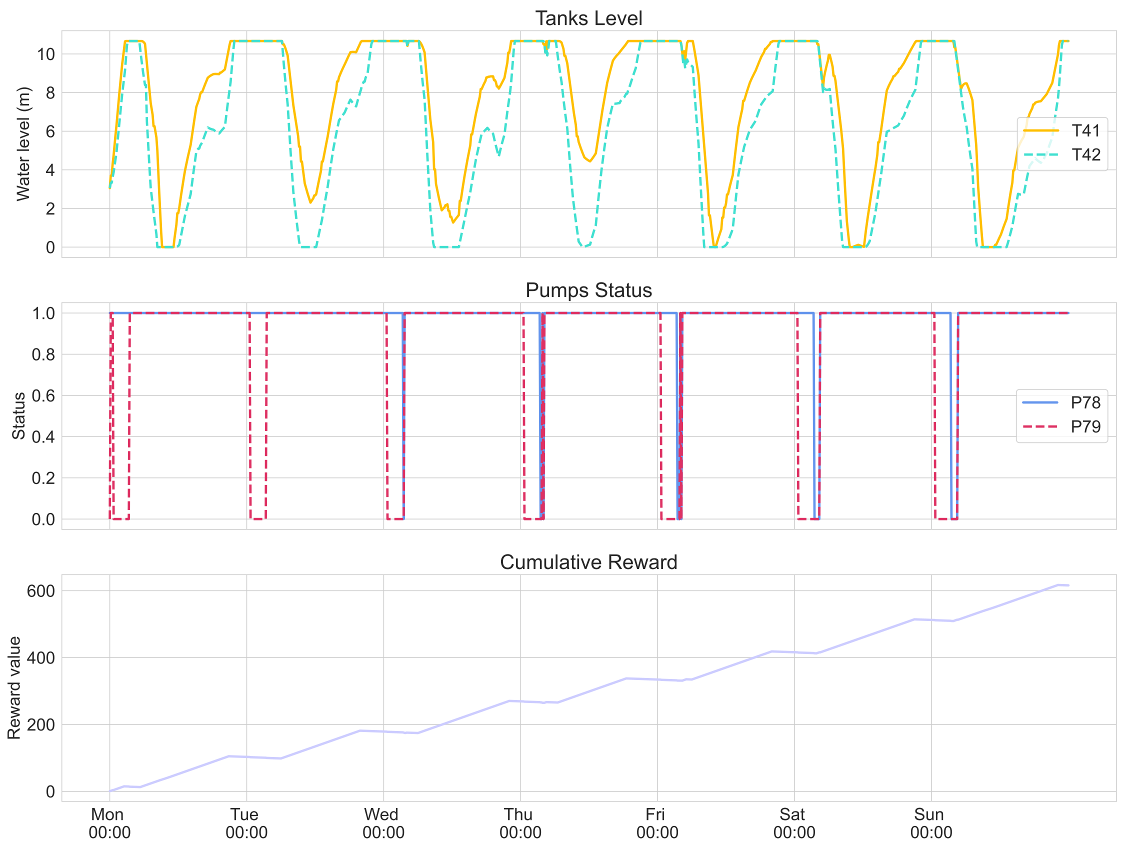

Figure 6.3: Uninformed Agent in normal operating conditions: middle demand.

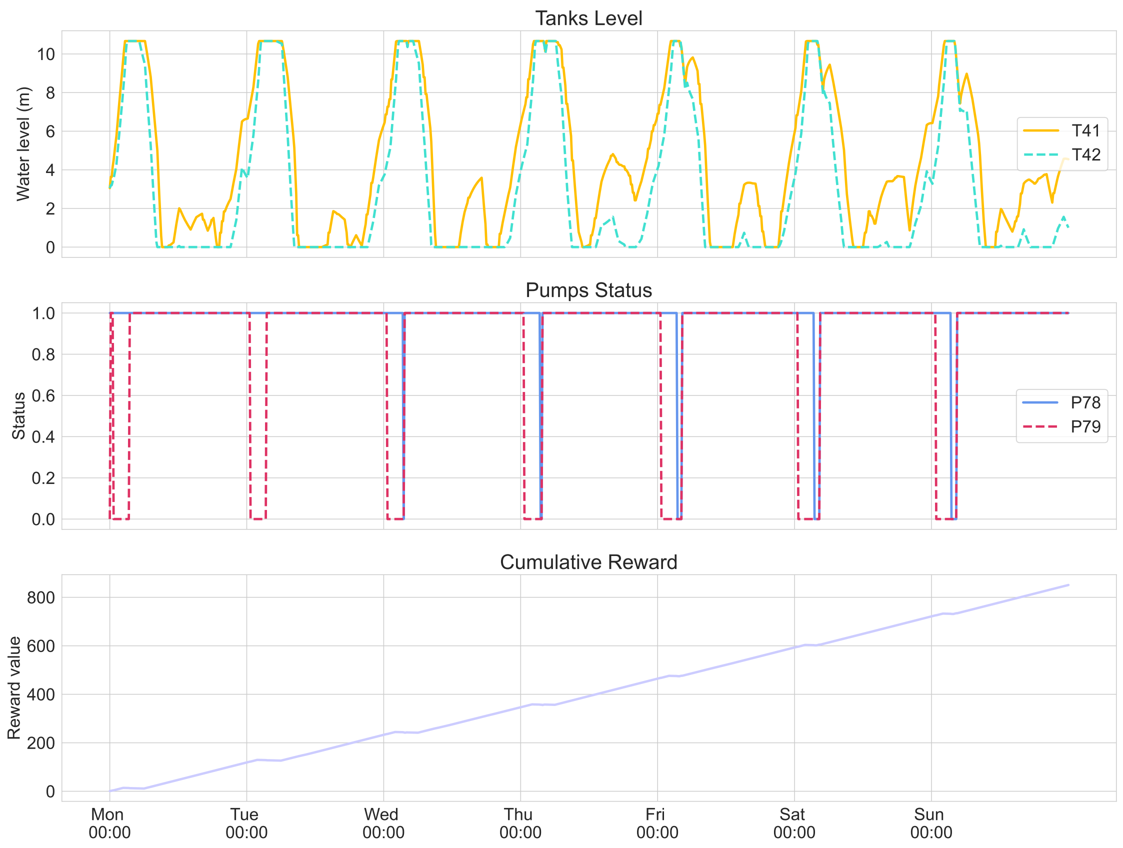

Figure 6.4: Uninformed Agent in normal operating conditions: high demand.

6.1.2 Undergoing Attacks

After the evaluation of normal operating conditions, we want to analyze what happens injecting corrupted data during test simulations. The performed attacks are the ones described in subsection 5.4.3, which are held also in the configuration with the Informed Agent. With the following results we want to understand if the trained Uninformed Agent can be reliable even in presence of unseen attacks, that cause an alteration of observed environment information.

In Table 6.2, we can see DSRs in line with those registered in normal operating conditions. However, we cannot consider the DSR a reliable measure in this case since the agent is fooled by an attacker and the computation of the metric is affected by corrupted data, even if for only a part of the entire episode. Finally, also pump updates and overflow risk across the different test demand patterns are similar to the values of previous experiment.

Demand Pattern	**DSR**	**Pump updates**	**Overflow Risk**
Very Low	0.864	29	High
Low	0.863	25	High
Middle	0.86	25	Moderate
High	0.854	25	Low

Table 6.2: Uninformed Agent: results undergoing attacks.

Delving into the analysis of Figures 6.5, 6.6, 6.7 and 6.8, we can study how the agent reacts during periods with alterations of tank level—corresponding to gray regions—, analyzing the received reward that is modified in accordance with data concealment events.

In the first two pictures 6.5 and 6.6, representing *very low* and *low* demand respectively, we can see that the agent is totally fooled by the combination of attacks 3 and 4—check attack schedule subsection 5.4.3—in correspondence with the third gray area, where the cumulative reward grows quickly since is not registered any risk of overflow. Moreover, charts highlighting pumps updates are very similar to those reported in normal operating conditions, which means that the agent doesn't recognize the different scenario in which has been placed.

Regarding the other two tests depicted in Figures 6.7 and 6.8, the results improve in term of fewer overflow events and higher cumulative reward, but still this doesn't depend on the agent's ability to recognize attacks and adequately react, rather on the increment of requested water that fosters the emptying of the tanks and the enhance of the DSR. Conversely from experiment in normal operating condition, even if the problem of weak training is still present, we don't expect that with more samples the situation could change, since the agent is not a meta-learner and cannot react to data corruption without insights on attack events.

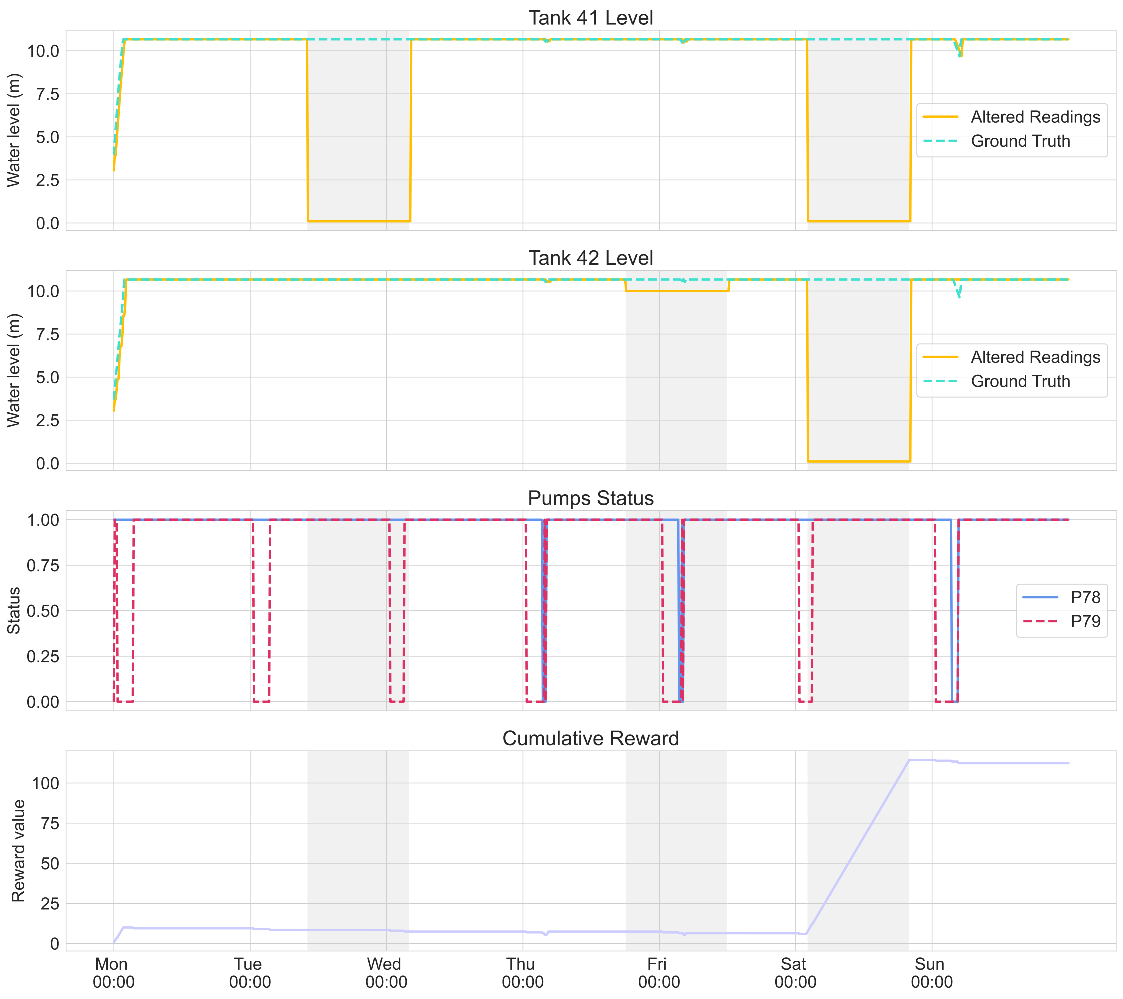

Figure 6.5: Uninformed Agent undergoing attacks: very low demand.

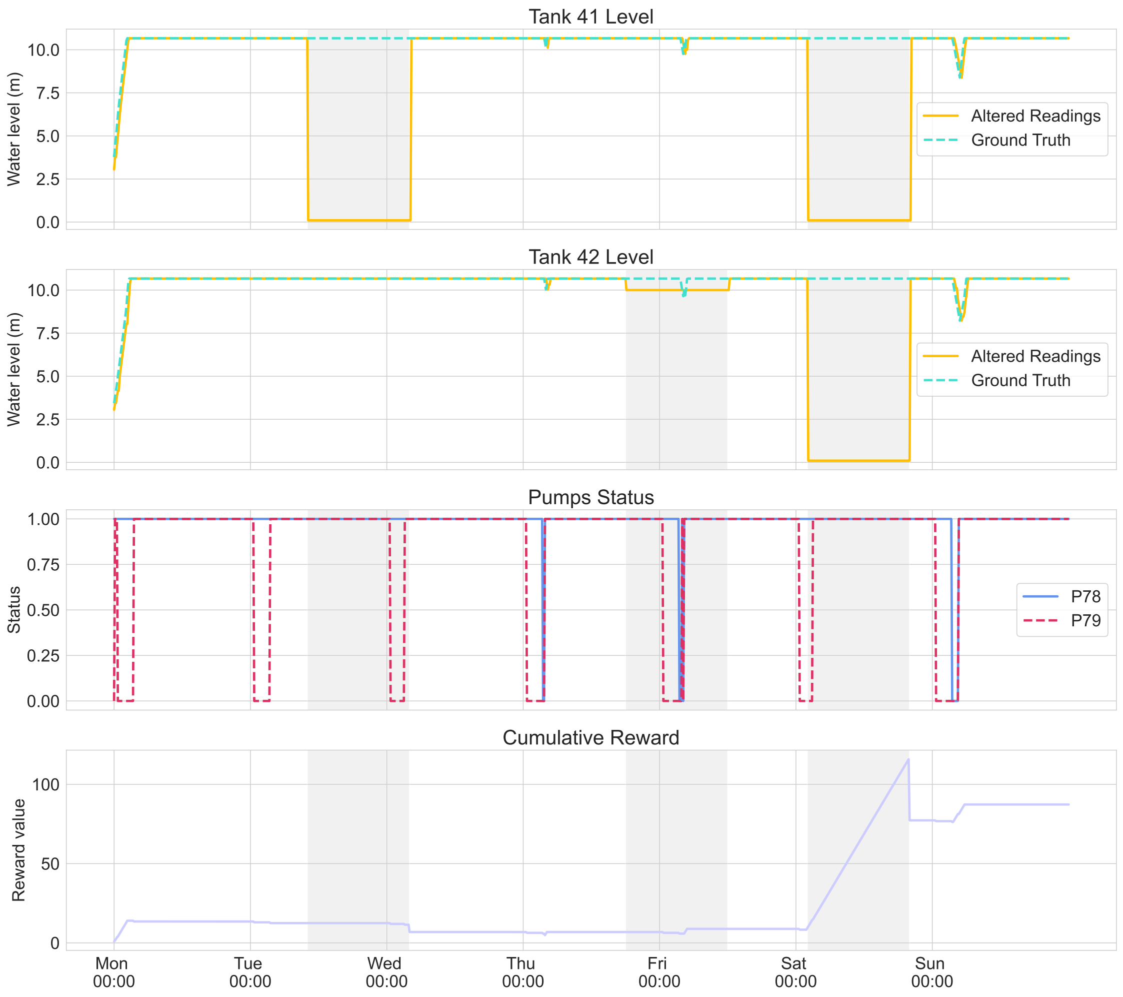

Figure 6.6: Uninformed Agent undergoing attacks: low demand.

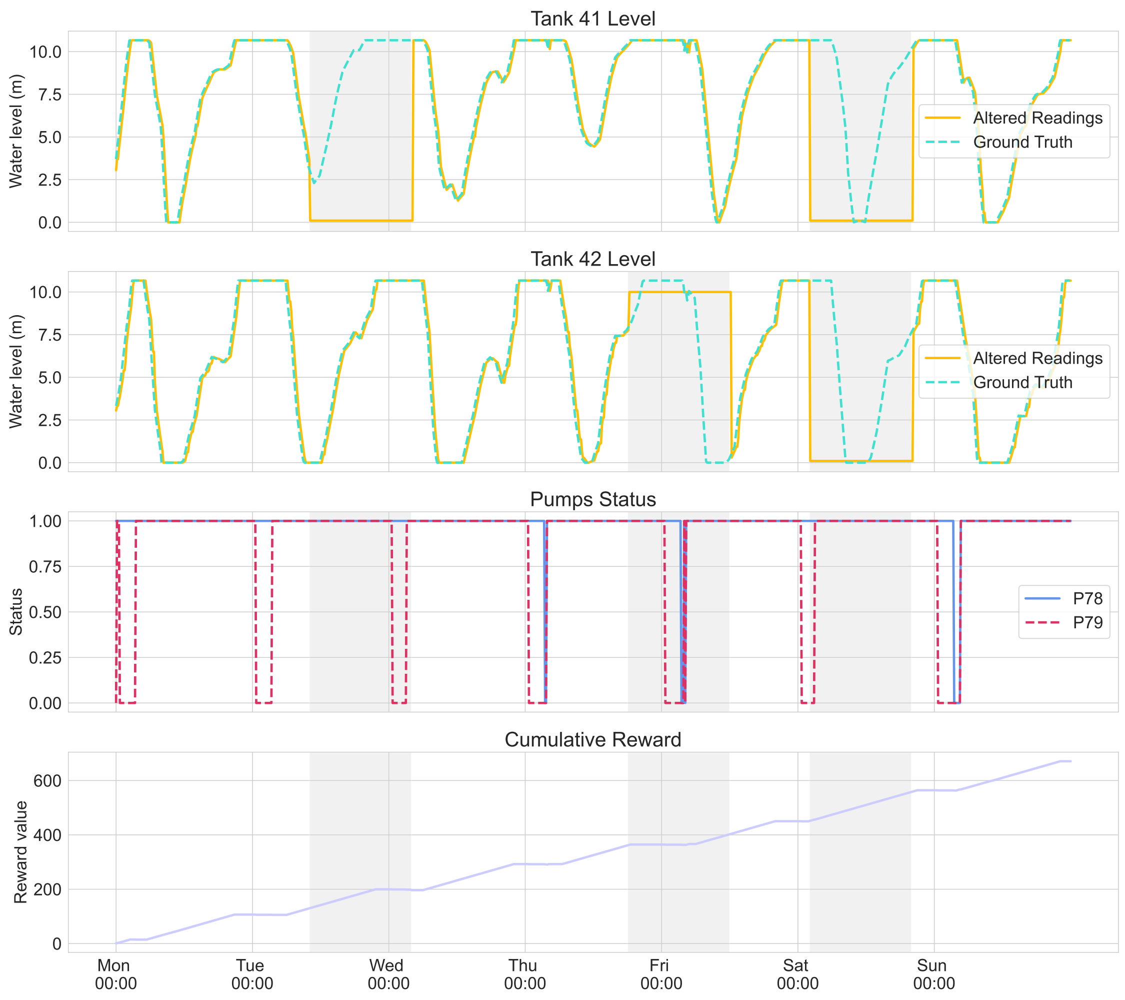

Figure 6.7: Uninformed Agent undergoing attacks: middle demand.

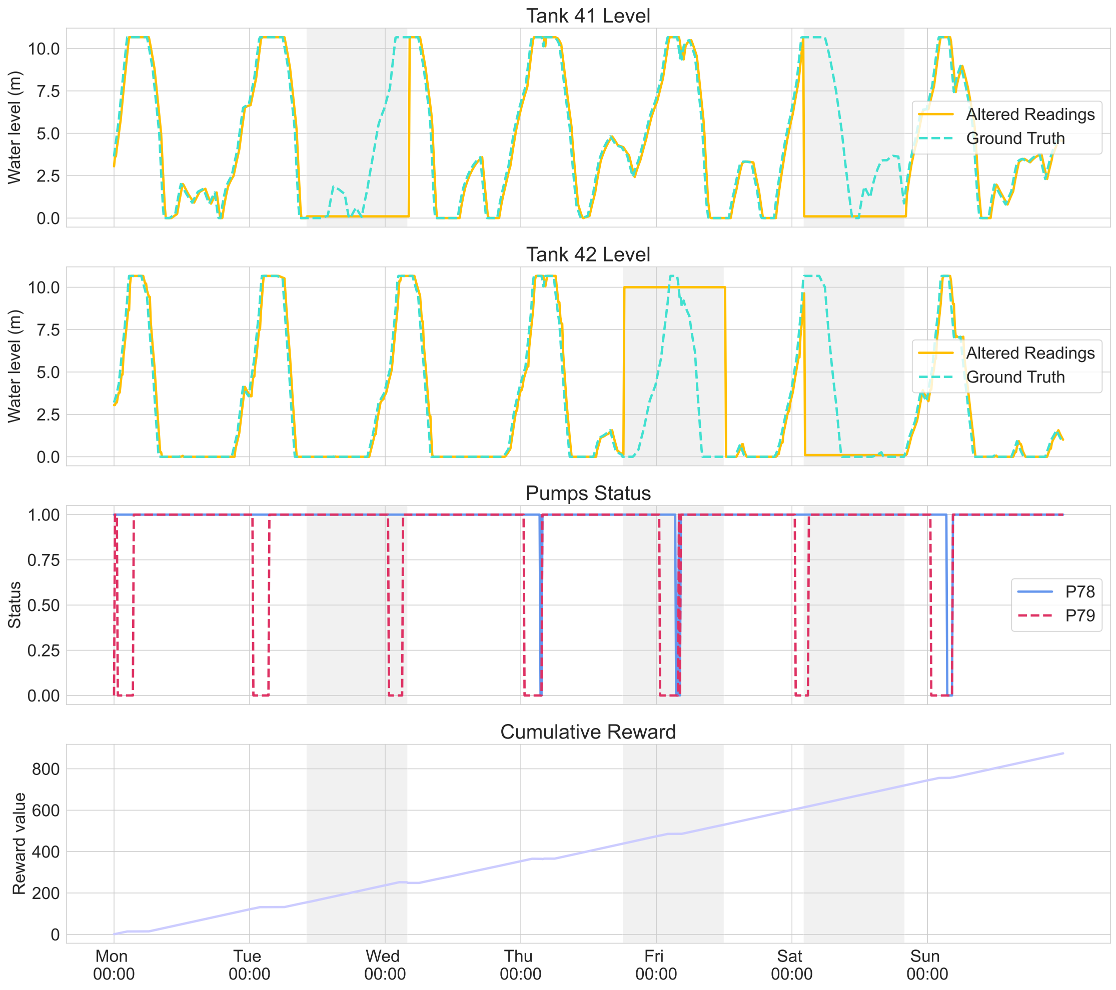

Figure 6.8: Uninformed Agent undergoing attacks: high demand.

6.2 Informed Agent

The Informed Agent, as explained in Section 5.4, consists in an agent which is aware of ongoing attacks that are perturbing the system. Indeed, in this setting we imagine to run a perfect anomaly detector in parallel with the RL algorithm that provides information about the presence of an attacker, without saying where it is striking. During the training phase, the agent sees different type of episodes, some poisoned with attacks and other in normal operating condition. The reason is that the agent has not to get used to attack events, but need to acquire a solid capacity of generalization.

The results, reported in Table 6.3, are pretty good in terms of DSR and overflow risk. The agent seems to have understood how to respond to water demands without risking too much to overcome the maximum tank level.

Demand Pattern	**DSR**	**Pump updates**	**Overflow Risk**
Very Low	0.953	5	Low
Low	0.947	20	Low
Middle	0.931	22	Low
High	0.902	18	Very Low

Table 6.3: Informed Agent: results undergoing attacks.

Analyzing the Figures 6.9, 6.10, 6.11 and 6.12 can be notice that, conversely with previous configuration, the agent is slightly more reliable with *very low* and *low* demands, and especially with the lowest test pattern gains a lot of cumulative reward avoiding to perform too many pump updates. Also in all the scenarios, the agent seems to have learnt how to not risk overflow of tanks and tends to use only one pump at the time. With this expedient can handle the data corruption caused by attackers and keeps a robust behaviour. Actually, it looks like that the agent doesn't see or doesn't care about the attacks at all and it could be interesting to investigate this aspect with a longer training session and more disrupting attacks, also to understand if the boolean variable considered in the observation space is really effective. However, in general the results got after injecting attacks in the training data are really better that ones achieved by the Uninformed Agent.

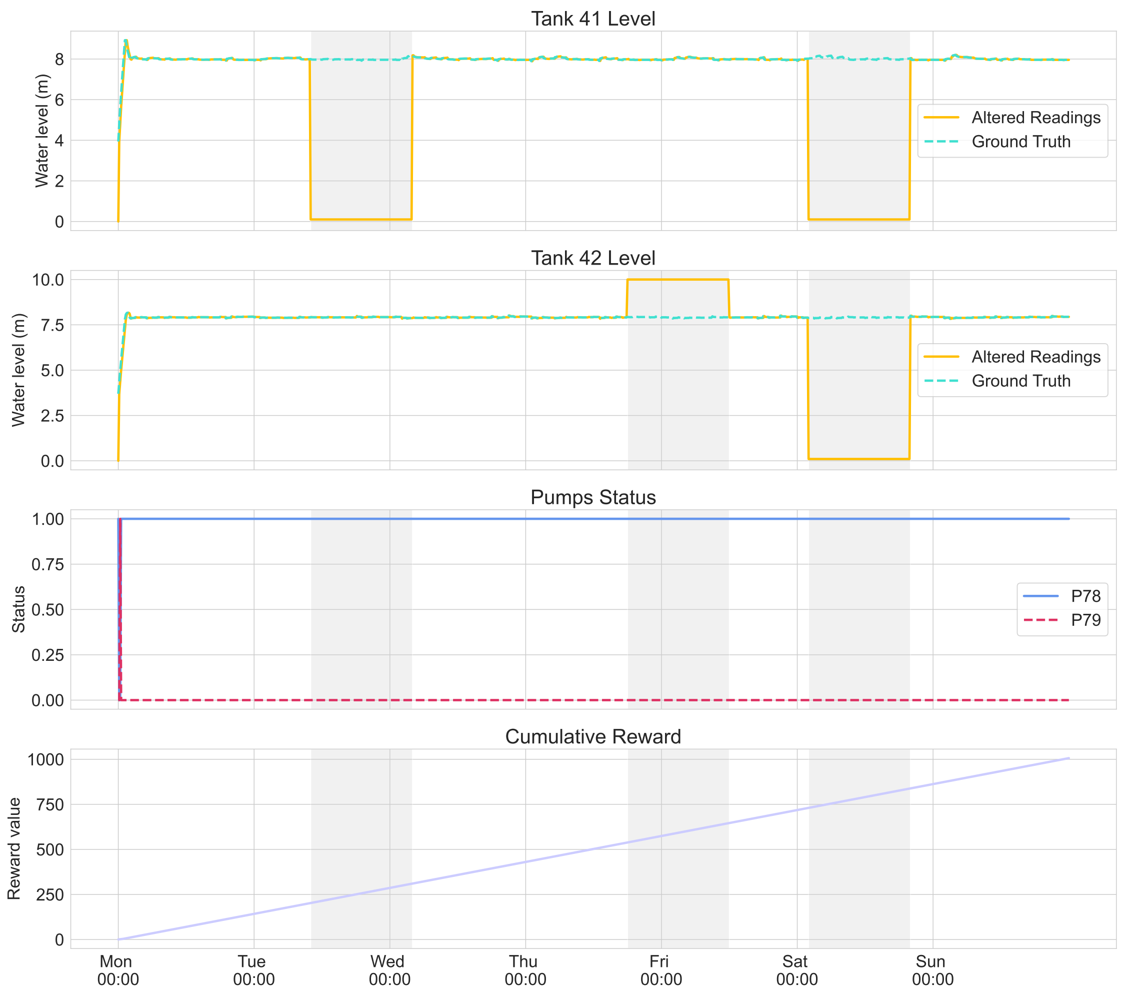

Figure 6.9: Informed Agent undergoing attacks: very low demand.

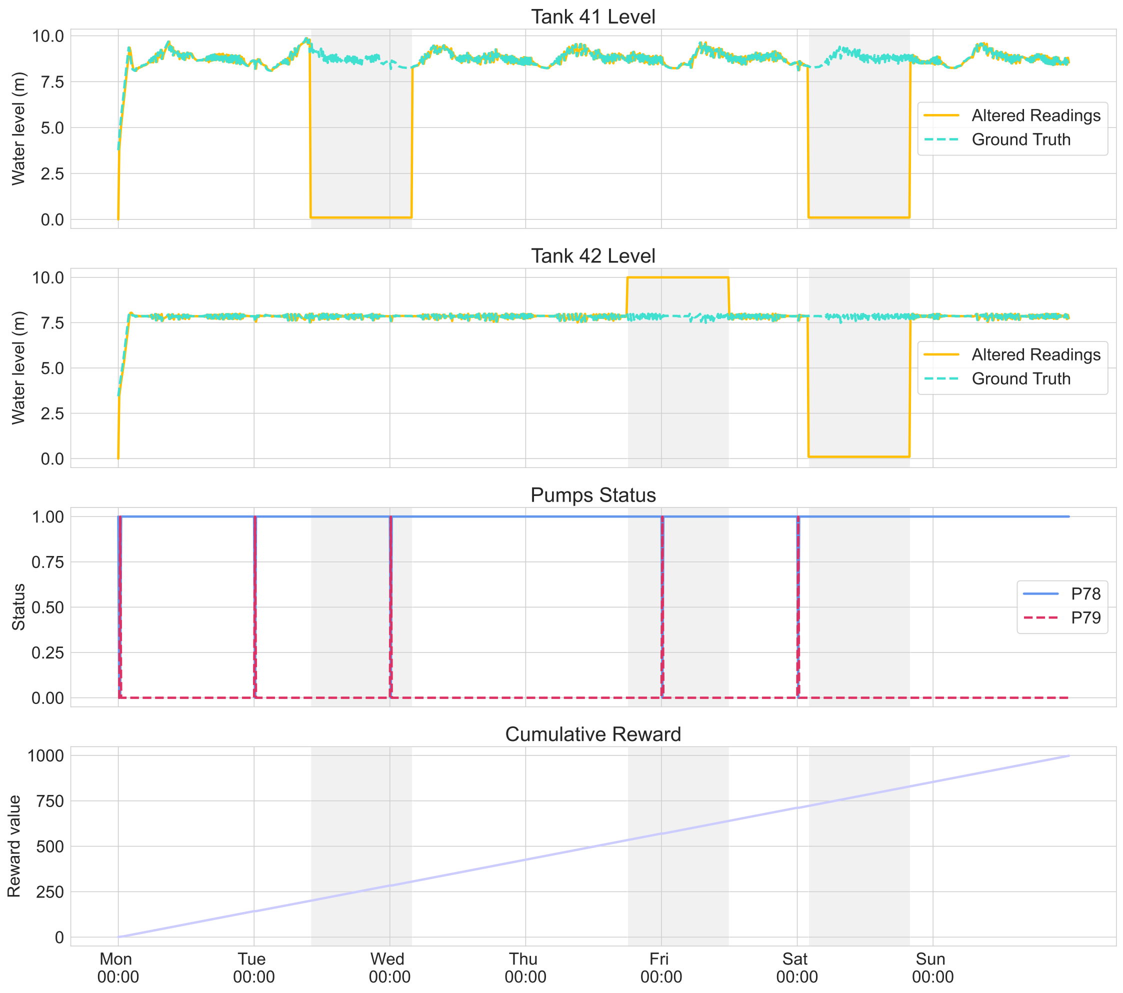

Figure 6.10: Informed Agent undergoing attacks: low demand.

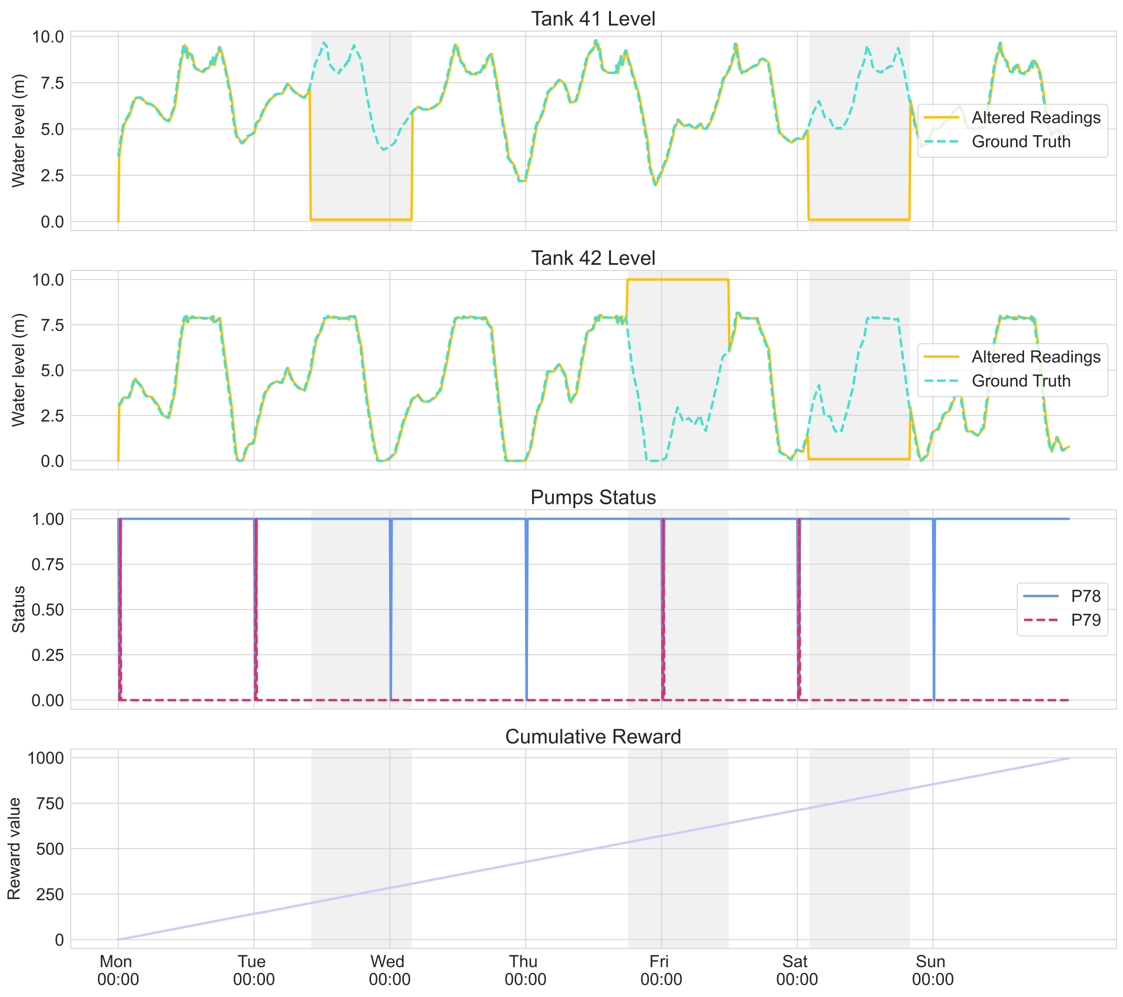

Figure 6.11: Informed Agent undergoing attacks: middle demand.

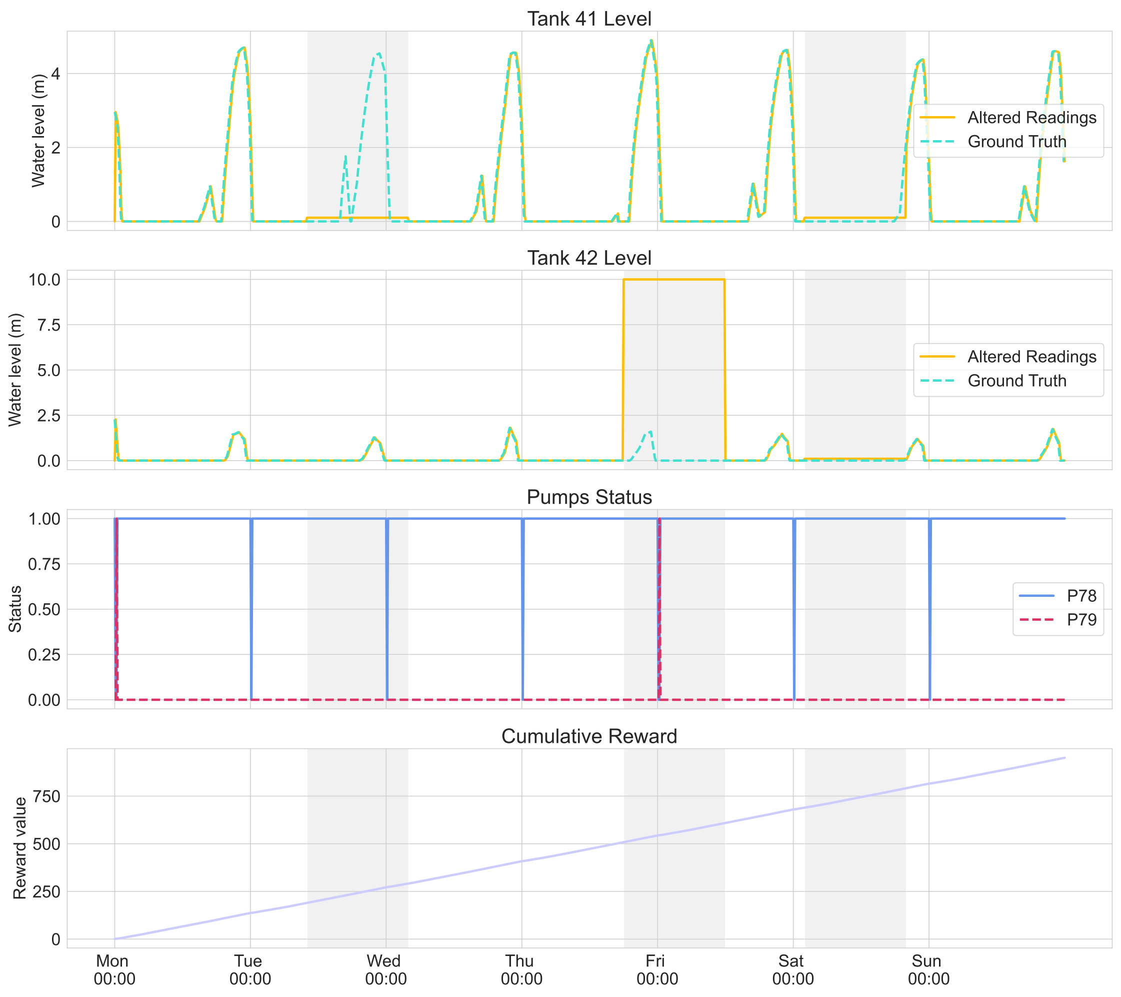

Figure 6.12: Informed Agent undergoing attacks: high demand.

6.3 Epynet Performances

We report in this section the comparison in term of performances between using DHALSIM with WNTR or Epynet as simulator. As can be noticed in Table 6.4, we have collected results coming from different experiment in which the control agent was disabled. The outcomes are pretty clear and justify the integration of Epynet in DHALSIM, not only to hold up the control agent, but also to run faster simulations. The shown data have been collected by the Singapore VM, whose hardware components are specified in Section 6.4, and the experiments have been launched with the same network parameters and time options both for WNTR and Epynet. Also the outcomes, in term of computed network properties, are the same.

Simulator	**Start time**	**End Time**	**Topology**	**Required Time**
WNTR	09/11 08:50	09/11 09:02	Anytown(1 day)	12 mins
Epynet	10/11 15:23	10/11 15:32	Anytown(1 day)	9 mins
WNTR	06/07 14:01	06/07 20:12	Ctown (10 days)	6 hours, 11 mins
Epynet	08/07 04:20	08/07 09:40	Ctown(10 days)	5 hours, 20 mins

Table 6.4: WNTR-Epynet Performance Comparison.

6.4 Software and Hardware Components

The software employed in this work includes:

- PyCharm IDE 2021.1.3 Professional, used for the implementation phase;
- Git bash, as version control system;
- Jupyter Notebook, to analyze results and draw relevant plots;
- VirtualBox, to emulate the Ubuntu 20.04 environment, required to run DHALSIM experiments;
- putty, to connect via SSH with the device where the framwork is deployed;
- WinSCP, to transfer results and input files via SFTP between local computer and remote server.

Presented experiments, and all DHALSIM simulations launched to achieve the results presented in this thesis, have been run either in a virtual machine locate in a desktop PC in Singapore, thanks to the collaboration of dr.Murillo, or in a laptop, containing a partition of Ubuntu 20.04, employed as a server and located at home.

The hardware components of the configurations are:

- Singapore VM:
  - CPU: Intel(R) Core(TM) i7-9700 @ 3.00GHz, 8 Core(s)
  - RAM: 8GB
- Home laptop:
  - CPU: Intel(R) Core(TM) i7-4710HQ @ 2.50GHz, 4 Core(s)
  - RAM: 8 GB

The experiments launched from the Home Laptop required an average of 1h and 40min per episode, since the Singapore VM is more faster and it took 40min per episode. However, an heavy limitation of Singapore VM is the impossibility of receiving connections from Italy, since cannot be released a suitable VPN. To overcome this problem is required to have a person living there that physically runs experiments. Moreover, the desktop used to run the mentioned VM is not always available, since it is used also for other studies.

The implementation of this work can be found at this link: `https://github.com/afmurillo/DHALSIM/tree/dev-dqn`.

The standalone version of Epynet used as workbench is here: `https://github.com/Daveonwave/msc_thesis`.

Chapter 7

Conclusions

In this work, we investigated the effectiveness of an approach based on Reinforcement Learning concepts to enhance the resilience of a water distribution system both in normal operating conditions and undergoing cyber-attacks. The problem has been conceived as an optimal real-time control problem, which requires an accurate scheduling of the pumps status within each scan cycle. The environment has been reproduced using DHALSIM, a digital twin able to simulate the Cyber-Physical System of the hydraulic network, integrated with Epynet, a Python wrapper of EPANET simulator, capable of performing an interactive stepwise experiment, essential to actuate real-time control actions. Moreover, DHALSIM has been also extended with the implementation of a control agent, the Deep Q-Network, broadening the range of features provided by the framework with an autonomous controller, able to supply reliable control actions, learnt by experiencing how the environment behaves.

The evaluation of our implementation has been performed with two different agent's configurations: one in which he is uninformed and not trained about incoming attacks and one where he already saw those kind of events and is aware about their presence. In both the scenarios, we have explained agent's strengths and limitations, highlighting that he has the potentiality to reach the optimal solution, but the lack of computational resources hinders deepened explorations, impeding to provide more satisfying results in terms of DSR, avoidance of overflow risk and resilience achieved by the system. However, considering the small training that has been performed, the agent's behaviour seem quite promising and it would be very interesting to test further configurations with suitable computational infrastructure.

7.1 Limitations and Future Works

Finally, we want to describe the limitations of our work and point to interesting research avenues that could be planned for the near future.

A major hurdle was the limited availability of computational resources—that generates problems of time complexity. Indeed, to get results, we often had to wait at least a couple of days for a simulation, which would be prohibitive without the double hardware setup. Related to the previous limitation, another lack of this thesis concerns the results, which, perhaps, could have been also better—in term of satisfaction of the demand and avoidance of overflow events—with a bit more of tuning, impeded by the available computational resources. Another important limitation is derived by the intrinsic structure of DHALSIM—and thus by the complexity of the distributed system—that presents different versions of Python and a pretty rough inter-process communication, which leads to longer waiting times and synchronization obstacles. This depends mainly from the underlying libraries (Mininet and MiniCPS), which would require a suitable update, and also from the difficulty at code level to grasp the entire structure of the system.

Interesting experiments that could be assessed in future works are certainly related to the employment of different RL algorithms to implement the control agent. Moreover, it would be very fascinating to see how the system behaves in the presence of distributed agents working on top of each PLC, in a partially observable scenario. Indeed, in this work, the agent, being directly connected to the SCADA server, is located in a centralized position and can know everything about the environment. Another intriguing topic, related to the Informed Agent configuration is to run in parallel a detection algorithm, which can provide the *under_attack* variable of the observation space. Finally, it would be really interesting to test the system with different attacks and to implement a meta-learning agent, which should be able to generalize to new tasks or new environments never encountered during the training process, such as other disrupting events, like a pipe burst.

Bibliography

[1] Wolfgang Wahlster. "From industry 1.0 to industry 4.0: Towards the 4th industrial revolution." In: *Forum Business meets Research.* 2012.

[2] Michael Rüßmann, Markus Lorenz, Philipp de Sousa Gerbert, Manuela Waldner, Jan Justus, Pascal Engel, and Michael J. Harnisch. "Industry 4 . 0 : The Future of Productivity and Growth in Manufacturing Industries April 09." In: 2016.

[3] *Industria 4.0 in Italia: un mercato da oltre 4,5 miliardi €.* Oct. 2021. URL: `https://www.osservatori.net/it/ricerche/comunicati-stampa/industria-4-0-italia`.

[4] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. "An Internet-wide view of ICS devices." In: *2016 14th Annual Conference on Privacy, Security and Trust (PST).* 2016, pp. 96–103. DOI: `10.1109/PST.2016.7906943`.

[5] Positive Technologies Security. *Information security risks at industrial companies.* Sept. 2021. URL: `https://www.ptsecurity.com/ww-en/analytics/ics-risks-2021/`.

[6] Claroty. *Claroty Biannual ICS Risk & Vulnerability Report: 2H 2020.* 2021. URL: `https://security.claroty.com/biannual-ics-risk-vulnerability-report-2H-2020`.

[7] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P. Mathur. "WADI: A Water Distribution Testbed for Research in the Design of Secure Cyber Physical Systems." In: *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks.* CySWATER '17. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2017, pp. 25–28. ISBN: 9781450349758. DOI: `10.1145/3055366.3055375`.

[8] Andres Murillo, Riccardo Taormina, Nils Tippenhauer, and Stefano Galelli. "Co-Simulating Physical Processes and Network Data for High-Fidelity Cyber-Security Experiments." In: *Sixth Annual Industrial Control System Security (ICSS) Workshop*. 2020, pp. 13–20.

[9] Sharon Weinberger. "Computer security: Is this the start of cyberwarfare?" In: *Nature* 474 (2011), pp. 142–145.

[10] Kevin Hemsley and Ronald Fisher. "A History of Cyber Incidents and Threats Involving Industrial Control Systems." In: *Critical Infrastructure Protection XII*. Ed. by Jason Staggs and Sujeet Shenoi. Cham: Springer International Publishing, 2018, pp. 215–242. ISBN: 978-3-030-04537-1.

[11] Tejasvi Alladi, Vinay Chamola, and Sherali Zeadally. "Industrial Control Systems: Cyberattack Trends and Countermeasures." In: *Computer Communications* 155 (Mar. 2020). DOI: `10.1016/j.comcom.2020.03.007`.

[12] Amin Hassanzadeh, Amin Rasekh, Stefano Galelli, Mohsen Aghashahi, Riccardo Taormina, Avi Ostfeld, and M. Katherine Banks. "A Review of Cybersecurity Incidents in the Water Sector." In: *Journal of Environmental Engineering* 146.5 (2020), p. 03120003.

[13] Gaoqi Liang, Steven R. Weller, Junhua Zhao, Fengji Luo, and Zhao Yang Dong. "The 2015 Ukraine Blackout: Implications for False Data Injection Attacks." In: *IEEE Transactions on Power Systems* 32.4 (2017), pp. 3317–3318. DOI: `10.1109/TPWRS.2016.2631891`.

[14] Aditya P. Mathur and Nils Ole Tippenhauer. "SWaT: a water treatment testbed for research and training on ICS security." In: *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*. 2016, pp. 31–36. DOI: `10.1109/CySWater.2016.7469060`.

[15] Riccardo Taormina, Stefano Galelli, Nils Ole Tippenhauer, Elad Salomons, Avi Ostfeld, Demetrios G. Eliades, Mohsen Aghashahi, Raanju Sundararajan, Mohsen Pourahmadi, M. Katherine Banks, B. M. Brentan, Enrique Campbell, G. Lima, D. Manzi, D. Ayala-Cabrera, M. Herrera, I. Montalvo, J. Izquierdo, E. Luvizotto, Sarin E. Chandy, Amin Rasekh, Zachary A. Barker, Bruce Campbell, M. Ehsan Shafiee, Marcio Giacomoni, Nikolaos Gatsis, Ahmad Taha, Ahmed A. Abokifa, Kelsey Haddad, Cynthia S. Lo, Pratim Biswas, M. Fayzul, Bijay Kc, Saravanakumar Lakshmanan Somasundaram, Mashor Housh, and Ziv Ohar. "Battle of the Attack Detection Algorithms: Disclosing cyber attacks on water distribution networks." English. In: *Journal of Water Resources Planning and Management* 144.8 (Aug. 2018). ISSN: 0733-9496. DOI: `10.1061/(ASCE)WR.1943-5452.0000969`.

[16] Riccardo Taormina, Stefano Galelli, Nils Ole Tippenhauer, Elad Salomons, and Avi Ostfeld. "Characterizing Cyber-Physical Attacks on Water Distribution Systems." In: *Journal of Water Resources Planning and Management* 143.5 (2017), p. 04017009.

[17] Riccardo Taormina, Stefano Galelli, HC Douglas, Nils Ole Tippenhauer, Elad Salomons, and Avi Ostfeld. "A toolbox for assessing the impacts of cyber-physical attacks on water distribution systems." In: *Environmental modelling & software* 112 (2019), pp. 46–51.

[18] Dionysios Nikolopoulos, Georgios Moraitis, Dimitrios Bouziotas, Archontia Lykou, George Karavokiros, and Christos Makropoulos. "Cyber-Physical Stress-Testing Platform for Water Distribution Networks." In: *Journal of Environmental Engineering* 146.7 (2020), p. 04020061.

[19] Katherine Klise, Michael Bynum, Dylan Moriarty, and Regan Murray. "A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study." In: *Environmental Modelling & Software* 95 (Sept. 2017), pp. 420–431. DOI: `10.1016/j.envsoft.2017.06.022`.

[20] Daniele Antonioli and Nils Ole Tippenhauer. "MiniCPS: A Toolkit for Security Research on CPS Networks." In: *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*. CPS-SPC '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 91–100. ISBN: 9781450338271.

[21] Abel Heinsbroek. *EPYNET*. `https://github.com/Vitens/epynet`. 2016.

[22] Ahmed A. Abokifa, Kelsey Haddad, Cynthia S. Lo, and Pratim Biswas. "Detection of Cyber Physical Attacks on Water Distribution Systems via Principal Component Analysis and Artificial Neural Networks." In: *World Environmental and Water Resources Congress 2017*, pp. 676–691. URL: `https://ascelibrary.org/doi/abs/10.1061/9780784480625.063`.

[23] B. M. Brentan, E. Campbell, G. Lima, D. Manzi, D. Ayala-Cabrera, M. Herrera, I. Montalvo, J. Izquierdo, and E. Luvizotto. "On-Line Cyber Attack Detection in Water Networks through State Forecasting and Control by Pattern Recognition." In: *World Environmental and Water Resources Congress 2017*, pp. 583–592. URL: `https://ascelibrary.org/doi/abs/10.1061/9780784480625.054`.

[24] Sarin E. Chandy, Amin Rasekh, Zachary A. Barker, Bruce Campbell, and M. Ehsan Shafiee. "Detection of Cyber-Attacks to Water Systems through Machine-Learning-Based Anomaly Detection in SCADA Data." In: *World*

Environmental and Water Resources Congress 2017, pp. 611–616. URL: `https://ascelibrary.org/doi/abs/10.1061/9780784480625.057`.

[25] Hajar Hameed Addeen, Yang Xiao, Jiacheng Li, and Mohsen Guizani. "A Survey of Cyber-Physical Attacks and Detection Methods in Smart Water Distribution Systems." In: *IEEE Access* 9 (2021), pp. 99905–99921.

[26] Ahmed A Abokifa, Kelsey Haddad, Cynthia Lo, and Pratim Biswas. "Real-time identification of cyber-physical attacks on water distribution systems via machine learning-based anomaly detection techniques." In: *Journal of Water Resources Planning and Management* 145.1 (2019), p. 04018089.

[27] Riccardo Taormina and Stefano Galelli. "Deep-Learning Approach to the Detection and Localization of Cyber-Physical Attacks on Water Distribution Systems." In: *Journal of Water Resources Planning and Management* 144.10 (2018), p. 04018065. DOI: `10.1061/(ASCE)WR.1943-5452.0000983`.

[28] Noy Kadosh, Alex Frid, and Mashor Housh. "Detecting cyber-physical attacks in water distribution systems: One-class classifier approach." In: *Journal of Water Resources Planning and Management* 146.8 (2020), p. 04020060.

[29] Dragan Savić, Helena Mala-Jetmarova, and Nargiz Sultanova. "History of Optimization in Water Distribution System Analysis." In: July 2018. DOI: `https://ojs.library.queensu.ca/index.php/wdsa-ccw/article/download/11973/7536/`.

[30] Gergely Hajgató, György Paál, and Bálint Gyires-Tóth. "Deep Reinforcement Learning for Real-Time Optimization of Pumps in Water Distribution Systems." In: *Journal of Water Resources Planning and Management* 146.11 (2020), p. 04020079. DOI: `10.1061/(ASCE)WR.1943-5452.0001287`.

[31] L. Rossman. *EPANET 2 Users Manual*. Tech. rep. U.S. Environmental Protection Agency, June 2000.

[32] M Haestad, Tom Walski, Donald Chase, Dragan Savic, Walter Grayman, S Backwith, and E Koelle. "Advanced Water Distribution Modeling and Management." In: Jan. 2004.

[33] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. "Cyber-Physical Systems Security—A Survey." In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1802–1831. DOI: `10.1109/JIOT.2017.2703172`.

[34] Brendan Galloway and Gerhard P. Hancke. "Introduction to Industrial Control Networks." In: *IEEE Communications Surveys Tutorials* 15.2 (2013), pp. 860–880. DOI: `10.1109/SURV.2012.071812.00124`.

[35] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. USA: Prentice Hall Press, 2010. ISBN: 0132126958.

[36] Bob Lantz, Brandon Heller, and Nick McKeown. “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks.” In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. ISBN: 9781450304092.

[37] Carlo D’Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. “MushroomRL: Simplifying Reinforcement Learning Research.” In: *Journal of Machine Learning Research* 22.131 (2021), pp. 1–5. URL: `http://jmlr.org/papers/v22/18-056.html`.

[38] A. L. Samuel. “Some studies in machine learning using the game of checkers.” In: *IBM Journal of Research and Development* 44.1.2 (2000), pp. 206–226. DOI: `10.1147/rd.441.0206`.

[39] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[40] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: `10.1038/nature16961`.

[41] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: `1712.01815 [cs.AI]`.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing Atari with Deep Reinforcement Learning.” In: (2013). cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. URL: `http://arxiv.org/abs/1312.5602`.

[43] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning.” In: *Machine Learning* 8 (2004), pp. 279–292.

[44] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. “Trust Region Policy Optimization.” In: *CoRR* abs/1502.05477 (2015). arXiv: `1502.05477`. URL: `http://arxiv.org/abs/1502.05477`.

[45] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning." In: *CoRR* (2016). arXiv: `1602.01783`. URL: `http://arxiv.org/abs/1602.01783`.

[46] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *CoRR* (Sept. 2015).

[47] Hongming Zhang and Tianyang Yu. "Taxonomy of Reinforcement Learning Algorithms." In: *Deep Reinforcement Learning: Fundamentals, Research and Applications.* Ed. by Hao Dong, Zihan Ding, and Shanghang Zhang. Singapore: Springer Singapore, 2020, pp. 125–133. ISBN: 978-981-15-4095-0. DOI: `10.1007/978-981-15-4095-0_3`. URL: `https://doi.org/10.1007/978-981-15-4095-0_3`.

[48] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *CoRR* abs/1609.04747 (2016). arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747`.

[49] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. *Prioritized Experience Replay.* cite arxiv:1511.05952Comment: Published at ICLR 2016. 2015. URL: `http://arxiv.org/abs/1511.05952`.

[50] *EPANET-toolkit.* URL: `http://wateranalytics.org/EPANET`.

[51] Alemtsehay Seyoum, Tiku Tanyimboh, and Calvin Siew. "Comparison of Demand Driven and Pressure Dependent Hydraulic Approaches for Modelling Water Quality in Distribution Networks." In: Sept. 2011.

[52] Helena Mala-Jetmarova, Nargiz Sultanova, and Dragan Savic. "Lost in optimisation of water distribution systems? A literature review of system operation." In: *Environmental Modelling & Software* 93 (2017), pp. 209–254. ISSN: 1364-8152. DOI: `https://doi.org/10.1016/j.envsoft.2017.02.009`. URL: `https://www.sciencedirect.com/science/article/pii/S1364815216307769`.