POLITECNICO DI TORINO

Master Degree Thesis

A framework for system requirements verification in Industrial Network Systems

Supervisors
prof. Riccardo Sisto
prof. Guido Marchetto
dott. Fulvio Valenza
dott. Jaloliddin Yusupov

Candidate
Antonio Giannone

Academic Year 2018-2019

Contents

Listings

List of Figures

List of Tables

Chapter 1

Introduction

Industrial Control Systems (ICS) are undergoing a deep transformation of their communication infrastructures towards increased connectivity of devices and extreme flexibility of industrial plants. This is seen within the Industry 4.0 and Factory of the Future (FoF) frameworks.
Innovative industrial applications exploit benefits from Software Defined Network (SDN) and Network Function Virtualization (NFV) approaches; they can bring several advantages, such as a detachment from the traditional Industrial Network design and the power to administer the entire network environment from a centralized point.
As the higher flexibility of these systems will require frequent network reconfigurations, an enhanced level of automation in the management of cybersecurity will be necessary. The number of cyberattacks driven through the industrial network infrastructure is increasing; they can exploit both inadequate segregation between different network environments. Additionally, cyberattacks are increasing when the corporate network is not properly segregated from the industrial one, as well as exposed industrial systems that are potentially vulnerable. However, maintaining industrial network systems is very challenging due to their safety-critical mission inside the manufacturing process.
The following thesis provides high assurance levels, as required by the safety-critical nature of these systems, by leveraging formal models and verification. Cyberattacks may have different purposes and they can cause industrial process disruption, leakage of secrets or even stealing of money. In order to face these threats, a more complex infrastructure must be deployed. On the other hand, innovative industrial applications have strict requirements on end-to-end latency during critical events. In ICS, systems involved within the network infrastructure should regularly make a decision, report data to a centralized collector, and execute a remote command with a deterministic end-to-end delay. If the operation is not completed in a specific timeslot, all the entire process may be invalidated. The consequence could vary depending on the industrial environment type. In some environments, it can cause production line breaks and only human assistance can restore the proper system

functioning. There are also safety-critical systems where issues in industrial systems communication could cause ecological disaster or incidents that result in loss of life. Bringing innovation inside these particular systems is very hard to achieve. It is imperative to meet several requirements before deploying solutions inside real ICS.
A solution has been designed and developed to assure formal industrial requirements verification with a particular focus on real-time control systems. This is done by exploiting the innovative approach led by SDN and NFV technologies.

Thesis structure

The objective of the thesis is to define and implement a framework that allows the verification on network requirements (e.g. network reachability, real-time communication, loop-free) in Industrial Network Systems. The topic is addressed with the following partitioning in chapters:

1. **Industrial Control System**: a brief introduction on ICS infrastructure and its changes through fourth industrial revolution. It contains also a clarification of salient features of SDN and NFV solution deployed in an industrial environment, with particular emphasis on the innovative infrastructure design respect the traditional one.

2. **Related Works**: a summary of tools and already existing solutions exploited that made feasible the presented solution implementation.

3. **Thesis Objective**: a brief overview of industrial security and reachability issues with thesis work contextualization.

4. **VerINS design**: an overview of core module architecture design, showing how service graphs and physical graphs are handling inside VerINS engine and how to interact with core module.

5. **VerINS implementation**: explanation of module functioning presented as a MAX-SMT problem solution. A particular attention have been given to objective functions declaration and constraints attachment on Z3 formulation.

6. **Test and validation**: a brief test case presentation that validate the work and give some performance results.

7. **Conclusion and future works**: analysis of developed module, with its strengths and weaknesses, pointing out difficulties encountered during module implementation and focusing in what could be improved in future module development.

An appendix has been added to the thesis work to explain the interaction with the developed module. The "RestAPI developer's guide" shows to the readers how interaction with module has been standardised through a RestAPI available that allows you a great interaction experience with the developed module.

Chapter 2

Industrial Control System

With Industrial Control System (ICS), we refer to a particular slice of a company network environment. Inside this chapter will be analysed the most important features of industrial networking. In particular, the focus will be on the main difference with the standard network environment, in production in a typical general-purpose company.
Firstly, an overview of the network model design will be given to the reader. Different types of industrial control systems will be shown and analysed, with their application contexts. Segregation from the standard corporate network, industrial middle-box, protocols, and requirements in terms of both end-to-end connectivity and threats defence will be examined.
Then, the focus will be moved on innovative features on ICS, through the fourth industrial revolution. SDN and NFV deployment in the industrial network will be introduced to the reader with a general overview of the main features that distinguish this new way of viewing the networking world. There will also be a brief introduction to innovative applications in the industrial environment, such as IIoT applications and 5G integration in already existing network environments.
Devices and protocols used in an ICS are used in every industrial sector and critical infrastructure such as the manufacturing, transportation, energy, and water treatment industries [1]. On the one hand, new applications deployment enabled a meaningful evolution in the ICS environment. But on the other hand, they lead to the birth of new requirements that have to be met to ensure proper functioning of the entire system. Innovation is the keyword of the general networking environment, but it must be contextualized. An unbridled change may hurt a safety-critical environment like the industrial one.

2.1 Network Environment Overview

2.1.1 Industrial Infrastructure

There are many types of ICSs environment, regarding their purpose and functioning. The most common are Supervisory Control and Data Acquisition (SCADA) systems and Distributed Control Systems (DCS).
Now we will analyse main differences between these industrial system types. First of all, DCS are process-oriented platforms that rely on a network of interconnected sensors, controllers, terminals and actuators. A DCS allows you to control and monitor a process. It works on a closed-loop control platform, and it is ideal for one facility control. SCADA systems do not allow you to have a full control of industrial processes. SCADA systems are data-oriented, and they are capable of concentrating data acquisition without geographical restrictions from a single factory or a dozen. In general, they are more scalable and flexible. This is why they are adopted when there is the need to deploy remote monitoring platform.
Both SCADA and DCS are essential for a comprehensive automation platform to take advantage of the latest innovations in Industry 4.0 technology. Nowadays, many DCS solutions integrate SCADA systems into their operations. While SCADA and DCS platforms started to develop with different purpose, attributes from both architecture are often incorporated into a hybrid system the includes both two architecture design. They're now merging to create a unique supervisory system for all facility's needs [2].
Field Devices control local operations. They are responsible for a geographical or contextual limited area, receiving and executing supervisory commands from remote stations. They are also used to report data to a centralised monitoring system to analyse general phenomenon behaviour. For instance, sensor networks are exploited to explain the behaviour of a very different phenomenon, from the simple ones like traffic congestion in an urban area to the most safety-critical ones, like analysis of an alarming situation in a particular area. Over the decades, network connectivity of these singular devices changes a lot. The standard architecture consists of a mainframe computer located in a control room which coordinates all Field Device through a point to point connection. Limits imposed by the standard model are many. Wiring complexity, high risk related to having a single point of failure often without any recovery plan, because of its high cost, are limits all associated with the use of standard architecture design [3].
Many industries started to move to distribute Field Devices control to remote collectors, that report data towards a centralised system. So network starts to become more complex, obtaining architecture like the following one:

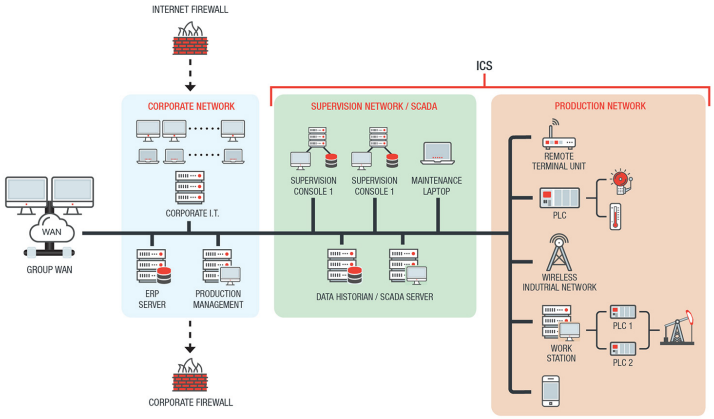

Figure 2.1. Modern industrial network architecture model

As shown in the figure 2.1, modern network infrastructure involves many different types of devices. The network which connects field devices operates with so-called Fieldbus protocol. Both device types and Fieldbus protocols will be analysed in next paragraphs.
Actually, during last year, overall network industrial architecture is changing a lot. This is due to the need to reach higher flexibility respect the one obtained through the traditional approach guaranteeing at the same time security and reachability requirements.

2.1.2 Field Device

Different special-purpose devices are involved in an ICS environment. An ICS could choose to use a subset of them accordingly to its purpose. Besides, many of them could be deployed in both two network architecture model analysed before (DCS and SDADA systems). Deployment way of these devices changes accordingly to network architecture model, but their function remains almost the same.
The most used ones in common industrial networks are:

- **Programmable Logic Controller (PLC)**: this is a type of industrial device that is deployed in both DCS and SCADA systems. It provides local processes management processing signals coming from sensors and actuators, implementing functions more or less complicated.
- **Master Terminal Unit (MTU)**: it communicates with RTUs in the same industrial field, providing commands to be executed and collecting related information.
- **Remote Terminal Unit (RTU)**: it is a microprocessor-controlled field device. It receives commands from MTU, executes them a provides results to the MTU responsible of the industrial field in which it is located.
- **Human Machine Interface (HMI)**: a graphical user interface (GUI) application that allows interaction between the human operator and the controlled hardware.

- **Intelligent Electronic Device (IED)**: a smart device capable of acquiring data, communicating with other devices, and performing local processing and control.
- **Data Historian**: it is a centralised database. It is used to log process data coming from other field devices in an ICS environment. Obviously, data that are collected by this kind of software are used by the company to analyse process and improve quality.
- **Control Server**: it hosts the DCS or PLC supervisory control software issuing commands to other industrial devices under control.

These devices operate a safety-critical mission inside overall network environment communication. They interact directly with machines issuing them commands. Environment conditions establish what command must be sent. A fault in end-to-end communication between these devices or decision process may be fatal for the entire industrial process.

2.1.3 Connectivity Requirements

From what discussed at the end of the previous paragraph, Industrial Network Systems (INS) demand that different type of requirements must have met during their design operations. Engineering a reliable industrial network is challenging. There are too many variables to be taken into account. INS features also depend on surrounding conditions of the place of installation. In adverse weather conditions, device materials have to be chosen carefully. Or if devices involved in the industrial process are far apart, end-to-end latency must be validated. In general, requirements that an industrial application has to meet are the following:

- **High security**: securing network infrastructure is a very critical process during the last years. Cyber threats over the network are increasing day by day. Industrial devices often are the main target of a cyber attack because they have a straightforward software. In many cases, it allows them only to perform their function in a very simple way. Interesting in Cyber Security is increasing a lot. Many and many INS model architectures have been proposed to fight cybersecurity threats. Among the ones, the following figure 2.2 shows how an elementary industrial infrastructure could be designed providing segregation from other network environment delimiting access from outside network.

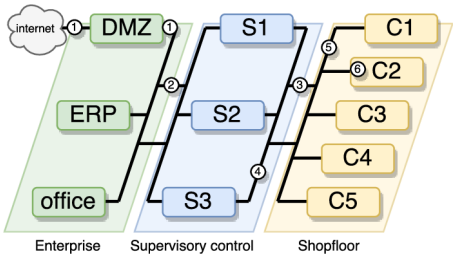

Figure 2.2. Industrial Network Hierarchy [4]

In this example, industrial environment is reachable by a limited set of devices in the upper layer of the network. It is clear from the figure 2.2 the network layering. At the higher level, there are devices used by an ordinary employee in a corporate network environment. Since they have no safety-critical mission, they are located at the forefront of network infrastructure. As moving towards lower network levels, security increase thanks to firewall policy deployment, that give access to the deepest network layer only to authorised personnel. Segregating industrial systems from the corporate one is a first example of leveraging security in INS [5].
Many different strategies to fight cybersecurity threats have been adopted, also exploiting advantages coming from Software Defined Network (SDN) and Network Function Virtualization (NFV) frameworks. They allow being more flexible against cybersecurity threats since they are often conducted with many different strategies. In the next section will be investigated how INS environment is changing concerning the deployment of these innovative frameworks.

- **Very low end-to-end latency and packet loss**: field devices during an industrial process exchange multiple messages over the network. Message integrity is critical. Adopting a reliable network protocol that gives itself this kind of feature could be a solution. But end-to-end latency between field devices must be considered. In particular industrial environment, real-time applications need that message must be delivered to the destination in a within a specific time. Message validity may depend on time. If the message would be provided after a particular instant, it will not be valid for the destination. In this way industrial process could enter in an unpredictable condition and consequences may hurt entire industrial infrastructure.

2.1.4 Fieldbus Protocols

Fieldbus protocols are the one used in field device communication. The word Fieldbus come from a combination of term "Field" and "Bus". The word "Fields" represent devices involved in industrial process. The term "Bus" stands for a line that electrically connects various units to allow data transfer and in this context represents point to point communication between devices involved in industrial transmission. There is a large variety of Fieldbus Protocols used in production INS. The most interesting ones for the thesis purpose are:

- **Modbus**: it is a layer 7 communication protocol. It builds a client/server communication model between connected field devices [6]. Through its flexibility, it can operate over many different types of network.
The Modbus protocol defines a simple Protocol Data Unit (PDU). Inside the PDU, function code has been set to deliver commands during client-server communication.
An interesting Modbus architecture is the one that allows interaction with the TCP/IP world. Today TCP/IP protocol stack is widespread and widely implemented in production devices. Moving toward TCP/IP protocol stack

could be an advantage in terms of interoperability between different devices type. Besides, it allows that field devices could be reachable through the Internet network. Despite these could be a problem in terms of security issues, on the other hand, it allows the birth of a more flexible industrial world against the past one.

- **Profinet**: Profinet is an industrial protocol that is used to provide some real-time requirements to critical applications that require them. These requirements are satisfied, providing a direct interface with Ethernet layer [7]. During years, a large number of Profinet implementation models has been proposed, with particular attention on real-time industrial applications that have a low latency requirement to be met. Among the most popular ones, the Isochronous Real-Time (IRT) Profinet implementation is the most interesting. It guarantees low end-to-end latency between industrial devices. In the next section will be analysed how this requirement has been reached.

They are interesting for this thesis purpose because the first one shows how it is possible, through its variant, building communication over the IP network between industrial field devices. The second one is an example of PTP (Precision Time Protocol) deployment to guarantee low end-to-end latency communication. PTP protocol will be exploited to make a real implementation of the solution proposed with another Intel technology (DPDK, Data Plane Development Kit).

2.2 Innovative Features

2.2.1 Industry 4.0

Industry is changing a lot. The fourth industrial follows the third industrial revolution, which started in the early 1970s. It was based on electronics and information technologies. The primary purpose was the achievement of an high level of automation in manufacturing [8]. Many market leaders are used to refer to this new industrial revolution with term "Industrial Internet" (GE, General Electric) or "Internet of Everything" (Cisco) or other variants [9].
The fourth industrial revolution changes radically:

- **technological paradigm**: digitalisation brought to many changes throughout the value chain. The business world starts to consider new challenges in term of efforts needed to face them and risks that an enterprise is willing to take. Challenge regards many industrial infrastructure fields[10]. Among them:
  - intellectual property protection;
  - personal data and privacy;
  - design and operability of systems;
  - environmental protection and health and safety.

- **social paradigm**: public must develop a new awareness and acceptance of internet-based product and service. While the "smart" objects market grows, new skills probably are required to interact efficiently with this modern society. Industry 4.0 could have a positive or negative impact on everyday people habits. Many sociological studies have been conducted to analyse industrialisation effects on the population. An uncontrolled massive commercialisation of "smart" objects that distinguish the fourth industrial revolution could have an unwanted impact on society and relationships in between private and public sectors.

- **business paradigm**: industry sustainability can change if we take into account the entire industrial scenario affected by this revolution. One of the most revolutionised fields is the manufacturing one. Now, small and medium enterprise starts to collaborate in the entire supply chain and a new business model is being developed. There is more flexibility than the previous model. It allows to minor company to enter in the global trade.
  On the other hand, for the first time, we are experiencing a lack of standardisation. This could be a problem in the case that two or more different devices have to talk each other. Open standards will be crucial in environments 4.0. One of the most critical factor is maintaining all industry production features, despite this new revolution. Without a standardisation process, benefits that could come from Industry 4.0 may be limited to local production.

Industry 4.0 changes also the production processes organization. Now it is based on technology and devices that autonomously communicating with each other, exchanging different kinds of information. A model of the 'smart' Factory of the Future (FoF) where computer-driven systems monitor physical processes has been developed. Internet of Things (IoT) connects devices, objects and people in real-time. It creates a virtual copy of the physical world and makes decentralised decisions based on self-organisation mechanisms. Integration of these technologies brought to the birth of a Cyber-Physical System (CPS). It is intended as a smart system that can organise a network of billion physical and computation components. CPSs are the heart of the fourth industrial revolution, and much research are done of their capabilities.

2.2.2 NFV and SDN Overview

Concurrently with the fourth industrial revolution, a network infrastructure reorganisation is taking place. Innovations in networking are the key enabling of the fourth industrial revolution. Network Function Virtualization (NFV) and Software Defined Networking (SDN) give the power to introduce many innovative features in a general network environment [11].

- **Network Function Virtualization**: it introduces a substantial paradigm shift breaking the traditional link between hardware and software. Today's network equipment is often designed to implement a particular network function. Network devices like firewalls or DPI (deep packet inspector) for instance, are designed to implement security network functions. Or routers

are designed to route traffic between the entire network environment. Now, with NFV, network functions are virtualised. Network operator can deploy a specific network function on a traditional server COTS (Commercial Off-The-Shelf) without the need to buy a specific device for that network function. It becomes virtualised (VNF, Virtualised Network Function). In general, NFV virtualises network services and abstracts them from dedicated hardware.

- **Software Defined Networking**: it is a complementary technology to NFV. Although it addresses a different area, it has a similar intent of exemplifying the network environment. It promotes the decoupling between the forwarding function and the control function, which is seen at a higher level of abstraction. Until now the two functions of forwarding and the one of control have collaborated in extreme synergy. Tendentially, the control function consisted of routing or switching protocols (depending on the network configuration, whether at level 2 or level 3) which defined how the packet had to be forwarded on the device's ports. Now the landscape is changing. The control function is assigned to a top layer with which the user can interact. The user can, therefore, create service chains that are independent of the routing/switching protocols implemented within that network. For example, you can force a package that has specific characteristics to follow a particular path rather than another.

Both NFV and SDN are technologies that if used together can undoubtedly bring significant benefits in the general network architecture, as well as extreme flexibility to the operator who finds himself configuring increasingly large and sophisticated networks, also due to the new minimum security requirements that must be met if you do not want to incur administrative penalties.
In the following sections, the proposed architectures are analyzed concerning the introduction of these two new technologies in the world network landscape. Even in this case, having a standard to refer to is extremely important. Ensuring interoperability between the various network environments is essential. On the other hand, we are avoiding a closed standardization of the architectures that have been proposed within the two technologies just examined. In addition to being a long process of standardization, it is also a process that makes the introduction of new elements extremely complicated. It is not possible to dampen the development of these new technologies now also given the potential and benefits that they promise to bring into production even in the most complex network environments. So it was decided to work on open standards. More and more companies interested in these technologies are starting to invest in their development and then use them in their environments. We are facing a very different panorama from the one that has dominated the field of computer networks so far.

2.2.3 ETSI NFV Management and Orchestration

ETSI (Industry Specification Group) NFV (Network Functions Virtualization) with a two-year duration and the objective of defining use case, requirement and a reference architecture was created from this initiative. [12]

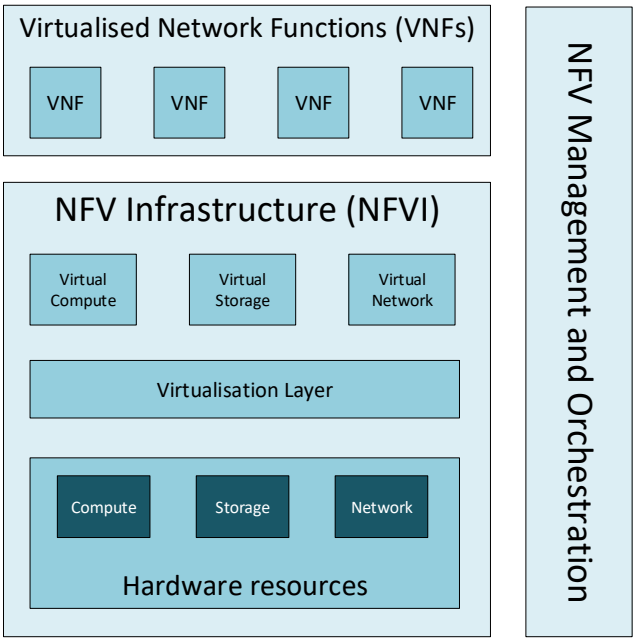

Figure 2.3. High-level NFV Framework

This architecture introduces the different domains into which an NFV network is divided:

- **NFV Infrastructure**, which includes the physical and virtual resources made available by the virtualization layer;
- **Virtual Network Functions**, which includes the set of virtual machines that perform the network function or the virtualized service;
- **Management and Orchestration (MANO)**, which includes the tools needed to manage other domains.

The virtualization infrastructure domain, called NFVI (NFV Infrastructure), is the environment in which VNFs are deployed, managed and executed. It consists of generic hardware components whose task is basically to provide a pool of physical computing, storage and connectivity resources. An intermediate virtualization layer then abstracts these resources. Through this layer, the VNFs are decoupled from the hardware on which they are executed. The functions exposed by the virtualization layer are therefore the abstractions of the underlying computing, storage and network. Finally, the MANO domain is the management and orchestration environment of infrastructure resources and virtualized functions with the ultimate goal of allowing the management of network services on the virtualized infrastructure.

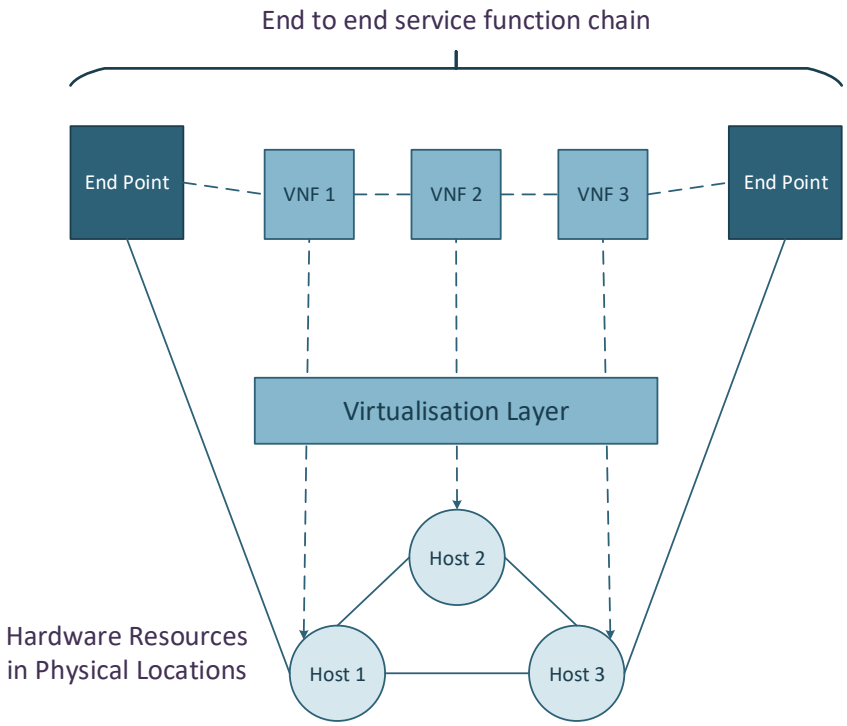

Figure 2.4. Example of and end to end network service with VNFs and nested forwarding graphs

The figure 2.4 clearly shows the decoupling of physical resources on the one hand and virtualized network functions on the other. The various virtual services that network packet must pass through for its correct processing are implemented on the physical resources available in that particular network environment. There does not necessarily have to be a 1 to 1 mapping, in the sense that multiple virtualized network functions can be implemented on the same physical resource.

2.2.4 Leveraging NFV and SDN in ICSs

This innovation in the field of networks has been exploited enormously in the industrial network infrastructure. Applications, on the one hand, become increasingly flexible and dynamic, but at the same time require certain guarantees to function correctly. The fourth industrial revolution, as seen in the previous section, has significantly changed the technological, social and business paradigms. It brought "smart objects" to the foreground, offering a point of contact between the virtual world and that of physical resources. In an industrial scenario, however, requirements concerning low latency reachability and security must be guaranteed first. Migrating to a virtualized world has made it possible to achieve much higher levels of flexibility. For example, if you are the victim of a cyber attack, I may want to reconfigure the network in such a way as to allow the correct performance of safety-critical functions in my infrastructure [13].

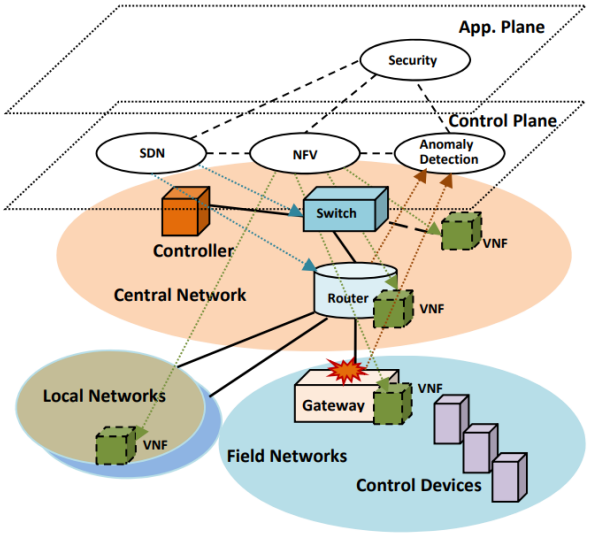

Figure 2.5. NFV/SDN enabled DDoS mitigation scheme for ICS against DDoS attacks scenario [13]

Operating this reconfiguration in a virtualized world is much simpler. In the industrial field it has many possible uses, also considering the critical nature of the systems that are often involved in networks of this kind.
Furthermore, suppose that I have implemented some application that at a particular moment requires to reach a node on the network while maintaining very low latency. I can reconfigure the network to give higher priority to this application and pay attention to the positioning of the virtualized network functions so that these they are positioned to give priority to the incriminated flow. There are many examples of integrations of these new technologies in the industrial field, and it is precisely in this field that the thesis work is placed.

Chapter 3

Real-Time Industrial Systems

In this chapter, the focus is on real-time industrial systems. As analyzed in the previous chapter, thanks to the rise of new network virtualization technologies (NFV and SDN), a very high degree of flexibility has been achieved. This revolution has led to the rise of applications with very high potential. But they require certain guarantees in order to function correctly and carry out their safety-critical mission in an area such as the industrial one.
This chapter starts with a quick overview of the main features of some real-time applications, increasingly widespread in the industrial field. Two solutions are therefore presented, developed to guarantee the correct functioning of these new industrial applications. There is also a high-level description of technologies that have been used concerning these two solutions. These technologies are then reused to make possible the thesis work implementation that is extensively analysed in the next chapters.

3.1 Overview

A real-time control system is a system in which the various elements that compose it must have the ability to communicate with very low latency, as close as possible to zero. Often they are closed-chain systems that serve to monitor a particular process. Depending on the conditions detected, the system must be able to make a decision as quickly as possible, without introducing a considerable latency that could make the decision no longer valid. Since applications of real-time industrial control systems are multiple, consequences that the introduction of a delay in the chain of operations may have is different depending on the criticality of the system. In general, real-time control systems are divided into two large categories: hard real-time systems and soft real-time systems. The first ones are control systems where real-time communication constraint must necessarily be met. A failure to meet this minimum requirement can lead to fatal situations. They are, therefore, systems that have an extremely critical nature. Soft real-time systems are systems that have much more flexibility. A failure of these systems generally does not lead to critical consequences. So constraints imposed by the fact of being real-time control systems can be relaxed in some situations. Today we have real-time control systems almost everywhere. Examples of soft real-time systems are telephone switching,

image processing. In these systems, the accuracy of result decreases after the real-time requirement is not met, but there are no catastrophic situation linked to it. Examples of hard real time systems are autopilot system in plane, nuclear plant control system, anti-lock brakes, and pacemakers. A failure in these systems obviously brought to unwanted situation because they are safety-critical system. Depending on the system treated, different precautions can be taken. When dealing with real-time control systems of the hard type, in the context of a network design operation, extreme attention must be put to the devices involved in low-latency flows. Failures may occur at any time, with a higher or less probability depending on the conditions of the infrastructure installation location and other boundary conditions.

3.2 Related Works

Over the years, different solutions have been proposed to face emerging problems in the industry 4.0 environment. In particular, two solutions have been taken as a reference. They exploit the advantages coming from the use of new technologies such as SDN and NFV in a critical area such as the industrial one. The main objective, common to both solutions, is to provide guarantees regarding the reciprocal reachability of devices that require very low latency times. The study of these two solutions is very interesting because it allows the analysis of some uncovered features on which the solution proposed in the next chapters is focused. The framework developed, whose name is VerINS, will integrate the approach followed by these two solutions, trying to solve open issue that come from approaches adopted.

3.2.1 Latency Aware VNF Placement

This first solution examines the problem of positioning virtualized network functions on the available physical infrastructure [14]. Particular reference is made to the IIoT environment, which has evolved enormously in recent years. The IIoT paradigm finds application in many different domains, such as Industry 4.0, smart grids, smart production, and smart logistics. The introduction of new technologies, such as 5G mobile networking, has allowed the creation of reliable point-to-point connections with low latency, thanks to the URLLC (Ultra Reliable Low Latency Communication) service provided by 5G itself. By studying the problem of Virtual Network Embedded (VNE) regarding VNF positioning, a formal verification is required regarding:

- the correct configuration of the VNFs involved within the network flows taken into consideration;
- the low latency requirements are met for the network flows considered.

By introducing network device configuration errors involved in a service chain, you can have more or less significant problems depending on the nature of the environment you are working on. Misconfiguration of the security parameters can

allow malicious access by third parties. Or it can block flows that should be allowed to guarantee the correct execution of all the features deployed in the network under consideration. Configuration errors can, therefore, compromise the mutual reachability of the nodes. So it is essential implementing within the framework an automatic function to verify the minimum requirements that the network must satisfy. Automating these verification operations is essential. Operator actions in a dynamic environment like the industrial one are entirely useless. Virtualized network technologies such as NFV and SDN also allow the network to be automatically reconfigured according to certain boundary conditions. For example, as seen in the previous chapter, to mitigate a DDoS-type attack, I can set the automatic network reconfiguration to divert malicious traffic and allow authorized traffic only. Here we need to design a framework that can automatically perform these formal verification operations.
The problem of VNF placement (alse known as Virtual Network Embedded problem) is formalized using the MAX-SAT approach. A maximum satisfiability problem (MAX-SAT) is a problem of determining the maximum number of clauses that can be made true by an assignment of truth values to the variables of the formula. In this first solution, the target is the search of an optimal positioning of VNFs minimizing the end-to-end latency between the various nodes involved in communications. This approach is the only existing one for IIoT systems that solves both problems of placement and constraints verification in single instance by merging these two concepts together. Problem is solved through z3 optimizer, a tool developed by Microsoft (in the next chapter it is analysed with all its peculiar features since it is also used in thesis work development). There are three input parameters:

- **VNFs model** that shows their forwarding behaviour and their configuration parameters. This corresponds to the virtualization layer definition of the ETSI NFV Framework.

- **Physical substrate model**: There is the definition of each physical resource in the studied infrastructure in term of computational power and storage available.

- **Constraints** that must be satisfied in term of security and end-to-end reachability. Since this is a MAX-SMT problem, there is the possibility of introducing hard and soft clause. Hard constraints are those that must necessarily be satisfied for the formulation of a valid solution to solve the problem. Soft ones, instead, can be falsified. To each soft constraints there is the possibility to attach a weight, in order to find a truth assignment to the propositional variables that maximizes the total weight of satisfied clauses. In this particular case, hard constraints concern the consumption of physical device resources available in the network. These constraints must necessarily be respected; otherwise, failures may occur since a device may not be able to support too many VNFs. Soft constraints, therefore, concern the placement of VNFs on a specific host. Each constraint is associated with a negative weight. It corresponds to the average latency associated with that VNF if it is deployed on that particular host. Subsequently, optimal VNFs placement is chosen to maximize the negative weight associated with all the VNFs crossed by the flow

taken under examination; therefore, global network behaviour is optimized, minimizing the latency.

The validation of the module thus developed was addressed on a Smart Grids network. The Smart Grids network topology is an exhaustive example of all the innovations introduced by the industrial 4.0 within the IIoT. Smart Grid means the combination of an information network and an electricity distribution network. We try to manage the electricity grid in an "intelligent" way under various aspects or functionalities through the use of data collected from billions of sensors. From the results of this first validation, for small topologies, the framework can provide a solution to the problem within acceptable times. However, for increasingly larger topologies, the time of the verification phase begins to grow exponentially due to the considerable number of constraints that must be validated. However, it is an excellent starting point for the development of the thesis project since it represents the first example of how verification and VNF placement optimization work together.

3.2.2 Time-Sensitive SDN

This solution instead provide guarantees regarding low latency communication between two or more devices in an industrial network [15]. These guarantees in today's industrial networks are achieved using some proprietary protocols, the Fieldbus protocols, which however require specialized hardware to be correctly used, as explained in the previous chapter concerning use of Profinet in industrial network. Fieldbus protocols dramatically limit the interoperability between different systems. Meeting end-to-end low latency requirements was also the main focus for two large organizations in the world of networking: IETF DetNets Working Group and the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group (TG). Solutions that they have been proposed over the years have not been easy to deploy in a production environment. There were some non-trivial implementation difficulties. Most of these solutions require the use of some particular strategies on a physical level. Sometimes an ad-hoc built infrastructure and devices capable of interacting through the use of these protocols are also required. They require the modification of the ISO / OSI protocol stack in the lower layers. It is often necessary to have a particular level two packet to be able to provide guarantees regarding the end-to-end low latency reachability. For instance, in TTEthernet (Time Triggered Ethernet), in addition to specific switch, ethernet frame have been modified compared to the tradition one.

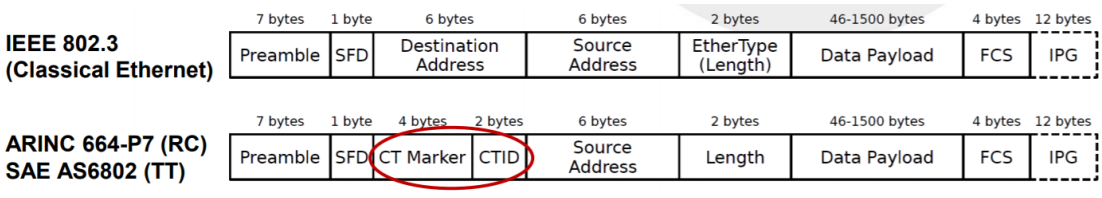

Figure 3.1. Time Triggered Ethernet: layer 2 frame

In place of the MAC destination field is a 4 byte CTMarker and a 2 byte CTID. The CT Marker is a static identifier used to distinguish time-triggered frames from other Ethernet traffic. The CTID is used by the switches to route time-triggered frames through the network [16].

However, these modifications clash with the flexibility required by new industrial applications. More and more times, it is necessary to introduce new devices within the network for various reasons. Due to the increase in the number of cyberattacks targeting safety-critical infrastructures like the industrial one, the introduction of some security middle-boxes is fundamental today. Firewalls, DPI, IDS / IPS are very used in the world of networking, whose use then becomes mandatory in the case of mixed infrastructure. In some corporate infrastructure, there is often a network part exposed to the public Internet and which requires communication with some devices of the industrial network. The segregation of the two networks is fundamental. Introducing devices that are built ad hoc to interact according to a non-standard ISO / OSI protocol stack is challenging in terms of implementation as well as extremely expensive. In the world of industry 4.0, it is no longer acceptable. We are trying to migrate to the most widespread protocol on the network today: IP (Internet Protocol). IP allows extreme flexibility and interoperability between the various systems in the network but, since it has a best-effort nature, it does not provide any guarantees regarding the processing of data on the network, nor is it able to privilege a network flow rather than another.

In this second proposed solution, we try to exploit the potential of new technology, the SDN, to guarantee a deterministic delay in the communication between the devices on the network. The concept of time-triggered networks is resumed where, using appropriate protocols, transmission scheduling of the packet is possible, making sure that the end-to-end delay is constant. We are now trying to schedule packet transmission in the IP world. It is possible thanks to the use of the Precision Time Protocol (PTP), whose details and uses are analysed in the next section. In general, it allows a precise synchronization of the clocks of the devices in the network. In particular, in the context of this solution, it is used to schedule packet transmission over the network keeping all synchronized. The main idea of the SDN technology is to have a device that can provide a global view of the network. Through this device, you can manage the control plan of the switches from above to influence their forwarding behaviour. This solution proposes algorithms that assign timeslots to the time-triggered flows and route them such that in-network queuing is avoided (constraint) while maximizing the number of such flows in the network (optimization objective). In this way, it is possible to calculate the end-to-end latency in a deterministic manner. This is the benchmark topology taken as a reference:

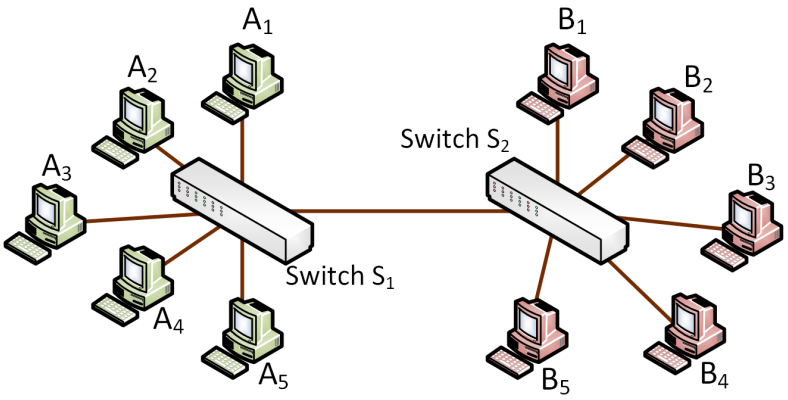

Figure 3.2. TSSDN Benchmark topology [15]

There are five time sensitive flows ($Fi : Ai \rightarrow Bi; i \in [1...5]$) that have to be schedule through the entire network environment. According to the logic of the proposed solution, each flow has a reserved timeslot to pass through the network. In this way, overlapping between different flows in the network and the creation of queues through shared network devices are avoided. Below is an example of a network architecture that contains all the key elements of this proposed solution:

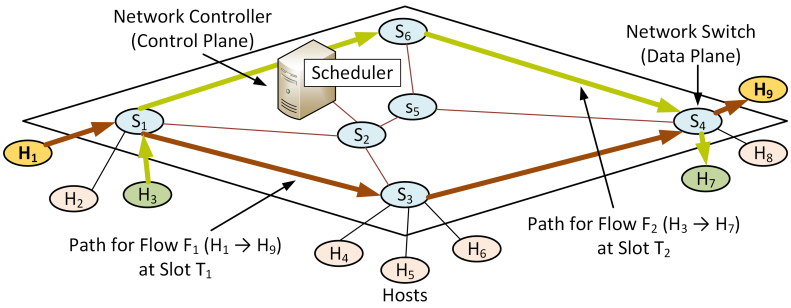

Figure 3.3. TSSDN Simple Test Case[15]

The scheduler manages the network control plan. It has a high-level view of the network and plays the role of a standard SDN architecture controller. In the TSSDN architecture, it has the task of coordinating the actions of the devices involved in the network forwarding operations handling various flows avoiding queues and overlaps. In the example above, two are the flows declared as Time-Sensitive: F1, from H1 to H9, and F2, from H3 to H7. According to the TSSDN, the two flows must be managed in two disjoint timeslots (T1 and T2). This means that all the devices on the network are synchronized and enabled to perform forwarding actions for both flows in the defined transmission timeslots.

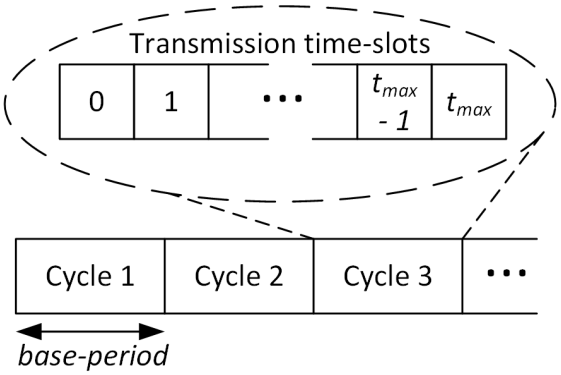

Figure 3.4. TSSDN Timeslots Partitioning [15]

3.3 Interesting Exploited Technologies

In this section, we examine in detail some technologies used in the context of the solutions just presented. Both the PTP (Precision Time Protocol) and the proprietary Intel DPDK (Data Plane Development Kit) technology are essentials in the real implementation of the framework illustrated in the following chapters.

3.3.1 Precision Time Protocol

The Precision Time Protocol is documented within the IEEE 1588 standard. During the development of computer networks, it played different roles. The initial purpose of the PTP was to provide as accurate as possible synchronization between various devices clocks on the network. The clock is dictated by a device that, within the network environment , is elected to master role. The other devices, slaves, synchronize their clock to that of the master, achieving precision in the order of microseconds.

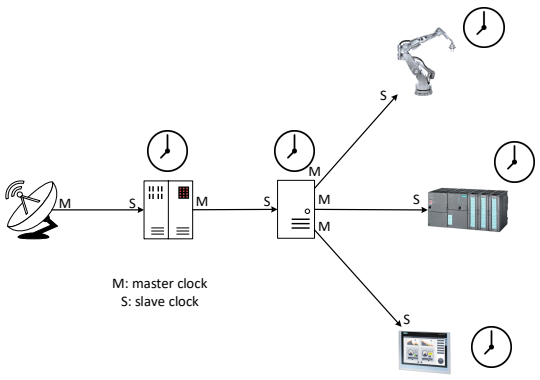

Figure 3.5. PTP original purpose

Initially this protocol was used in the industrial automation field to synchronize actions of various devices in the network. It, therefore, had a role very similar to

the one currently performed by the NTP (Network Time Protocol) in corporate networks. Since the PTP allow to achieve much higher accuracies, it was chosen to be used within critical environments such as the industrial one. In fact, as in automated production chains, some operations must necessarily be carried out before others. Suppose we have a conveyor belt like the following:

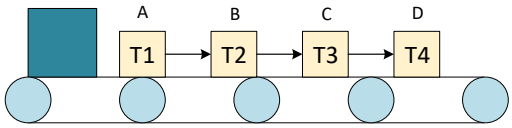

Figure 3.6. Conveyor Belt Example

In the example of the conveyor belt shown in the figure 3.6, four are the actions that must be coordinated [17]. In particular:

- A must be performed at time T1;
- B must be performed at time T2;
- C must be performed at time T3;
- D must be performed at time T4.

To guarantee that the actions are performed at the fixed time, various devices involved in the execution of the actions could be configured to be synchronized each other through the PTP protocol. Therefore, the execution of a given action can be scheduled at a particular time.

For years, PTP has also been exploited as part of the Filedbus protocols. As anticipated in the previous chapter, Profinet, one of the most widespread protocols in the industrial field, has used PTP in its internal implementation for the safety-critical real-time flows management. Profinet is a protocol that is built on Ethernet. By definition, Ethernet provides a shared medium where everyone can transmit in any instant. This makes Ethernet a probabilistic shared medium access. Mechanisms have been introduced to allow deterministic access to this shared medium. The CSMA-CD (Carrier Sense Multiple Access - Collision Detection) allows multiple access nodes to the shared medium. They transmit at any time. Network nodes themselves are responsible for the transmission they are conducting and, if they detect collisions with some other node in the network, retransmit the data again. In today's Ethernet networks this phenomenon has been overcome thanks to the introduction of network switches, which separate the collision domain. PROFINET Conformance Class A and B devices leverage standard Ethernet infrastructure to achieve cycle times as short as 1 millisecond and jitter of about 10-100 microseconds. This is the standard "Real-Time" (RT) PROFINET communications channel [18]. However, some applications require deterministic network behavior and should never be vulnerable to collisions or jitter. In fact, Profinet has

designed an Ethernet extension at the MAC level. It allows each network-enabled switch to provide time slots in which the flow associated with these safety-critical applications can be managed in a deterministic manner. In those defined time slots, access to the network is therefore of the TDMA (Time Division Multiple Access) type. PROFINET extends PTP in a wrapper protocol called the Precision Transparent Clock Protocol (PTCP) used to measure end-to-end latency considering devices and physical link involved in the network infrastructure, in addition to the capability of sharing a common clock between enabled devices.

PTP can also be used in the virtualization world [19]. Virtualized network functions are performed within a server just as if they were virtual machines. On each of these a PTP daemon is run and therefore it is possible to schedule with extreme precision the action of a virtualized network function on a given flow. Many works show how PTP is actually used in URLLC environment dealing with synchronization problem [20].

3.3.2 Intel Data Plane Development Kit

Intel DPDK technology has been defined as the key enabling technology for the virtualization world in the networking field. It is playing a fundamental role in the world virtualized network services orchestration. It provides a set of libraries that allow the programmer to get a standard interface to the underlying specific hardware by creating an EAL (Environment Abstraction Layer). We are leaving the idea of a network device designed on a hardware level to cover a specific function. Network functions are now virtualized and can be performed by any general-purpose processor. Most Unix-like systems currently allow the processing of network packets at the application level, exploiting a capture system such as the one shown below:

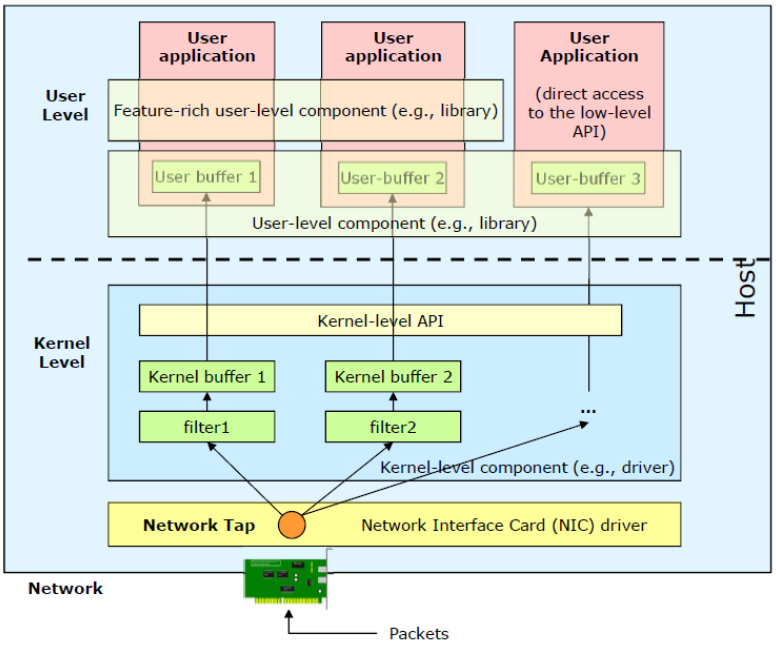

Figure 3.7. Typical Packet Capture Architecture

Today it is widely used for the implementation of numerous network functions that require the capture of packets in transit (Firewall, IDP / IPS, DPI ...). At the base of this architecture is the concept of virtual CPU (vCPU). It is nothing more than a straightforward piece of code that runs at the kernel level. Generally, they are filtering operations that are performed just at the moment of packet capture. The packet, therefore, passes through the kernel and is then delivered at user level, where more complex applications can process it depending on the function that the device is called to perform. Various optimizations have been introduced over the years to this basic architecture. Going all over the stack for some applications can be too expensive in terms of the processing time invested. Intel DPDK allows to bypass the kernel completely, and therefore user-level applications can interface directly with the network card driver. You can achieve very high performance while maintaining another level of flexibility at the same time (since writing code at the user level is much simpler than injecting code at the kernel level).

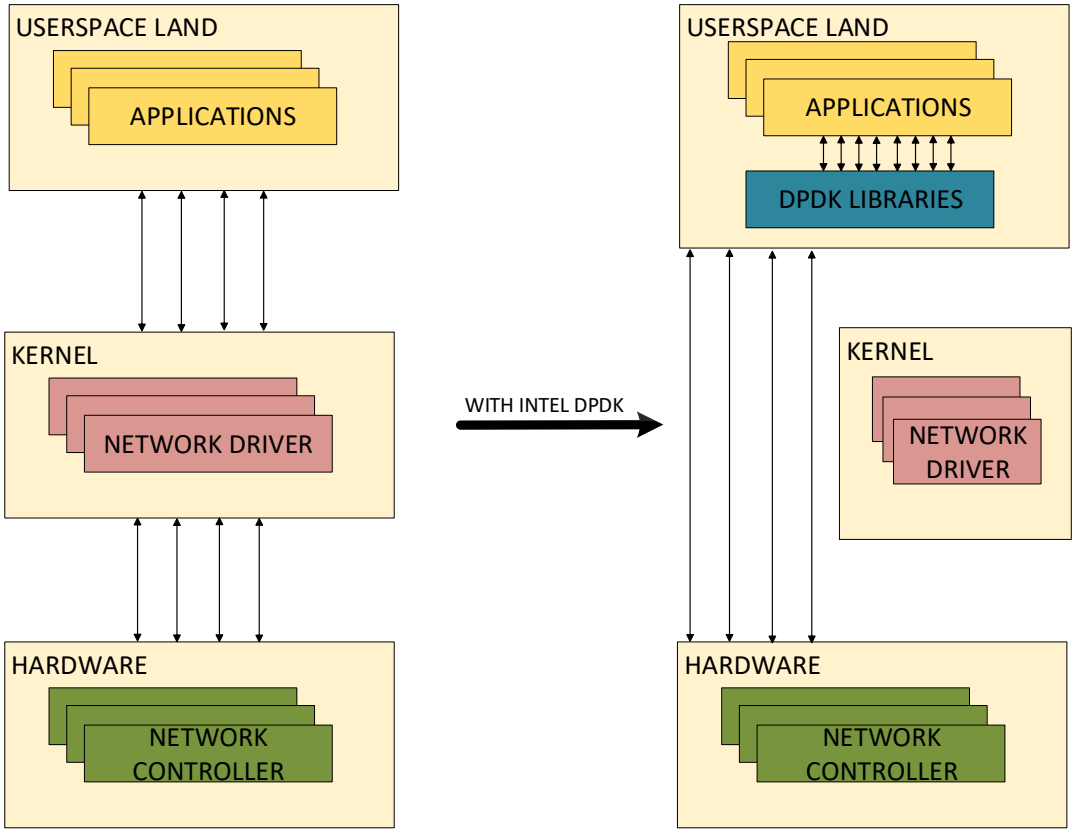

Figure 3.8. Linux Kernel With DPDK

Since Intel DPDK 2.2 release, support for PTP has been introduced. User-level applications can now take advantage of a PTP daemon that provides a synchronization layer between all the applications that implement it on the network. Thanks to strict cooperation between DPDK and PTP, it was possible the design of a network environment ready to host solution presented in the following chapters.

Chapter 4

Thesis Objective

Within this chapter, weaknesses of the previously examined solutions are analysed. We want to provide a contextualization and justification of the thesis work done. A section is dedicated to the exposition of the difficulties to address in the hard real-time flows management in industrial network environments. So a short section is dedicated to the presentation of what is expected of the framework being studied. Finally, a high-level presentation of the developed module is provided to introduce it to lectors.

4.1 Open Issues

The solutions presented in the previous chapter tried to propose a formal methodology for hard real-time flows management in a safety-critical network such as the industrial one. The common goal of both solutions was to provide guarantees regarding end-to-end latency. In fact, the limits of both have been analysed, and a solution has been sought that can address them. This is the backdrop for the entire thesis work.

The framework proposed by the first solution, VerifOO (Verification and Optimization Orchestrator), addresses the virtualized network functions placement problem, providing an optimal solution that minimize end-to-end latency between the various hosts on the network. However, it does not provide an upper bound on the end-to-end latency. It proposes only an intelligent positioning of the VNFs within a physical substrate, taking into account the delays that are introduced when flows pass through a VNF. In particular, the delay can be divided as follows:

- Time to reach the host on which the virtualized network function is deployed. The transmission speed of physical links strongly conditions this first step;
- Queuing time on the various hosts crossed. Generally, more flows pass through a host on the network, and the creation of queues is very common, introducing a delay. The packet must wait for all network packets already in the queue to be processed by the disputed host and then processed as well;

- Network packet processing time on the host on which the virtualized network function is deployed;
- Time to reach the destination host. Again the transmission speed of physical links strongly conditions this step.

There is, therefore, no upper bound guarantee on the end-to-end delay between two or more hosts. There is great variability depending on the network conditions instant by instant. The devices and physical links involved in the various forwarding operations can be more or less discharged depending on the network flows. This is not taken into account by the framework which therefore operates with a best-effort logic. This approach is valid for the management of less critical industrial flows, supposing to design the network by oversizing it. In this way, you can avoid to have a network that works at the limit of the conditions for which it was designed. However, some critical applications need real guarantees regarding the end-to-end of the hosts. These guarantees can be achieved by changing the underlying logic. In addition to an optimal virtualized network functions placement on the available physical substrate, also scheduling of the network flows can be proposed. It can avoid the formation of queues and the overlapping of two or multiple flows that can lead to overloading the devices and links involved. However, the study of this first solution was very interesting. In literature, it is one of the few solutions that face together the problem of VNF placement and formal verification of the configuration of the devices crossed by the various network flows, to avoid problems during automatic network reconfigurations.

The TSSDN (Time Sensitive SDN) solution has proposed a real optimal scheduling of network flows, taking up the scheme proposed by the various examples of TTEthernet (Time-Triggered Ethernet). In this way, real guarantees are provided regarding the end-to-end latency between two or more devices on the network. A deterministic end-to-end delay is achieved with a very low and bounded jitter. However, the detail with which the network flow scheduling problem is analyzed is at the end-to-end flow granularity. Tendentially, for every network flow, there is a timeslot allocated for the management of that single flow. If two flows intersect, even if only for a while, these must be managed on two separate timeslots. Considering the topology below:

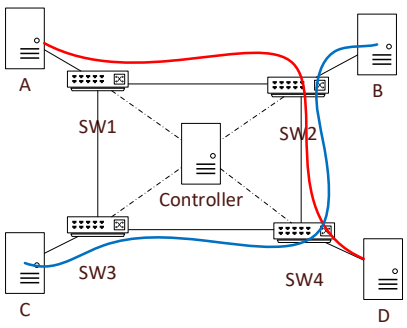

Figure 4.1. Simple network topology with two time-sensitive flows

Suppose that the two flows, the red one and the blue one, are time-sensitive flows and therefore related to applications that need to obtain a guarantee regarding end-to-end latency. According to the formulation proposed in the TSSDN context, since the two flows overlap themselves, it is necessary to allocate them on two different timeslots. Furthermore, the timeslot must last long enough to guarantee the correct end-to-end communication. Sizing the timeslot in a more complex topology could be very complicated and limiting in terms of reachable performances. In the topology of the figure 4.1, assuming that the maximum length of the network can be managed in a time equal to 4 milliseconds, the duration of the timeslot must be at least 4 milliseconds. Since the two flows intersect, two timeslots are needed to manage them and therefore, a time equal to 8 milliseconds is needed. In our solution, the end-to-end flow is broken down into its minimum units (in the case of the example in the figure 4.1 the minimum unit corresponds to the crossing of a physical link in the given topology). The transit of the packet along each minimum unit is scheduled. In this way, it is possible to obtain better scheduling since not overlapping parts of the two flows can be scheduled to be managed in parallel. Returning to the figure 4.1, time-sensitive flows can be scheduled as follows:

Table 4.1: Scheduling Example

Timeslot ID	**Flow Element (AD)**	**Flow Element (CB)**
T1	from A to SW1	from C to SW3
T2	from SW1 to SW2	from SW3 to SW4
T3	from SW2 to SW3	-
T4	-	from SW3 to SW2
T5	from SW3 to D	from SW2 to B

In addition, VNFs placement problem is not managed by the TSSDN solution. The VNFs management involved handling the mapping of virtual topology in the physical one, which does not always coincide. Addressing the problem of VNF placement becomes of fundamental importance, especially when dealing with an extended network. In small industrial networks, the network operator can manually determine which host has to deploy on its a particular VNF. In these cases simple network flows scheduling can be sufficient to handle end-to-end communication guaranteeing a maximum delay, using an approach similar to the one introduced with the TSSDN. The size of industrial networks is gradually becoming important due to a large number of devices involved. And the architecture of a typical industrial network is also significantly evolving to include middlebox devices. These devices can have different functions. The most important is undoubtedly the one related to security. In particular, middleboxes often have functions of:

- logging and monitoring. For example, they can track intercepted traffic;
- inspection with consequent blocking of malicious traffic or that does not meet certain rules (IDS / IPS function and firewalling).

One of the most famous industrial network architectures is the smart grids one. The smart grids network architecture involves a number of devices destined to

grow more and more in the future. Controlling their functioning becomes more and more complicated for a network operator, as the resolution of the problem of VNF positioning is no longer obvious. We need a framework that can handle everything as transparently as possible to the end-user. The VNFs placement management was, therefore included within the solution proposed in this thesis work as a fundamental problem to be solved in parallel with the scheduling of network flows. Details of the implemented solution are discussed in the chapter 5.

4.2 Expected Results

The idea of VerINS is developed, starting from the analysis of the weaknesses of one and the other solution. We want to develop a framework capable to:

- provide an optimal VNFs placement minimizing the end-to-end delay between various hosts involved in the various network flows;
- elaborate a formal verification of the configuration of the devices involved in the various network flows;
- schedule the network flows to obtain guarantees regarding the end-to-end delay and avoid queuing in the various hosts involved in the network forwarding actions and overlap between the various flows

The VerINS framework could, therefore, be used in harmony with VNFs management and orchestration software, playing a fundamental role during network reconfiguration operations in safety-critical environments.

Finally, chapter 7 is dedicated to the validation and analysis of module performance. Various test cases have been implemented which have allowed us to validate the actual achievement of the objectives set and the performance achieved in terms of required processing time.

Chapter 5

VerINS Design

Within this chapter, a first section is dedicated to a high-level overview of the framework developed, which is then further investigated in the next chapter in terms of implementation details. Subsequently, we move to the description of the main features of the tools used in the framework implementation phase. These tools have allowed us to better focus on the problem formalization. They have been used with different purpose, but with the same aim to simplify module development since already implemented materials have been used. So the standard of interaction with what has been developed is also presented. Finally, the chapter ends with a logic description of network flows modelling. Also, the mapping between service graphs and the available physical infrastructure is presented, since it is considered undoubtedly one of the most salient aspects of the VerINS framework.

5.1 High-Level Framework Overview

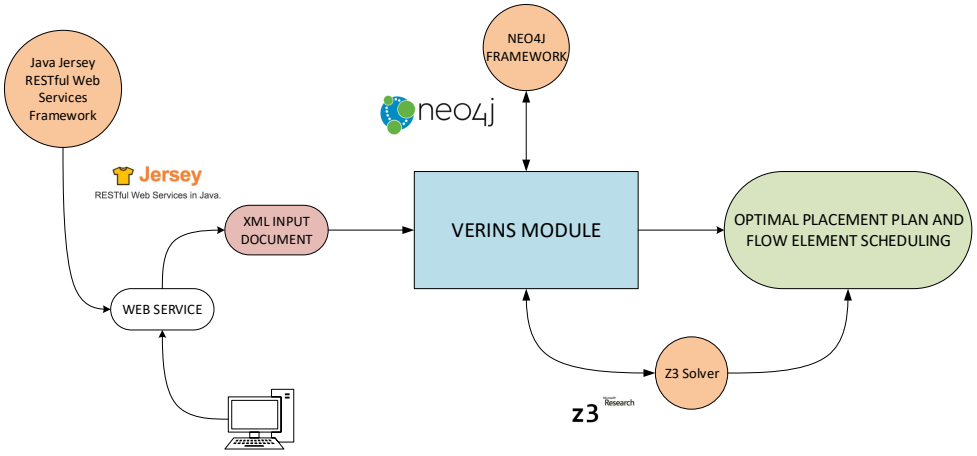

Figure 5.1. Framework Design Overview

The figure above highlights the main elements that were involved during the implementation of the VerINS framework. The module was developed in Java. A

document in XML format (eXtensible Markup Language) is required as input. The produced output contains information regarding the execution of the developed framework. In particular, it tells us if a solution has been found or not by the VerINS module. In the positive case, it also provides us with an indication of the optimal positioning of the VNFs on the available physical substrate and optimal scheduling of the time-sensitive flows in the network. The goal is to find an optimal solution for VNFs placement on the physical substrate that uses the least possible number of timeslots to manage all the time-sensitive flows declared by the user during the module interaction phase.

The VerINS module interact a series of already available and tested frameworks. Among these ones:

1. **Microsoft Z3 Solver**, a theorem prover provided by Microsoft Research;
2. **Neo4J Graph Database**, a database to store graph information;
3. **Java Jersey RESTful Web Services Framework**, a framework which provide libraries to develop easily a Web Service.

Interaction between VerINS module and these tools is handled through a main Java class that acts like a proxy, dispatching operation through other Java class developed.

In the next section, tools are presented in detail to the lector. In particular, their role inside the developed module is investigated to justify the need to include them inside the developed module.

5.2 Tools Used

5.2.1 Microsoft Z3 Solver

Z3 is a tool that allows you to verify the satisfiability of a series of logical formulas regarding a given theory [21]. Z3 is a low-level tool. The optimization and search algorithms developed by Microsoft are accessible through a high-level interface, convenient to use by a programmer. Different programming languages are supported by the tool. The most common and updated ones supported are C ++, Python and Java. Classes that support certain types of operations are displayed, depending on the needs of the programmer. Z3, therefore, allows us to verify if a series of logical and mathematical conditions are respected.

For example, suppose we want to verify De Morgan's theorem. It means that, given two logical variables A and B, we must verify the below conjecture:

$$\neg(A \wedge B) \equiv \neg(A) \vee \neg(B) \tag{5.1}$$

Accordingly to De Morgan conjecture, expression above is verified for each A and B boolean value.

Table 5.1: De Morgan Truth Table

A	B	$\neg(A)$	$\neg(B)$	$A \wedge B$	$\neg(A \wedge B)$	$\neg(A) \vee \neg(B)$
false	false	true	true	false	true	true
false	true	true	false	false	true	true
true	false	false	true	false	true	true
true	true	false	false	true	false	false

As shown in the truth table, equality always remains valid for any value of the Boolean variables A and B. This can be verified by z3. We need the declaration of the two Boolean constants A and B and of the boolean function that must be checked (the logical formulation of De Morgan's law). In the following examples, we will use a standard language, the SMT-LIB Language [22]. This language is strictly referred to the world of SMT solvers. The goal was to have a common language with which the various SMT solver available in the academic field could be tested, validating their functioning. From our point of view, SMT-LIB language is particularly explanatory. It can, therefore, be used to provide a general overview of the main features of the z3 module and its potential. Therefore it is necessary to declare the constraint that states that De Morgan's law must be verified.

```
(declare-const a Bool)
(declare-const b Bool)
(define-fun demorgan () Bool
  (= (and a b) (not (or (not a) (not b)))))
(assert demorgan)
```

Listing 5.1. z3 Problem Formulation Example

In particular, Z3 searches for possible Boolean values A and B that satisfy the declared formulation. If it is able to find two logical values for the declared Boolean variables that satisfy the constraint declared on the De Morgan function, z3 gives SAT (satisfiable) result. The problem, in this case, can be solved, and z3 gives us a possible Boolean value of the variables A and B that satisfy what is declared. If the declared function is satisfied for any value of A and B, z3 does not give indications about possible values A and B which satisfy what declared but gives only SAT as a result. Z3 can also be used to solve mathematical problems.

Another crucial aspect of the z3 optimizer is that it allows the addition of two different types of constraints. As previously analyzed, concerning the first solution proposed to address the problem of VNF placement in time-sensitive network flows management, we can declare hard and soft constraints, which are generally associated with a weight. Soft constraints must not necessarily be verified. The problem then turns into a MAX-SMT problem, whose main purpose is to satisfy as many logical propositions as possible. For our thesis work, the problem of VNFs placement and time-sensitive flows scheduling has been formalized just like a MAX-SMT type problem. We try to find the best solution in terms of VNFs positioning and allocation of available timeslots to the various network streams to minimize end-to-end latency between the multiple devices involved in time-sensitive communications. So the goal is to solve a problem of research and optimization. Objective functions and constraints that are injected into the environment managed by z3 are analyzed in details in the respective sections of the next chapter.

5.2.2 Neo4J Graph Database

Neo4j is an open source graph database software [23]. It is developed entirely in Java. This type of software is currently used when the relationships that exist between the data are as meaningful as the data itself. The contexts in which the graph databases can be useful are different. They are not only used in the context of social networks, where their structure is perfectly suited to the representation of the data. For example, Facebook, Google, Twitter and similar systems certainly use graph databases, even to keep trivially track of the relationships between users. Graph databases are also used for instance in:

- **Business Process**: they allow to perform real time operation in real time, making decisions faster and in more reliable way, handling more accurate data. For instance, in the real time recommendations system graph databases play a very important role. To have a high-quality recommendation you have put together data coming from different source. Graph databases are designed to do this kind of operations.
- **Fraud Detection**: the increasing number of online threats has led to the formulation of new techniques to prevent them. Since graph databases enhance the relationship between the data in the same way as the data itself, detecting a fraud with a database so structured becomes extremely easy. Analyzing millions and millions of transactions and finding relationships between them is now much easier. And it is precisely in this sense that these new graph databases have being used.
- **Master Data Management**: it allows you to have a centralized view of the entire company data but also external data. Since these data are full of reference, graph databases are suited to handle them. The aim is to better understand relationships between these kind of data. They can include employees data, customers data, product, orders.

As concerning our thesis work, Neo4j was used to handle the network environment in which the VerINS framework works. In particular, Neo4j allowed a simple schematization of the mapping between the physical substrate and the virtual service graphs. Interaction with Neo4j can occur in two ways:

- **Cypher Query Language**: it is a graph query language that has been built to interact with the graph database in a very efficient way. It is a SQL inspired language and allows to do CRUD operations on graph databases. It is suited to handle with nodes and relationship with straightforward instructions.
- **Neo4j RestAPI**: it makes available a simple Web Service with which you can interact to operate with a graph database hosted in a machine. This is the solution chosen concerning development of VerINS graph database. Many functions have been developed to interact with the database through Rest calls. They are investigated in details in the appendix of this work. Mainly it is introduced in the VerINS framework to handle paths in physical and virtual topology exploiting algorithm already implemented in Neo4j framework, such as minimum path research.

5.2.3 Java Jersey RESTful Web Services Framework

Jersey is a REST framework. It provides a JAX-RS, Java API for RESTful Web Services implementation [24]. Within the VerINS project, this tool is used for the implementation of a web service through which interaction with the developed tool was possible. A rest web service has been developed because of its great flexibility and interoperability. It allows resource organization. They can be managed through unique identifiers (URI, unique resource identifier) and many operations could be make available on each resource, since methods available inside HTTP protocol are used to distinguish operations. For instance, often HTTP GET operation is used to retrieve information on the resource requested or HTTP POST and HTTP PUT operations are instead used to update a resource on the VerINS organization. Through the web service developed, it is possible the interaction through the VerINS module.

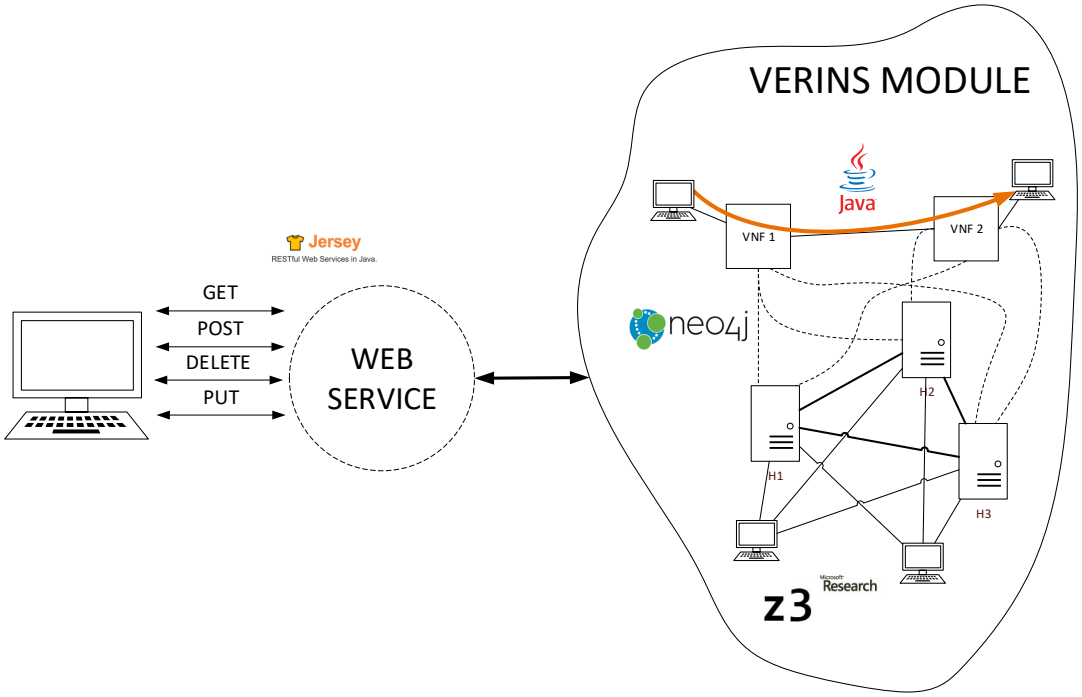

Figure 5.2. Interaction through Java RESTful API

The details of the interaction are analyzed in the appendix available at the end of the paper. An example that shows how it is possible to load a network topology on the VerINS environment and obtain placement and scheduling results via HTTP requests is also available.

5.3 Service Graph Management

The core of the VerINS module turns around the Service Graph concept. It represents a virtual network topology that is defined on a physical substrate that consists of general-purpose servers capable of hosting VNFs declared on the Service Graph. We are increasingly leaving the idea of network devices seen as devices built to perform a specific function. Thanks to the virtualization layer provided by the ETSI framework for the NFV environment, it is possible to define a series of functions that can then be deployed on one device rather than another in a completely transparent way. Various network reconfigurations, also depending on the general environment state, are managed by an orchestrator software.

The entities involved in the virtual and physical graphs are different. In general, within the virtual graph is the configuration of the network devices that act as middlebox. Middleboxes are devices that perform an active function when forwarding packets within the network infrastructure. Examples of middlebox are firewalls, proxies, DPIs, NATs. The task of the operator is to specify the configuration of these devices and then define the role they must play within the infrastructure in question. For example, an operator may want to define the firewall configuration, defining access lists that allow the transit of only one type of traffic. Or NAT configuration in the network peripheral areas may also be handled for publications and traffic management to the external network.

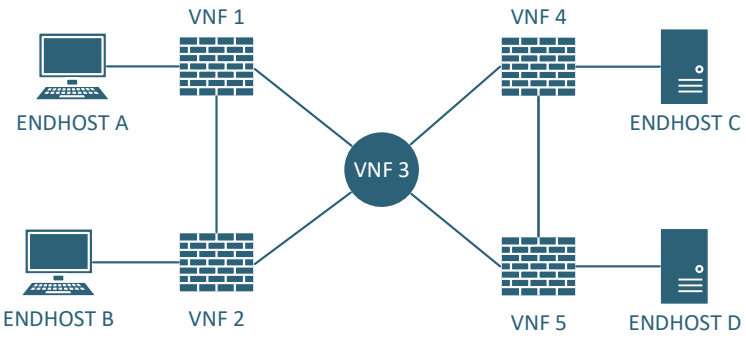

Figure 5.3. Service Graph Example

The network operator defines the role that VNFs play within the virtual topology represented in the figure 5.3. For example, concerning the figure 5.3, ENDHOST A and ENDHOST B can be two client devices in an industrial network that need to exchange information with ENDHOST C and ENDHOST D, which instead act as servers. VNF1, VNF2, VNF4 and VNF5 can play the role of a firewall, filtering out not allowed traffic while VNF3 can be a DPI, analyzing the traffic to prevent possible attacks

Instead, the entities that are involved in the physical infrastructure are very simple general-purpose servers. The user defines the characteristics in terms of available resources and therefore quantity of RAM, CPU power, storage and connections with other hosts. It should be noted that both the processing operations

of the packet on each host and each physical connection between the various hosts influence the forwarding of the network packet, introducing a latency that is not always acceptable.

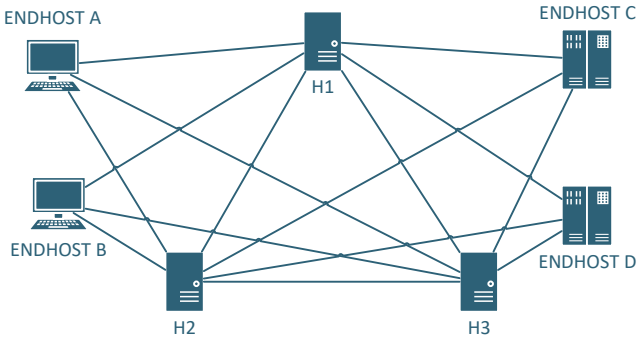

Figure 5.4. Physical Topology Example

In the figure 5.4 there is an example of a physical topology that can be exploited as a substrate for the virtual topology represented in the figure 5.3. When defining physical topology, it is important specifying the physical resources available for each physical host. These parameters are then taken into account during the placement operations of the VNFs. Physical resources must not saturate to ensure the correct functioning of the network and therefore, the correct management of the flows. There are several possibilities for deploying VNFs on the physical substrate. All the various possibilities are taken into account to then be chosen the best VNF positioning that can ensure the correct management of time-sensitive flows declared by the user in the shortest possible time.

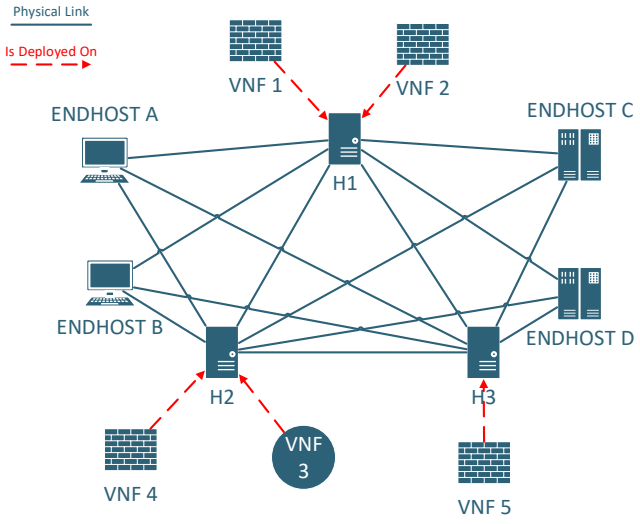

Figure 5.5. Deployment Example

In figure 5.5 is shown a possible deployment example of VNFs declared in virtual topology in figure 5.3 in the physical substrate 5.4, accordingly to physical resource

required by VNFs to operate and physical resource available on the substrate graph. We are assuming that ENDHOSTS entities are allocated on fixed endpoint. This is why there is an exact mapping between virtual and physical graph.

5.4 Flow Design

Network flows are declared using the virtual topology as a reference. The network operator, for example, may want to put two machines in communication but the flow must be processed through particular network functions. It could be wanted to keep track of the flow or verify that legal operations are performed during data exchange between the devices involved in that flow. In an environment like the industrial one, it is essential because of the criticality of the devices involved. Having a virtual abstraction of what happens inside the network can be of great help for the network operator itself.

Concerning the example illustrated in figure 5.3, suppose we declare two time-sensitive flows, so with the need to be handled providing them guarantees about end-to-end latency communication.

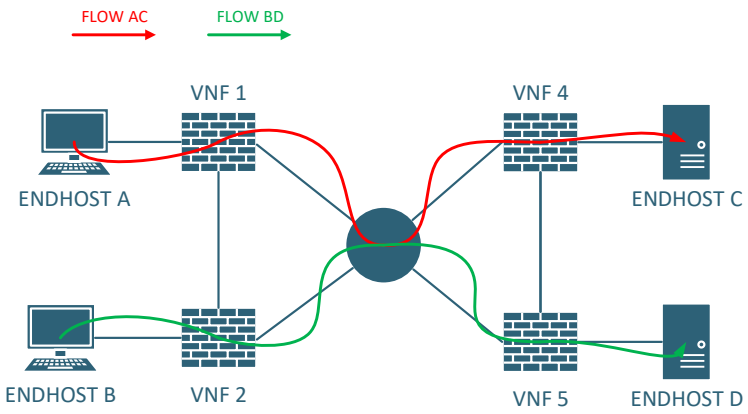

Figure 5.6. Virtual Flows Definition Example

One of the most challenging actions that have been addressed during the development of the VerINS framework is the mapping of the flows that are declared in the virtual network topology in the corresponding flows in physical topology. At this moment, to schematically show the design of the framework developed, we are taking for granted the fact of having a deployment example of virtualized network functions on the physical substrate. In reality, this is part of the result provided by the VerINS module since its goal is to find an optimal positioning of the virtualized network functions that can satisfy the user-defined requirements on the virtual topology, scheduling transmission over time.

According to the deployment scheme shown in figure 5.5, the flows can be mapped as follows:

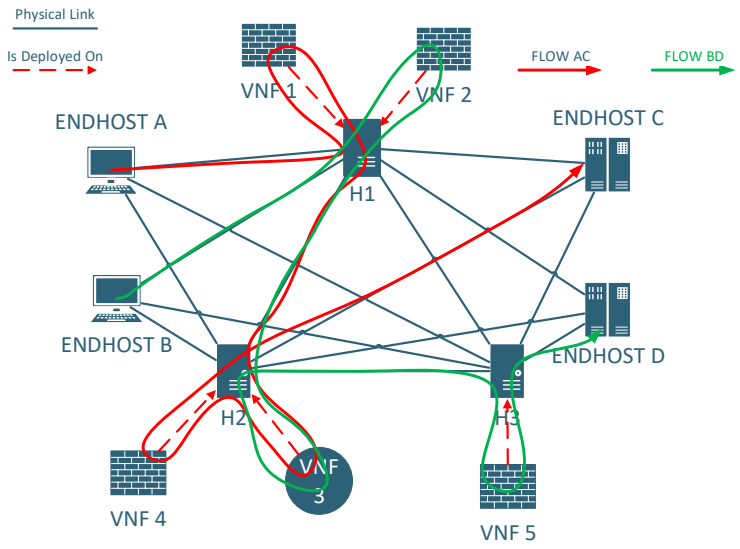

Figure 5.7. Physical Flow Mapping Example

As can be seen in the example just illustrated, many overlaps between the flows must be managed to avoid the creation of queues in the network.

The network flow elements that must are handled by VerINS module are:

- **VNF execution**: the execution of a VNF on a given network flow takes some time. Assuming that queuing and overloads cannot occur since the objective is to schedule packet transmission to avoid these phenomena, it is possible to calculate the delay introduced by a VNF for a given flow in a deterministic manner. Instant by instant, all the resources are allocated to the VNF that must be executed. Using the DPDK technology, it is possible to bypass the kernel, as analyzed previously, and therefore work closely with the hardware. It is, therefore, possible to perform tests that provide us in a deterministic way the time needed to perform the processing of an MTU sized packet through the VNF, also considering the configuration of the device. Some firewall implementations, for example, have a processing time that depends on the number of access lists declared in their configuration. It is, therefore, necessary to have this information to calculate the introduced delay in a deterministic manner.

- **Physical Link Crossing**: crossing a physical link introduces a delay proportional to the characteristics of the medium crossed. It is advisable to use reliable transmission media in critical communications, possibly redundant to tolerate any faults.

In general, it is important to have an idea about the delay introduced by every single element so far analyzed. Scheduling operations require the knowledge of these details to be able to schedule operations correctly within the network.

5.5 Timeslots Allocation

Resuming the solution previously illustrated concerning the TSSDN, at this point, it is necessary to define some configuration parameters to proceed with the scheduling of transmissions in the network. In particular:

- **Single timeslot length**: this can be intended as the shortest time scheduling unit. It must be long enough to handle the flow element that takes the most time to run inside the network. In this way VerINS module can schedule every VNF or physical link crossing in one timeslot.
- **Base period length**: this is determined by the number of timeslots needed to manage the flows within the network.

As previously shown in figure 3.4 transmission timeslots are allocated inside a base period that is repeated over time. Since these two parameters have been defined, it is possible to start estimating the maximum and average latency for each time-sensitive network flow verifying that all constraints defined by a network operator are satisfied. Concerning example shown in the figure 5.7, supposing that timeslots have been size to host the VNF or physical link that take the most time in the declared network environment, a possible schedule that minimises the number of used timeslots maximising the number of time-sensitive flows handled could be the following one:

Table 5.2: Timeslot Allocation Example

Timeslot	**FLOW AC**	**FLOW BD**
T1	AH1	BH1
T2	VNF1	-
T3	H1H2	VNF2
T4	VNF3	H1H2
T5	-	VNF3
T6	VNF4	H2H3
T7	H2C	VNF5
T8	-	H3D

The table shows how the AC flow is started to be managed at the T1 timeslot of each base period and ends at the T7 timeslot, while the BD flow management, even when it starts at the same time as the AC flow, ends at T8 timeslot. Note that if an AC flow packet is ready to be sent to the T2 timeslot, it will have to wait for the T1 timeslot of the next base period to be managed by the network, to then be delivered to the T7 timeslot. The same applies to the BD flow, but in return, there is the guarantee of having a deterministic network behaviour. This could be generalized in the following manner. Called:

- ts_{len}: timeslot length defined by network operator accordingly to the VNF properties declared in the network environment and to the physical link type

used in the physical topology;

- n_{ts}: number of timeslot proposed by VerINS module to optimize management of flow declared;
- ts_{start_i}: timeslot where starts handling of i-th flow declared in network environment;
- ts_{end_i}: timeslot where ends handling of i-th flow declared in network environment.

maximum end-to-end delay for i-th flow could be computed as following:

$$d_{ee_i} = ts_{len} \times (n_{ts} + ts_{end_i} - ts_{start_i}) \tag{5.2}$$

This formula is then pushed in our z3 problem formulation to find optimum VNF placement solution and flow element scheduling that minimized end-to-end delay for each time-sensitive flow declared. Formula and constraints pushed in z3 formulation are presented in the next chapter where implementation part is investigated.

Chapter 6

VerINS Implementation

In this chapter, an analysis is carried out regarding the implementation details of the VerINS module. Guidelines that were followed during the development phases of the module are presented to the reader. In the first section are illustrated functions that have been implemented inside the developed module and their relationships. Subsequently, the format of the XML document with which it is possible to interact with the VerINS module is deepened. Then the correct way to declare the elements characterizing the virtual topology, the physical topology and the declaration of time-sensitive flows by the network operator is illustrated. So we pass through the analysis of the constraints generation for the VNF placement part and flow scheduling part and with objective functions analysis that later are pushed inside the z3 solver. Finally, the way to present the results to the user is investigated, again through an XML document.

6.1 Framework Tasks Relationships

The functions whose coordination was managed within the VerINS module are multiple. They have different purposes and can be summarized as follows:

- XML Input File Parser: given a XML well-formed file, it provides to VerINS core module following data:
  - Physical Topology;
  - Virtual Topologies;
  - Time Sensitive Flows Properties.

  Explanation of XML structure is shown in the next section.
- VNF placement constraints generation module;
- Flow scheduling constraints generation module;
- z3 Solver.

In the following schema there is a simple representation of interactions between results coming from XML reading and two core functions of constraints generations regarding developed module:

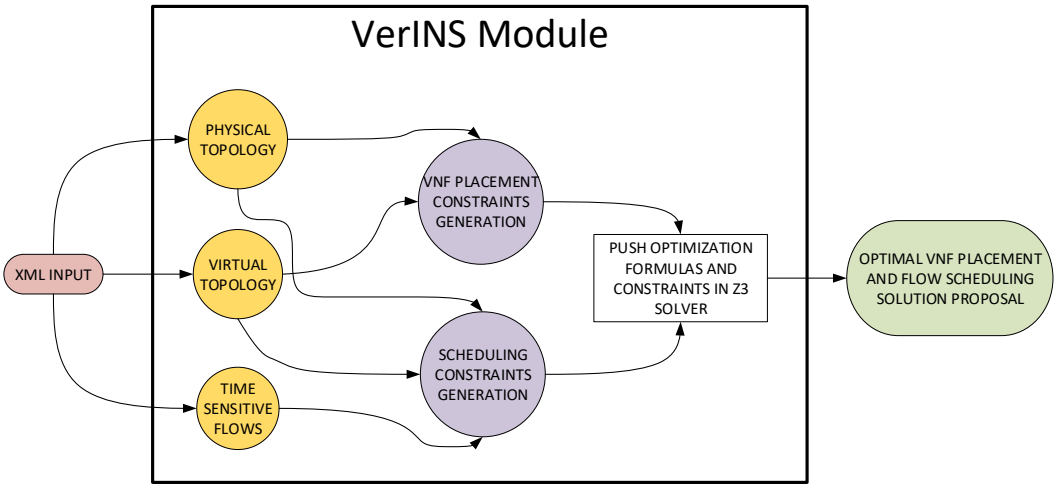

Figure 6.1. Framework Tasks Relationships

As shown in figure 6.1, physical and virtual topology data are needed for both placement and flow scheduling constraints generation modules. Besides, scheduling module needs knowledge of time-sensitive flows declaration. Otherwise, no flow scheduling constraints are generated since the module see all flows as normal flows that could be handled in a best-effort manner, without providing them specific guarantees.

6.2 XML Input Format

The XML files that are injected towards the VerINS module are validated against an SML schema (XSD, XML Schema Definition) verifying their syntactic structure correctness. The XSD is organised to represent main elements with which the developed module operates. In particular, the XML file must contain the:

- physical topology definition;
- service graphs defined on the physical structure;
- properties that must be verified by the module.

These resources are organized inside a wrapper structure, NFV. It sums up the environment with which the module has to interact. It handles elements definition and relationships, providing also a check on constraints such as the uniqueness one or referential one.

Following a graphic representation of NFV element through a tree diagram:

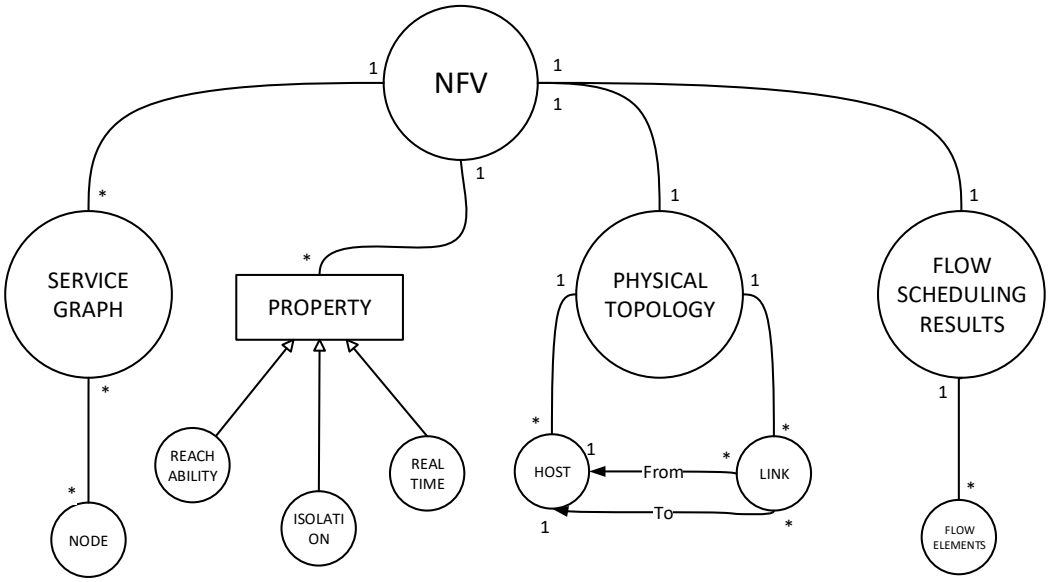

Figure 6.2. XSD Schema Tree Overview

To be noticed that lines represent relationships between two element with related multiplicity. Arrow instead indicates a specialization of the parent element.

Both VNFs placement and flow scheduling module reference Service Graph resource and Physical Topology resource. Each service graph is composed of nodes that are uniquely identified by a string. They can be used in different service graph declared on the same physical topology. The physical topology is composed of host and physical connection, link, between declared host. They are again uniquely identified inside the whole environment.

Real-time properties instead are used by the network operator to declare real-time flow that must be handled by flow-scheduling module. In particular, they are composed of:

- service graph in which property must be verified. The network operator could declare different real-time properties to be verified by VerINS module in the different Service Graph defined on the same physical topology;
- source node of the service graph from which time-sensitive flow starts. It must be referenced by its unique identifier;
- destination node of service graph to which time-sensitive flow ends. It must be referenced by its unique identifier;
- maximum acceptable latency for the declared flow. It must be indicated in timeslot unit accordingly to the formula 5.2 illustrated in the previous chapter.

6.3 Z3 formulas

The VerINS module core is composed of the solver z3, to which constraints that must be verified during the search for the optimal solution are pushed. This section describes formulas that define constraints and objective functions that are taken into account during the engine optimisation phase with a high-level notation. Formulas are split in three subsection:

- the ones related to the VNF placement problem. They are expressed through hard constraints, and so they must be necessarily verified by z3 engine;
- the ones related to the flow scheduling problem. They are again hard constraints that must be verified by z3 engine;
- the ones that must be optimised by z3 engine, minimising their associated weights.

All of these constraints are analysed within this section, with a particular focus on flow scheduling constraints, that is the VerINS core. To be noticed concerning constraints related to VNFs placement problem that some of them have been already defined in the VerifOO module. Now we proposed these constraints in a revised way, adapting them to our VerINS module. VerifOO was developed concerning the first solution proposal presented in chapter 3 to solve the problem of latency aware VNFs placement. With regard to the VerINS module development, constraints regarding maximum theoretical latency computation have been ignored since VerINS purpose is to schedule flow in the network, giving them guarantees about end-to-end reachability that must always be lower than a specific threshold.

6.3.1 Mathematical Notations

For clarification purpose, since in the next subsections a special notation is used, here is reported a simple table that sums up all used mathematical symbols:

Table 6.1: Summary of Mathematical Key Notations

Symbols	Notations
V^s, L^s	Set of physical hosts/endpoints and connections
A^s_V, A^s_L	Set of physical hosts/endpoints and connections attributes
$G^s = (V^s, L^s, A^s_V, A^s_L)$	Substrate Network
V^v, L^v	Set of virtual nodes/endpoints and connections
A^v_V	Set of virtual nodes/endpoints attributes
$G^v = (V^v, L^v, A^v_V)$	Service Graph
F^{ins}	Set of Time Sensitive Flows
C^{ins}	Set of User-Defined Time Sensitive Flow Constraints

Both the physical and the virtual environment with which the VerINS module interacts are modelled as unidirectional graphs. The sum of vertices in physical

topology is composed of hosts capable of hosting VNFs or fixed endpoints. In the viral topology, the set of vertices is instead represented by VNFs. Attribute sets contain additional information about the elements defined in the physical or virtual topology. For example, in the case of physical hosts, two attributes of fundamental importance are the storage and CPU power that are taken into account during the operations of positioning the VNFs. Or, as far as physical connection, an important parameter defined inside the link attributes set that plays a very important role during flow scheduling operation is the latency. The Time-Sensitive F network flows set is built based on network operator requests during real-time properties formulation. Each is associated with a constraint in terms of maximum acceptable latency for end-to-end flow management, which instead belongs to the set denoted by FC.

6.3.2 Placement Constraints

The mapping function of VNFs in hosts that can host them within the physical substrate can be denoted as follows:

$$M_v v^v = v^s \tag{6.1}$$

This means that a vertex of Service Graph should be mapped on a vertex of substrate topology available. Since there are two type of vertex in the virtual topology, concerning node mapping function we refer to the following notation:

$$M_n n^v = h^s \tag{6.2}$$

This mapping function must satisfy a series of requirements. VNFs deployed on one single host are executed, in fact, sharing the same physical resource as far RAM capabilities. Since in every instant, all physical resources are allocated to the management of one single flow element, CPU consumption by other VNFs executed on the same physical host is not taken into account. However, it is important guaranteeing that there is enough CPU power to meet single VNF requirements. Given:

- $|N|, |H|$ as N/H set cardinality;
- $memory(i)$, a function that compute VNF memory for element i;
- $core(i)$, a function that compute CPU core for element i;
- $supported\_vnf\_type(i)$, a function that returns set of supported VNF types;
- $max\_vnf(i)$, a function that return the maximum number of VNF that could be deployed on i-th host;
- $n_i h_j$, a boolean variable that means that i-th VNF is deployed on j-th;
- h_i, a boolean variable that shows the i-th physical host is used or not;
- $int(x)$, where x is a boolean variable, is a function that transforms true/false boolean statement in 0/1 arithmetical number;

constraints to be added to the z3 environment regarding physical host resources type and consumption are:

- **RAM usage**; each VNF requires a certain amount of available RAM to be executed properly, namely required VNFs memory of nodes deployed on a physical host must be lower that RAM host capability:

$$\forall j \in H, \sum_{i \in N} (memory(i) \times int(n_i h_j)) \leq (memory(j)) \times h_j \tag{6.3}$$

- **CPU power consumption**; each VNF requires a certain CPU power to be executed properly, namely, since CPU could be measured in core number dedicated to VNF execution:

$$\forall i \in N, \forall j \in H, (core(i) \times int(n_i h_j)) \leq (core(j)) \times h_j \tag{6.4}$$

- **maximum VNFs capability**, since number of VNFs deployed on a physical host must be lower than a specific threshold, namely:

$$\forall i \in H, \sum_{j \in N} int(n_j h_i) \leq max\_vnf(i) \times h_i \tag{6.5}$$

- **supported VNF functional type**, since a physical host could hosts only VNF of the supported type, namely:

$$\forall i \in N, \forall j \in H, n_i h_j \Rightarrow functional\_type(i) \in supported\_vnf\_type(j) \tag{6.6}$$

While, concerning fixed endpoint mapping function we refer to following notation:

$$M_e e^v = e^s \tag{6.7}$$

A virtual endpoint has a reserved host in the substrate graph, so mapping function is very simple since it is fixed.

Furthermore, the following statement must be verified to do a reliable placement module:

- **single VNF deployment**: a VNF must be deployed on one single host exactly. Namely:

$$\forall i \in N, \sum_{j \in H} n_i h_j = 1 \tag{6.8}$$

  Following this pattern, all VNFs are guaranteed to be deployed on available physical hosts at least once; however, multiple deployment of VNF on physical substrate are not to be considered.

- **VNF deployment implication on physical host state**: if a VNF is deployed on a physical host, this host must be put on active state. Namely:

$$\forall j \in H, h_j \Rightarrow \bigvee_{i \in N} n_i h_j \tag{6.9}$$

Assertion 6.3, 6.4, 6.5, 6.6, 6.8, 6.9 must be added into z3 placement engine. In the following section, flow scheduling constraints are analysed and pushed again in z3 general model. In this way, overall z3 execution takes into account placement and flow scheduling constraints. This is the novelty of VerINS project since in only one execution all constraints type are considered and a solution that satisfies all them is searched for.

6.3.3 Flow Scheduling Constraints

These constraints are responsible for the temporal scheduling of time-sensitive network flows. The first operation to be managed by the module is to find all the possible flows in the physical topology in which the virtual flow that is declared by the user when defining the properties to be satisfied by the developed module can be mapped. The network operator defines the flow by specifying the source, the destination, the reference service graph and a number of timeslots. This represents the maximum latency that the network operator is willing to accept for the management of that flow. Therefore constraints will be explained in the following paragraphs.

Virtual Path Computation

virtual path between source node and destination node is computed in virtual topology. This path is computed taking into account VNFs involved configuration. For instance, suppose having the following case:

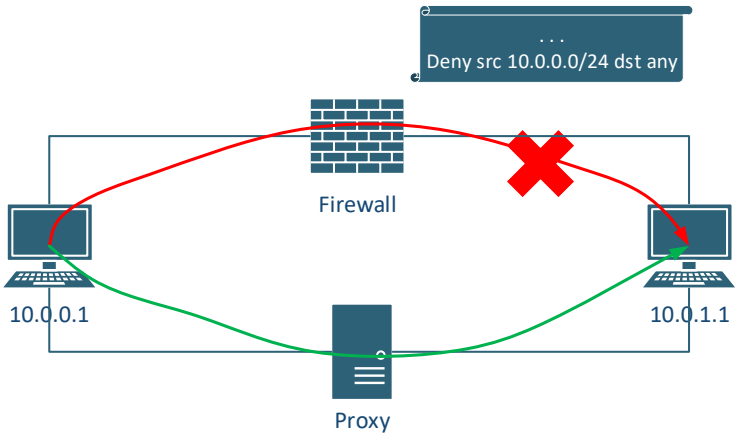

Figure 6.3. Virtual Flow Example

A firewall rule denies flow that starts from host 10.0.0.1 and ends to host 10.0.1.1. However host 10.0.1.1 is reachable through a proxy device. So module computes path between 10.0.0.1 and 10.0.1.1 as the best one in term of VNFs crossed (for instance, if another path between 10.0.0.1 and 10.0.1.1 is available but it includes more VNFs in its path, the smallest one is considered). Consequently a virtual flow is generated.

Algorithm 1 Virtual Flow Generation

```
1: procedure VirtualFlowGeneration(source, destination)
2:     COMPUTE all possible flows between source and destination
3:     for flow in just computed flows do
4:         if reachability betweeen source and destination is guaranteed then
5:             STORE flow in the nominated flows
6:     SELECT the shortest flow between the nominated ones
7: return
```

Selecting the best existing path between source and destination is a heuristic that has been introduced. If you want to obtain the best solution, all possible flows between source and destination should be discussed, if existing flows are more than one. Therefore, you can avoid overlapping flows that in the scheduling phase must be managed with particular attention. In fact, you must avoid allocating two elements whose processing requires the use of the same physical resources on the same timeslots.

Flow Mapping Constraints

Then module takes into account all possible virtual flow mappings in the physical topology. For each possible mapping, module generates automatically condition that must be satisfied. This is one of the most significant parts of the developed project. It is the moment in which we pass from virtual topology to physical topology. The actual network topology may not be known to the network operator that interacts with the VerINS module. Its interaction is limited exclusively to the virtual topology, which, based on the needs of the network operator, is deployed on the physical one. There is, therefore, a virtualization layer that separates the user from the physical topology, simplifying the interaction and maximizing its efficiency at the same time. This allows a very high degree of flexibility to be achieved. For example, it is possible to trigger automatic network reconfigurations because of the occurrence of a specific event, without a network operator action.

For instance, taking as a reference the virtual topology depicted in figure 6.3, giving the following physical topology:

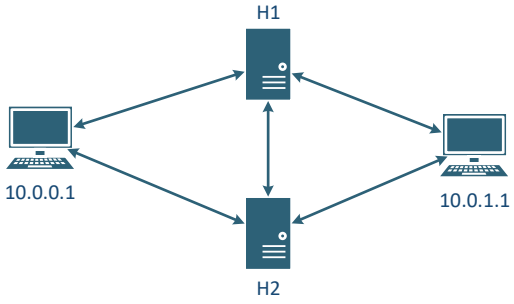

Figure 6.4. Physical Topology Example

it is clear that there are four deployment possibilities:

1. both VNFs deployed on host H1;

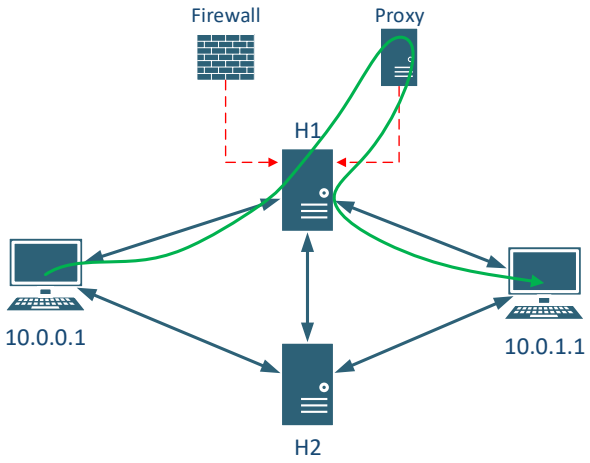

Figure 6.5. H1 Deployment Scenario

in this case flow element to be scheduled are:

   (a) 10.0.0.1 → H1, that is a physical link crossing;
   (b) Proxy action, that is a VNF execution;
   (c) H1 → 10.0.1.1, that ia a physical link crossing.

2. firewall deployed on host H1 and proxy deployed on host H2;

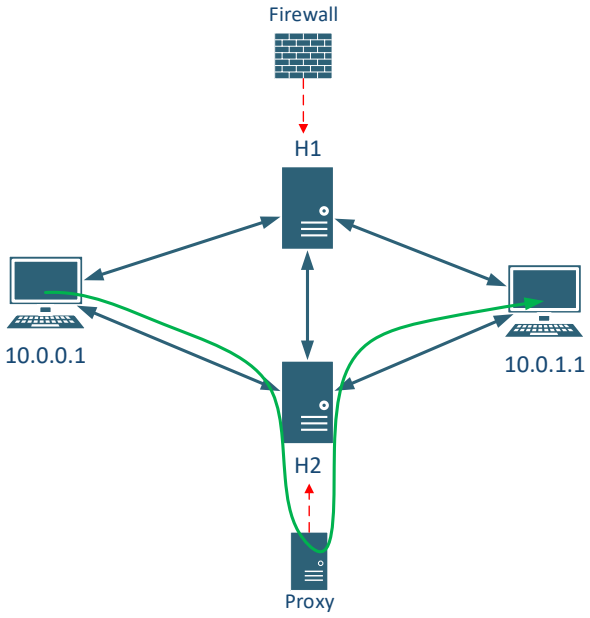

Figure 6.6. Firewall on H1 and Proxy on H2 deployment scenario

in this case flow element to be scheduled are:

(a) 10.0.0.1 → H1, that is a physical link crossing;
(b) Proxy action, that is a VNF execution;
(c) H1 → 10.0.1.1, that ia a physical link crossing.

3. firewall deployed on host H2 and proxy deployed on host H1;

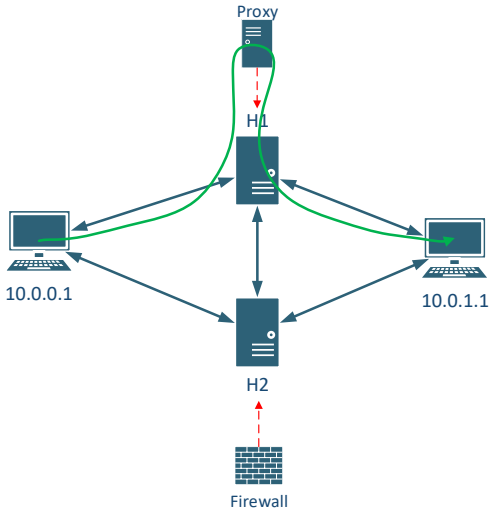

Figure 6.7. Firewall on H2 and Proxy on H1 deployment scenario

in this case flow element to be scheduled are:

(a) 10.0.0.1 → H2, that is a physical link crossing;
(b) Proxy action, that is a VNF execution;
(c) H2 → 10.0.1.1, that ia a physical link crossing.

4. both VNFs deployed on host H2;

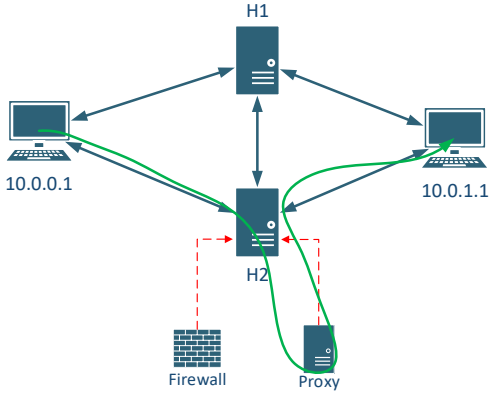

Figure 6.8. H2 Deployment Scenario

in this case flow element to be scheduled are:

(a) 10.0.0.1 → H2, that is a physical link crossing;

(b) Proxy action, that is a VNF execution;

(c) H2 → 10.0.1.1, that is a physical link crossing.

For each placement configuration and for each flow element there is a constraint that must be pushed inside the z3 model.

For instance, considering the H1 deployment scenario, calling Firewall VNF as n_1 and Proxy VNF as n_2, flow from 10.0.0.1 to 10.0.1.1 as f_1 and given a set of flow element E^{ins} which includes all the physical link defined in the physical topology, following constraints must be added in the z3 model:

$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{11} \equiv 10.0.0.1 \rightarrow H1$$
$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{12} \equiv Proxy\,Execution$$
$$n_1h_1 \wedge n_2h_1 \Rightarrow f_{13} \equiv H1 \rightarrow 10.0.1.1$$

Obviously, all possible deployment scenarios must be contemplated for each flow. Therefore, gathering all passages, following pseudocode could be taken as a reference:

Algorithm 2 Flow Mapping Constraints Generation Procedure

```
1: procedure CONSTRAINTS GENERATION(...)
2:     for TS-Flow declared as i do
3:         FIND all possible physical flow mapping
4:         for physical flow mapping as j do
5:             COMPUTE needed placement condition as cond
6:             for TS-Flow element as k do
7:                 GENERATE constraint cond ⇒ k ≡ j(n)        ▷ n is an index
8:                 PUSH generated constraint
```

We could sum up this procedure in formulas. Given:

- a TS-Flow $f \in F^{ins}$, where notation f_i denotes the i-th TS-flow element;
- a set P where each element $p \in P$ represent a possible physical flow map of the declared one;
- a function $cond(x)$ where x is a physical flow correspondent to the declared time sensitive one, returning placement condition that must be satisfied to have the x mapping;
- a function $length(x)$ where x is a flow, returning flow length in terms of flow element number:

$$\forall p \in P, \forall i \geq 0 \wedge i < length(p), cond(p) \Rightarrow f_i \equiv p_i \tag{6.10}$$

These constraints in formula 6.10 are hard constraints that must be satisfied by VerINS module.

Flow Order Constraints

To be sure of the fact that the flow scheduling operations do not alter the flow order thus calculated, it is necessary to add another constraint regarding the beginning and the end of a single flow element. Two functions have been introduced to handle flow scheduling operations. Both operate on a flow element. Given a TS-Flow $f \in F^{ins}$, where notation f_i denotes the i-th TS-Flow element:

- $start(f_i)$, which returns the first timeslots from which the flow element f_i starts to be processed;
- $end(f_i)$, which returns the last timeslots from which the flow element f_i ends to be processed.
- $duration(f_i)$, which returns the timeslots number required from the flow element f_i to be processed.

Suppose having a Time Sensitive Flow as the one depicted in the following figure, where each block represents a single flow element:

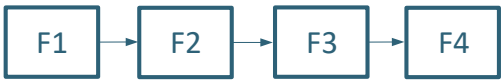

Figure 6.9. Time Sensitive Flow Elements

Scheduling order must be verified by module. Namely:

- F1 must be handled before F2, F3 and F4 processing;
- F2 must be handled after F1 execution, but before F3 and F4 processing;
- F3 must be handled after F1 and F2 execution, but before F4 processing;
- F4 must be handled after F1, F2 and F3 execution.

Following pseudocode could be taken as a reference:

Algorithm 3 Order Constraints Generation Procedure

```
1: procedure OrderConstraintsGeneration(flow)
2:     for i ← 0 to length(flow) do
3:         for j ← 0 to length(flow) do
4:             if f_i precedes f_j then
5:                 start(f_i) + duration(f_i) ≤ start(f_j)
```

We could sum up this procedure in formulas:

$$\begin{aligned} \forall f \in F^{ins}, \forall i \geq 0 \wedge i < length(f), \forall j \geq 0 \wedge j < length(f), \\ \forall i < j, start(f_i) + duration(f_i) \leq start(f_j) \end{aligned} \tag{6.11}$$

These constraints in formuala 6.11 are hard constraints that must be satisfied by VerINS module.

Flow Element Overlapping Constraints

A particular type of constraint that must be taken into account concerns the flow element scheduling that requires the same physical resources to be processed. The primary objective of the module is to avoid the formation of queues in network devices to provide guarantees regarding the maximum end-to-end delay between two or more hosts on the network. Therefore the same physical resource cannot handle multiple flow elements simultaneously. For instance, considering only flow already mapped in the physical topology, can occur the following case:

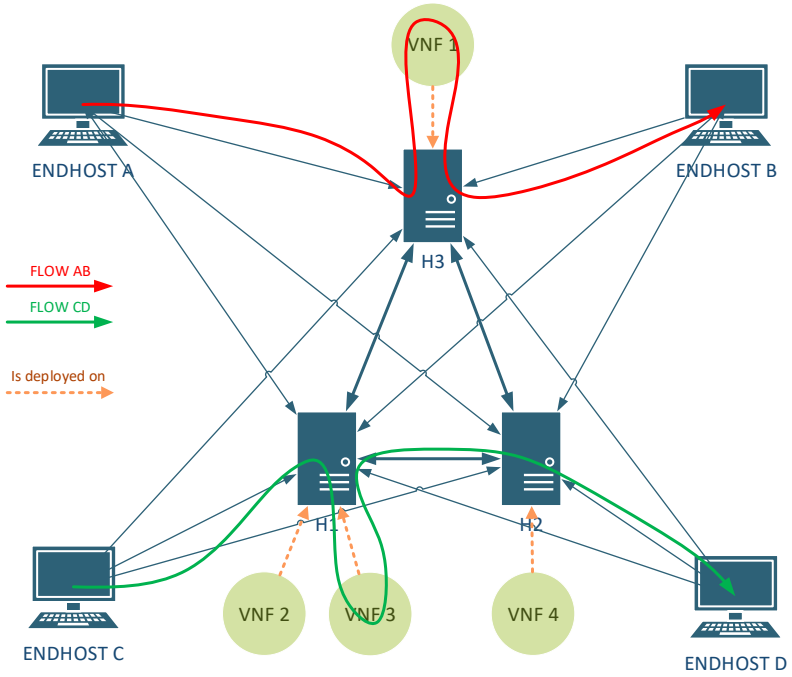

Figure 6.10. Non overlapping flows example

In the case considered in figure 6.10, the two flows in question can be scheduled simultaneously since the physical resources involved, physical links and hosts on which the VNFs are deployed, are different. Therefore, it is possible to save considerably on the number of timeslots globally used for the management of declared flows. However, this is not always feasible. It depends on source and destination host location and on middle devices configuration. In addition, heuristic introduced during virtual flow generation does not allow to search for possible virtual path that have no overlap in the physical topology. This could be an improvement that could be done as future work and extension of this framework.

However, suppose we have a case similar to the following:

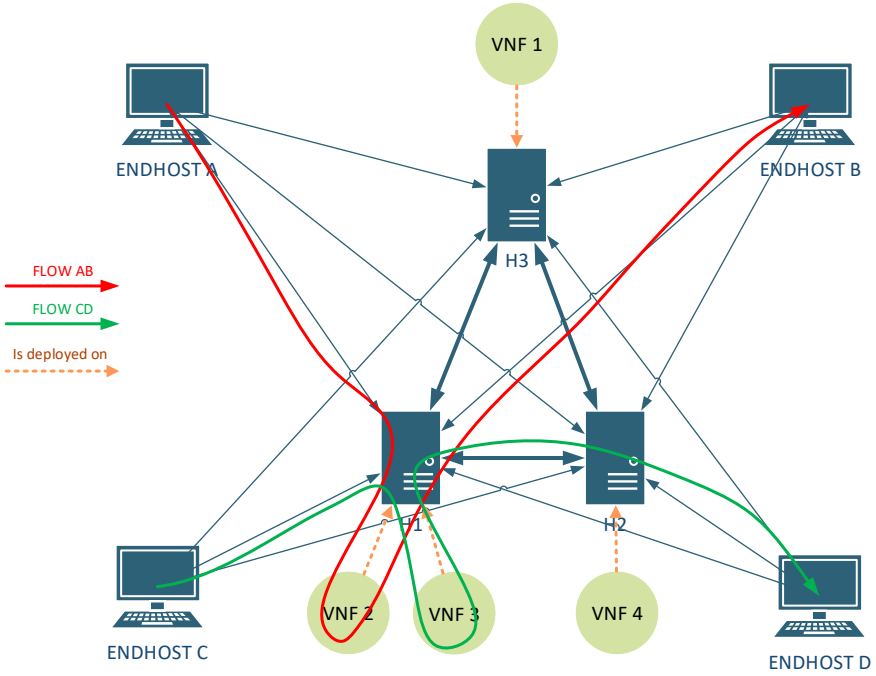

Figure 6.11. Overlapping Flows Example

In this case, VNF2 and VNF3 execution cannot be scheduled in the same timeslot since they require the same physical resource to be executed. It must be that:

$$start(VNF2) \geq end(VNF3) \vee end(VNF2) \leq start(VNF3)$$

Obviously this statement must be verified for each flow element couple that share the same physical resource within studied network environment. Generalizing, following procedure has been implemented to generate constraints concerning overlapping problem:

Algorithm 4 Overlapping Constraints Generation Procedure

```
procedure OverlappingConstratinsGeneration(flows)
    for each flow f_i in flows data structure do
        for each flow f_j in flows data structure, different from f_i do
            for each f_im flow element do
                for each f_jn flow element do
                    if f_im and f_jn are deployed on the same resource then
                        GENERATE constraint
                        PUSH constraint on z3 model
```

Constraints generated take into account relative timeslot allocation of different flow element. For instance, suppose having 2 different flow elements A and B belonging to two different flow deployed on the same physical resource. Only allowed relative allocation of this flow elements are:

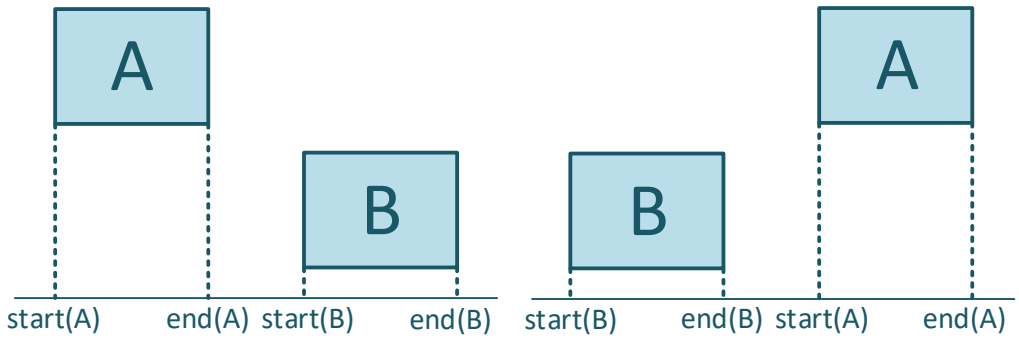

Figure 6.12. Flow Element Relationships Example

Namely, only allowed relative allocation could be expressed also with the following notation:

$$end(A) \leq start(B) \vee start(A) \geq end(B)$$

To explain these procedure in formulas we have to introduce a new function. Given a flow element x, $deployedOn(x)$ returns the physical resource where the flow element x is mapped on. So:

$$\begin{aligned} \forall x \in F^{ins}, \forall y \in F^{ins}, x \neq y, \\ \forall m \geq 0 \wedge m < length(x), \forall n \geq 0 \wedge n < length(y), \\ deployedOn(x_m) == deployedOn(y_n) \Rightarrow start(x_m) \geq end(y_n) \vee \\ end(x_m) \leq start(y_n) \end{aligned} \tag{6.12}$$

These constraints in formula 6.12 are again hard constraints that must be pushed in z3 model. They must be verified to have a reliable solution.

Total Timeslot Number Variable Constraints

A variable takes into account the total number of timeslots defined within that network environment for the management of all declared flows. The use of a variable has been made necessary because it is not possible to know in advance how many timeslots are necessary for the management of flows declared to be of time-sensitive type. By introducing a variable within our model, constraints must also be introduced to manage it correctly. First of all, this variable must be limited between

zero and maximum number of timeslots needed to handle all time-sensitive flows declared in our network environment. This number can be computed in the worst case as the algebraic sum of single flow length. Suppose having following function:

- $map(x)$, where x is a flow in virtual topology. It returns a set of flow in physical topology in which virtual flow could be mapped;
- $max(x)$, where x is a set of flow. It return the longest path between flows in set x.

Therefore, referencing total timeslot number variable with name "makespan":

$$makespan \geq 0 \wedge makespan \leq \sum_{f \in F^{ins}} max(map(f)) \tag{6.13}$$

This constraint imposes a limit on possible value for "makespan" variable.

Then, two constraints must be added to so formulated z3 model. Defining following functions:

- $last(x)$, where x is a flow. It returns last flow element of flow x;
- $bool\_to\_int(x)$, where x is a boolean variable. It returns an int correspondent to the boolean variable x. For instance, if x is *true*, function returns 1, otherwise it returns 0.

these constraints could be expressed as:

$$\sum bool\_to\_int(\bigvee_{f \in F^{ins}} (makespan, end(last(x)))) \equiv 1 \tag{6.14}$$

$$\forall f \in F^{ins}, makespan \geq end(last(f)) \tag{6.15}$$

The 6.14 states that variable "makespan" must be equal at least to the end of one flow in the physical topology, while 6.15 states that variable "makespan" must be greater or equal then the end timeslot of last flow element for each flow.

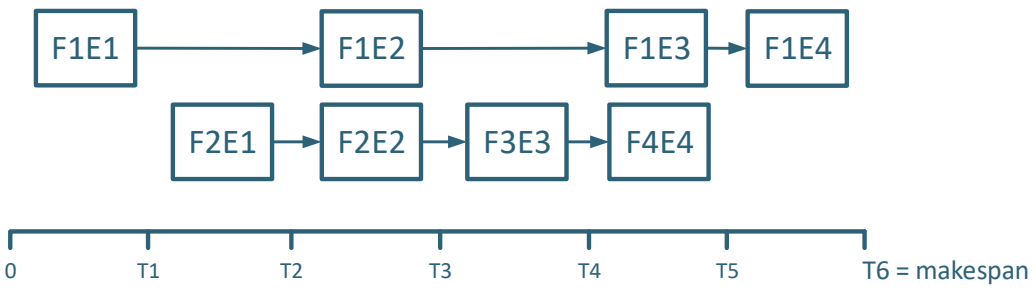

Figure 6.13. Schedulation Example

In addition, constraints that limit value field function $start(x)$ and $end(x)$, where x is a flow element must be pushed in z3 model. In particular must be that:

$$\begin{aligned} \forall f \in F^{ins}, \forall i \geq 0 \wedge i < length(f), \\ start(f_i) \geq 0 \wedge start(f_i) \leq makespan, \\ end(f_i) \geq 0 \wedge end(f_i) \leq makespan, \\ start(f_i) < end(f_i) \end{aligned} \tag{6.16}$$

Constraints in formulas 6.13, 6.14, 6.15 and 6.16 are of hard type and so must be verified by z3 engine.

Maximum Acceptable Latency Constraints

A constraint must also be pushed in z3 model regarding the maximum latency that the network operator is willing to have for the management of that specific time-sensitive flow. When declaring the time-sensitive flow, the operator can define this parameter, taking into account the duration of timeslot that has been defined within that particular network environment. Maximum end-to-end delay could be computed with formula 5.2. In fact, in the worst case:

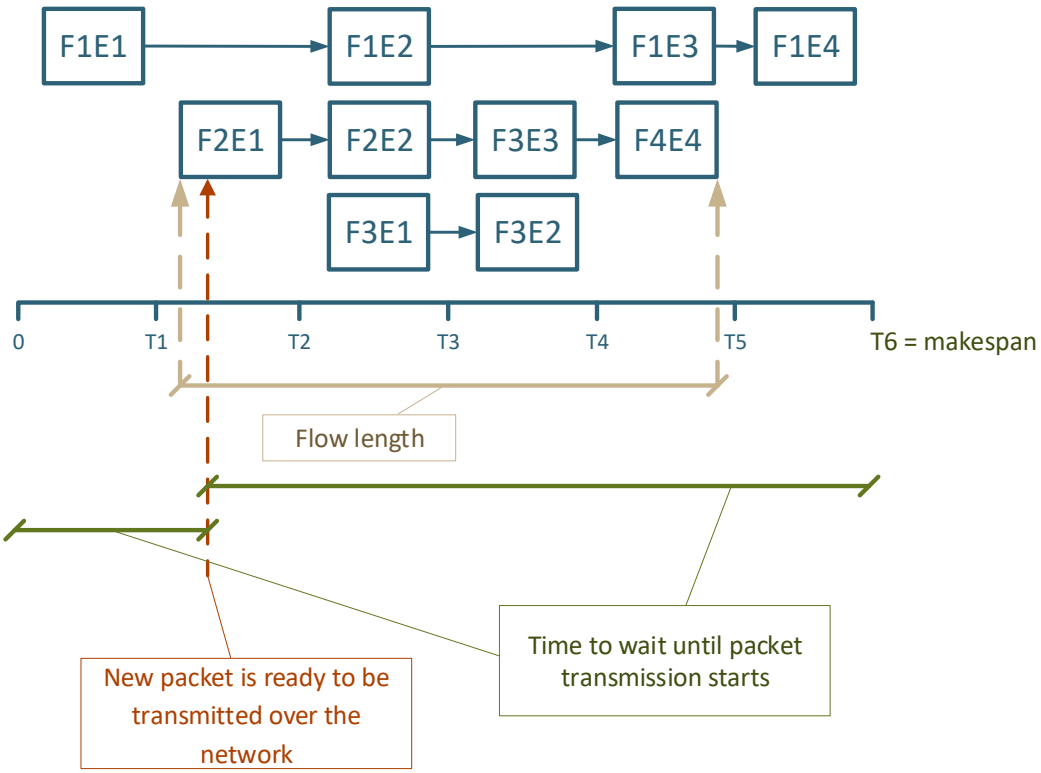

Figure 6.14. Max Latency Computation

As depicted in figure 6.14 above, in the worst case, a new packet transmission request arrives a few moments later the beginning of first flow element handling. So, packet has to wait a number of timeslot equal to "makespan" until it starts to be handled. Then packet will be delivered to destination when flow handling finishes.

Given a set of time-sensitive flow F^{ins}, a set of user-defined constraints C^{ins}, a function $find\_constraint(f)$ returning declared constraint $c \in C^{ins}$ in terms of maximum number of timeslot acceptable concerning flow $f \in F^{ins}$, following constraint must be added:

$$\forall f \in F^{ins}, makespan + end(f) - start(f) \leq find\_constraint(f) \tag{6.17}$$

These constraint in formule 6.17 are again hard constraint that must be satisfied by VerINS engine, during flow scheduling and placement operations.

6.3.4 Objective Functions

In z3, the declaration of objective functions is essential to address the search field of the solution by the module z3. The optimizer, while searching for the solution, tries to minimize the weight associated with the declaration of these functions. The objective of the VerINS module is to find an optimal VNF placement and flow scheduling solution able to minimize the end-to-end latency of the managed flows. For this reason, it was decided to include as an objective function the calculation of the maximum end-to-end latency in the worst case for each flow. Z3 optimizer have available an API that allow the declaration of a function that must be minimized. In this context, function that have been pushed in z3 model is:

$$\forall f \in F^{ins}, makespan + end(f) - start(f) \tag{6.18}$$

Besides the assurance that flow will be managed in time accordingly to user-defined constraints, VerINS module tries to minimize end-to-end as much as possible. It proposes the best solution that minimizes end-to-end latency for each time-sensitive flow. If a solution is not found, the module will return UNSAT as result, since it is not possible to find a VNF placement and flow scheduling solution that addresses all constraints seen in the previous sections.

6.4 Result Reporting

If the z3 module finds a solution, a SAT result is given. In addition, two types of results must be reported to the user:

- the VNFs placement proposed on the given physical substrate; a *nodeRef* element has been added to each host definition. It references node hosted.
- the proposed flow scheduling. A *flowElement* sequence has been added to NFV schema that contains a sequence of all scheduled flow element with timeslots allocation information.

For instance, following XML is an extracted result from a VerINS execution result concerning the VNFs placement part:

```
<Host name="host1" cpu="1" cores="4" diskStorage="50" memory="16"
  maxVNF="4" type="MIDDLEBOX" active="true">
  <SupportedVNF functional_type="FIREWALL"/>
  <SupportedVNF functional_type="CACHE"/>
  <SupportedVNF functional_type="FIELDMODIFIER"/>
  <NodeRef node="node1"/>
</Host>

<Host name="host2" cpu="1" cores="4" diskStorage="50" memory="16"
  maxVNF="4" type="MIDDLEBOX" active="true">
  <SupportedVNF functional_type="FIREWALL"/>
  <SupportedVNF functional_type="CACHE"/>
  <SupportedVNF functional_type="FIELDMODIFIER"/>
  <NodeRef node="node2"/>
</Host>
```

Listing 6.1. Placement Result XML Example

As can be easily seen, node1 and node2 are deployed respectively on host1 and host2. And this is an XML extracted again from a VerINS execution result concerning the flow scheduling part:

```
<FlowElements>

  <FlowElement graph="0" flow="from_nodeB_to_nodeD" type="connection"
  description="from hostB to host2" startOnTimeslot="0"/>
  <FlowElement graph="0" flow="from_nodeB_to_nodeD" type="vnfExecution"
  description="node2" startOnTimeslot="1"/>
  <FlowElement graph="0" flow="from_nodeB_to_nodeD" type="connection"
  description="from host2 to hostD" startOnTimeslot="2"/>


  <FlowElement graph="0" flow="from_nodeA_to_nodeC" type="connection"
  description="from hostA to host1" startOnTimeslot="0"/>
  <FlowElement graph="0" flow="from_nodeA_to_nodeC" type="vnfExecution"
  description="node1" startOnTimeslot="1"/>
  <FlowElement graph="0" flow="from_nodeA_to_nodeC" type="connection"
  description="from host1 to hostC" startOnTimeslot="2"/>

</FlowElements>
```

Listing 6.2. Flow Scheduling Result XML Example

It contains a list of flow elements and for each one is indicated the timeslot number from which its management starts.

Chapter 7

Test and Validation

This chapter is dedicated to the presentation of a potential architecture design able to implement in reality what has been theorized in the previous chapters. So a first section is dedicated to an illustration of the key concepts that guide the architectural design phase. In particular, focus is on technologies needed to assure a proper functioning of the network environment. Finally, some test cases are presented. They have been used only to show how VNF placements and flow scheduling illustrated previously work. So a performance evaluation have been conducted to show module weakness.

7.1 Real Architecture Deployment

Bringing as reference the work done in [15], a very similar scenario could be used to deploy our solution in a real environment. Technologies exploited are the ones introduced in the chapter 3, concerning the section 3.3:

- PTP, Precision Time Protocol, which provides a clock synchronization environment between devices involved in the network considered;
- Intel DPDK, a technology able to provide a direct mapping between a network interface and a process in the userspace.

In next subsections, cooperation between these two technologies is investigated in-depth, showing an innovative architectural design that could be drawn up to host solution previously presented. Obviously, they are integrated in an NFV/SDN ready network environment. It means :

- that devices involved have the capability to interact with a centralized network controller, where the control plane is executed, including all algorithms seen in the previous chapter, through a special dedicated protocol. It allows to make this control information exchange easier;
- the centralized controller must be able to orchestrate and deploy VNFs among different devices accordingly to the principles of NFVi (Network Function Virtualization Infrastructure).

7.1.1 OVS with DPDK

The OpenFlow standard is managed by the Open Networking Foundation (ONF). It has been introduced to propose a real implementation of an SDN network architecture. It allows the network controller to administer the forwarding behaviour of controlled devices through a control plane declared on it. It is a real implementation of the subdivision between the control plane and data plane. Control plane is executed on the centralized host while data plane is executed on the peripheral controlled devices.

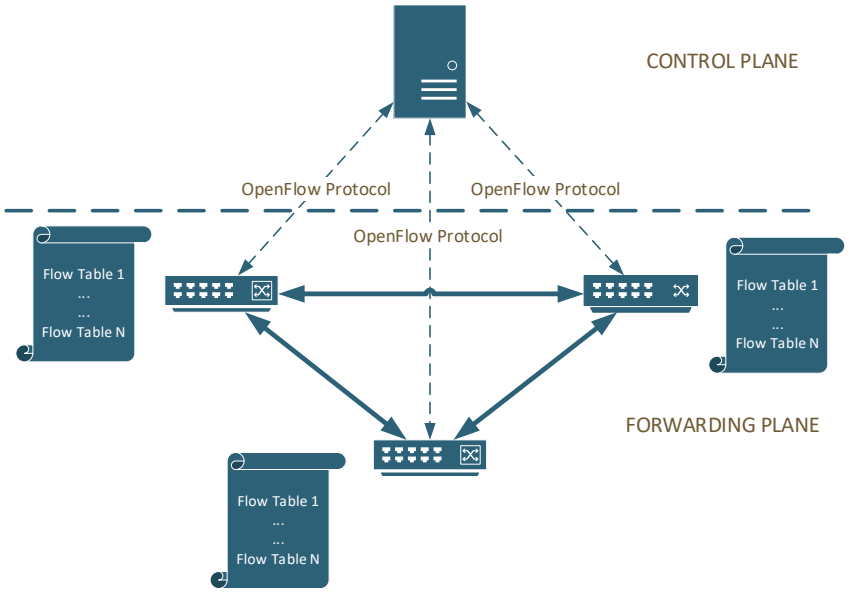

Figure 7.1. Open Flow Architectural Design

In figure 7.1 above is clear the decoupling between control place and forwarding plane. Each controlled device can interact with the centralized controller through the OpenFlow protocol. Controller makes decisions and push on controlled devices different Flow Table, which gathers the forwarding behaviour against each different type of flows seen in the studied topology. Controlled devices, besides, can report network status information to the centralized controller which, whenever is needed, push new Flow Table on the controlled devices to face, for instance, a failure occurred in the network.

Devices controlled are usually physical switches with have the capability to interact with the OpenFlow controller, in addition to standard forwarding capability already implemented in standard physical switches. In a virtualized environment, instead of physical switches, virtual switches are used to interconnect VNFs deployed on each host. Thanks to Intel DPDK technology, it is possible to built an OVS in the userspace of the operating system host.

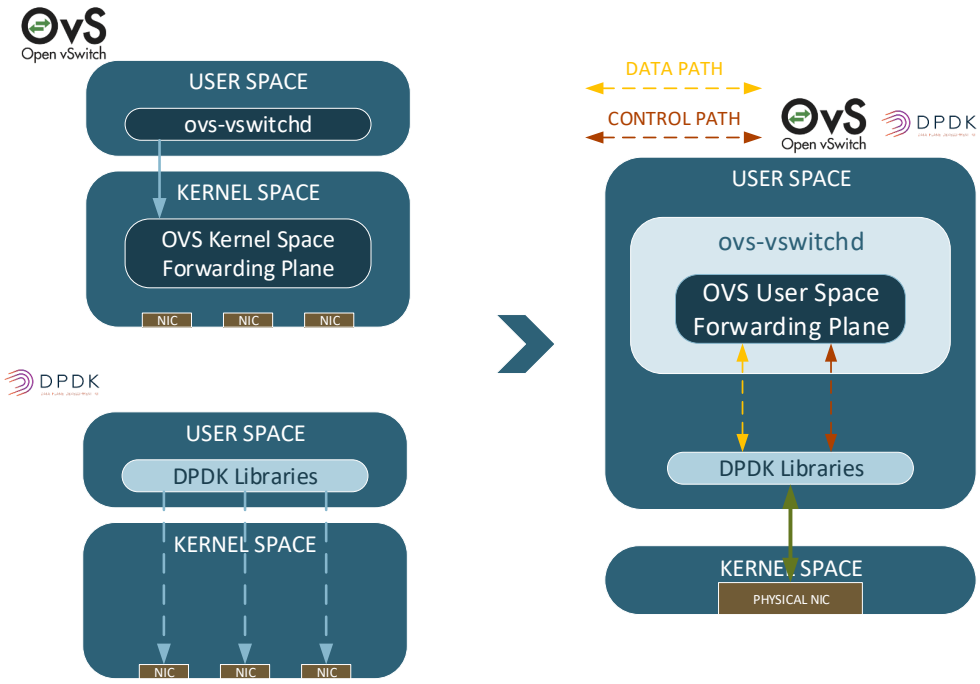

Figure 7.2. OVS with DPDK technology

In this way virtual switch is able to interact with the network controller following the rule dictated by OpenFlow protocol. At the same time, virtual switch is able to interact with VNFs deployed on the host considered. As a matter of fact, this is one of the most important reason that bring to the use of virtual switching in the networking world. More VNFs could be deployed on the same physical host. Instead of mapping each VNF on a physical host interface, a virtual switch could be use. In this way, you are able to bypass limits imposed by the number of physical interfaces installed on the physical host.

7.1.2 Middlebox architecture

In the general middlebox architecture, we also have to consider VNFs deployment. VNFs share data path with the Open Virtual Switch process through DPDK libraries. Packet forwarding through VNFs deployed on middlebox studied is managed by the Open Virtual Switch forwarding plane, in its turn controlled by central network controller shared through different network devices. Physical middlebox interface is mapped to Open Virtual Switch process trough DPDK libraries.
So, following architecture could be considered as the possible one capable to host previously defined environment, as middleboxes configuration:

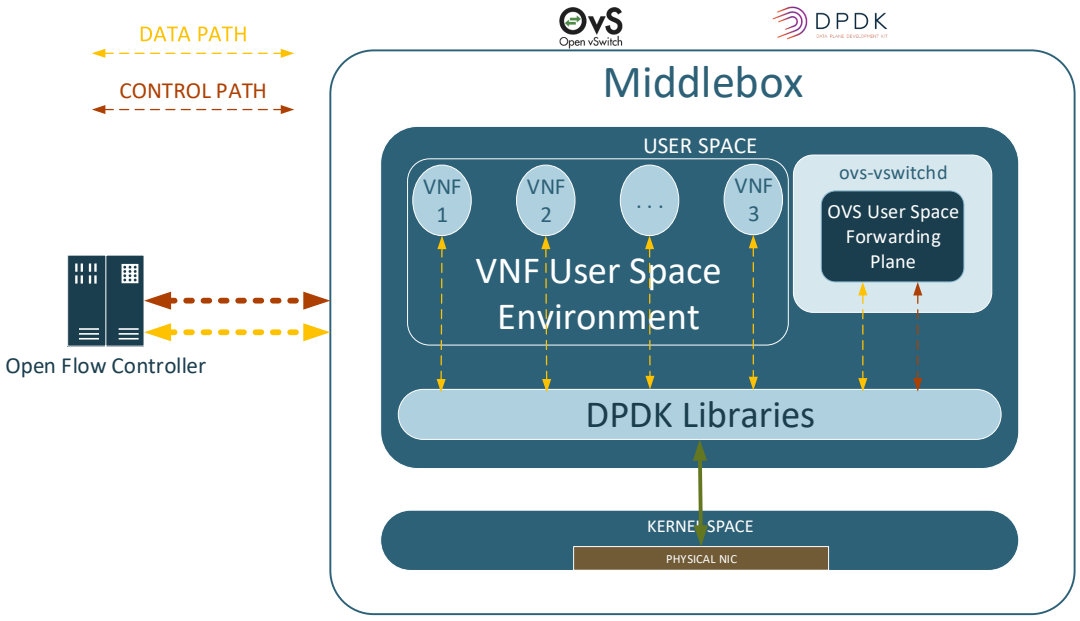

Figure 7.3. OVS and DPDK architecture with VNFs

7.1.3 PTP Deployment

One of the most challenging issue to be faced in the architectural design phase also highlighted in the previously shown work [15], is the PTP (Precision Time Protocol) synchronization daemon implementation. Deploying this synchronization layer is very important in our designed network environment. Algorithms previously described concerning flow-scheduling engine work correctly thanks to the fact that all devices clocks within the network are synchronized. If some devices have clock not synchronized with general master clock defined in the studied network environment, not managed situation could be raised, and the system goes into an unstable state. A dedicated physical network interface is needed because of accuracy required by PTP. As a matter of fact, network traffic handled with high-priority could negatively impact the PTP latency measurement.

7.1.4 Endpoint architecture

The endpoint architecture is very similar to the middlebox architecture. Endpoints must know in which instant they are allowed to start the packet transmission. So a data exchange with general network controller is needed to retrieve these kinds of information. Besides, it is mandatory that again the device clock is synchronised to a master clock; otherwise, algorithm previously described might not work properly.

7.1.5 Physical Architecture Environment

Here is illustrated a potential physical architecture environment. To be noticed that all devices involved in the network must have clock synchronized with a master clock to meet VerINS minimum requirements. Algorithms described in the previous chapter are run into the general network controller.

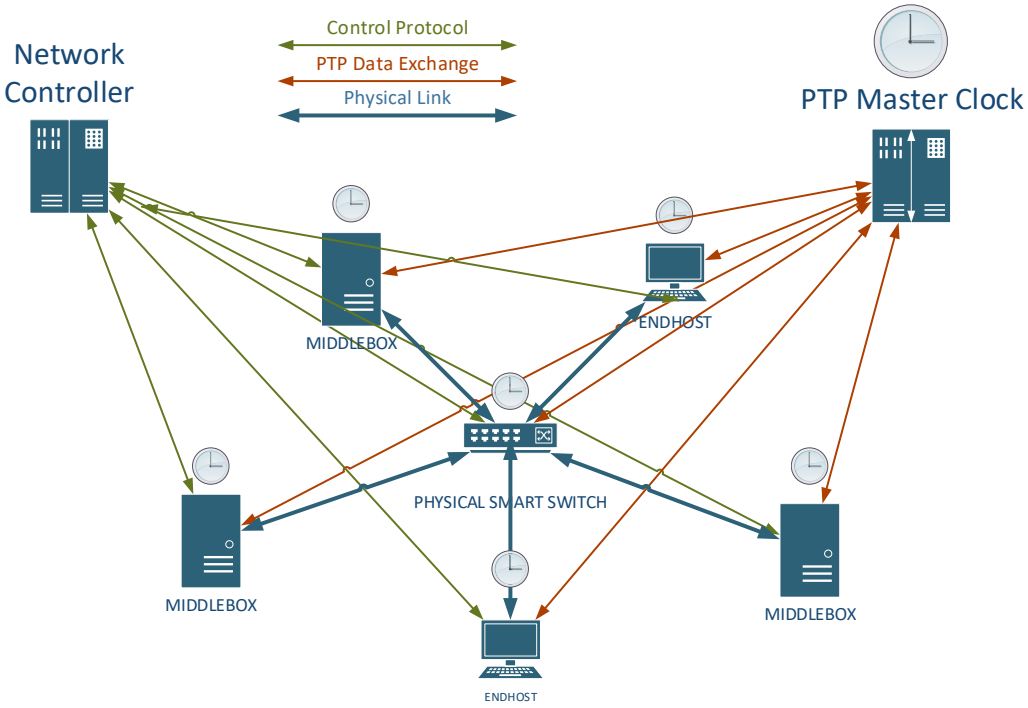

Figure 7.4. Physical Architecture Example

7.2 Test and Validation

The functioning of the VerINS module thus developed has been validated through some simple scenarios which are summarized here. First, the physical topology referred to during the test phase is presented. So a virtual topology is defined. It is the one that the network operator wants to deploy on the given physical substrate. Then five different scenarios are proposed, changing the number of declared time-sensitive flows and relative flows position in the virtual topology. In this way, it is possible to prove the correct functioning of the module developed. Solutions of VNFs placement and flow-scheduling directly suggested as output by the module itself is then presented. To be noticed that tests have as the only objective of demonstrating the module algorithm functioning and performance.
The studied physical topology includes six endpoints that are used during end-to-end communication in a network environment. Each endpoint appears both in virtual and physical topology since its placement is fixed. Instead, for simplicity,

only three physical middleboxes are defined. They can host VNFs defined in the virtual topology. Physical middleboxes resources have been defined through an XML file injected in the VerINS module. The XML file contains for each middleboxes, its available resource and capability concerning the VNFs placement. Among these:

- RAM capability;
- CPU power;
- max VNFs number that the middlebox is able to support;
- list of VNFs functional type able to support concerning software installed on the middlebox.

These features are taken into account during placement operations by VerINS module, generating proper constraints, as shown in the previous chapter.
In addition, we suppose that each endpoint is connected to each middlebox defined in the physical environment for high availability purpose. If a failure occurs in one physical link that connects the endpoint to the middlebox, a network reconfiguration is triggered on the network controller. It happens since some VNFs could not be reachable after the failure. Reconfiguration requires obviously the rerunning of all algorithms defined. So a new potential VNFs placement scheme and flow scheduling table is proposed. This operation takes times since all controlled hosts must be reconfigured to deploy the new configuration. The reconfiguration phase could be optimized introduction a smart logic that considers the entity of failures and triggers a reconfiguration only when there is a real reachability problem between the declared flows.
Concerning virtual topology, VNFs involved in the network environment are represented in a more abstract way, showing relationships between communication endpoints and other VNFs in the studied environment.
In the presented flow scheduling result, we assume that timeslots duration is equivalent to the longest flow element execution in the defined environment. In addition, we assume that all flow element require exactly one timeslot to be completely executed. This limitation can be overcome by setting a flow element duration different to the standard length (one timeslot). Results are presented, giving to the reader:

- placements result in tabular form;
- flow elements timeslots allocation in tabular form;
- graphical representation of time sensitive flow scheduling.

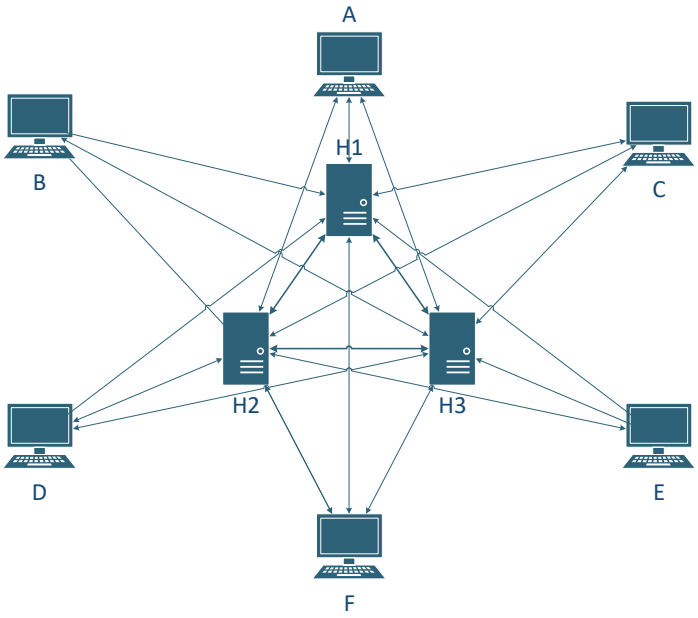

Figure 7.5. Physical Topology Test Environment

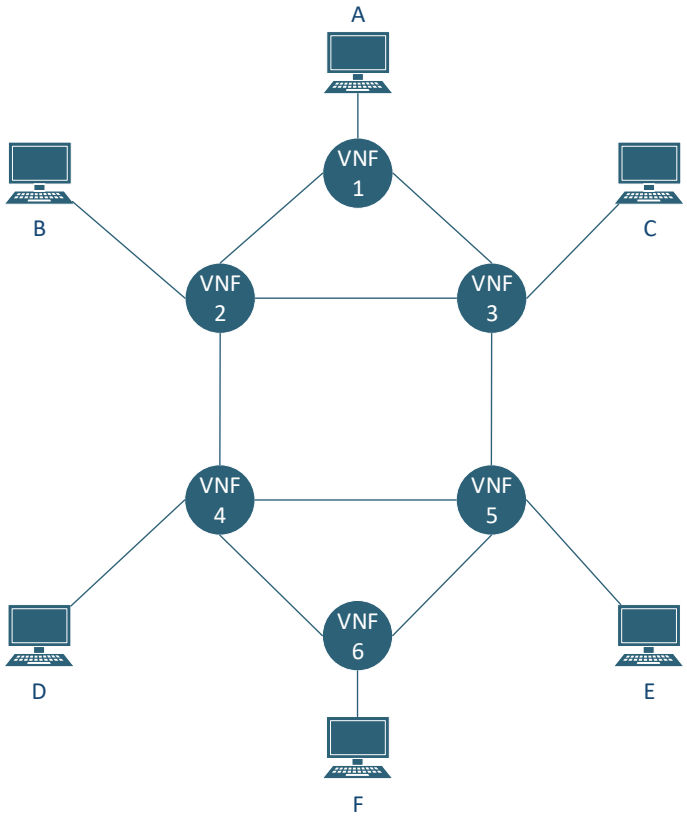

Figure 7.6. Virtual Topology Test Environment

7.3 Service Graph

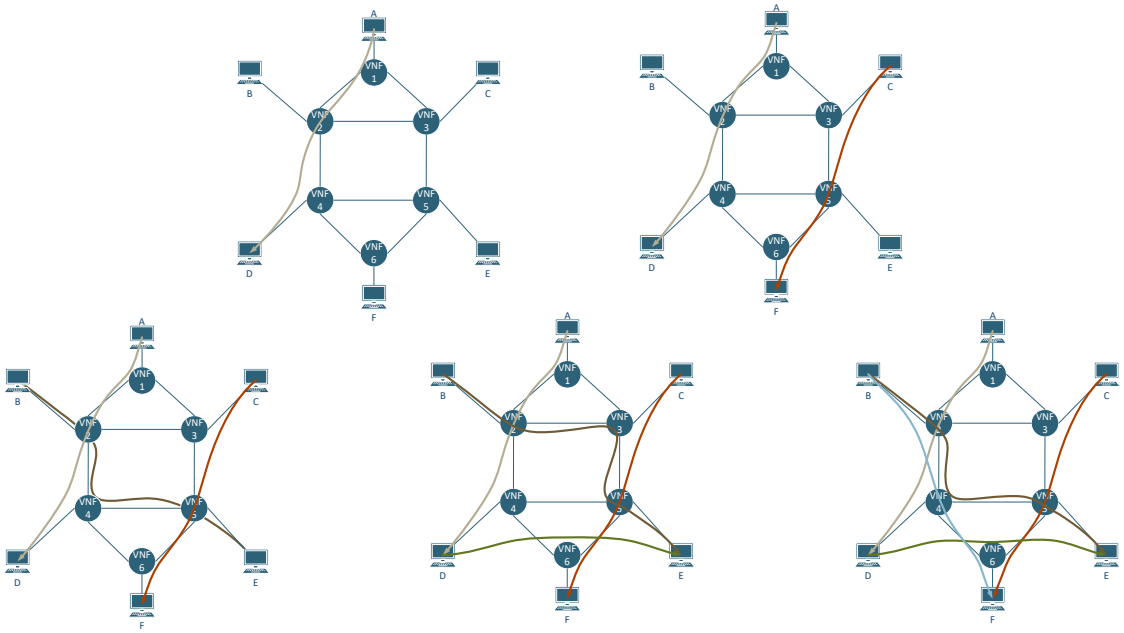

Figure 7.7. All Test Scenarios

7.3.1 First Scenario

In this first studied scenario, a single flow is considered. Each flow element is allocated to an available timeslot. Total number of timeslots needed to address this topology is equivalent to the number of flow elements in which the declared time-sensitive flow is composed.

Table 7.1: First Scenario Placement Results

Physical Host	VNFs Deployed
H1	VNF1, VNF2, VNF3, VNF4, VNF5
H2	-
H3	VNF6

Table 7.2: First Scenario Flow Scheduling Results

Timeslots	Flow AD
T0	A to H1
T1	VNF1
T2	VNF2
T3	VNF4
T4	H1 to D

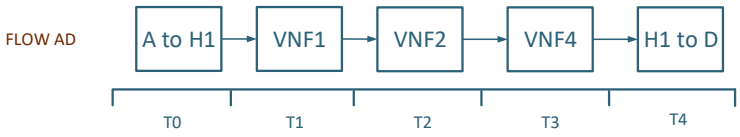

Figure 7.8. First Scenario Flow Scheduling Results

7.3.2 Second Scenario

In this second scenario, two time-sensitive flows are declared to be handled guaranteeing them a maximum end-to-end latency between host involved in communication. To be noticed that declared time-sensitive flows have no overlap in their path. This is why they have been scheduled to be handled in parallel. In fact, VNFs involved in their path, are deployed on different hosts. In this way, there are no flow elements of first time-sensitive flow that have intersections with some flow elements of second time-sensitive flow. As can be seen clearly from the tabular flow scheduling representation, no more timeslots than the previous scenario are needed to address this case.

Table 7.3: Second Scenario Placement Results

Physical Host	VNFs Deployed
H1	-
H2	VNF3, VNF5, VNF6
H3	VNF1, VNF2, VNF4

Table 7.4: Second Scenario Flow Scheduling Results

Timeslots	Flow AD	Flow CF
T0	A to H3	C to H2
T1	VNF1	VNF3
T2	VNF2	VNF5
T3	VNF4	VNF6
T4	H3 to D	H2 to F

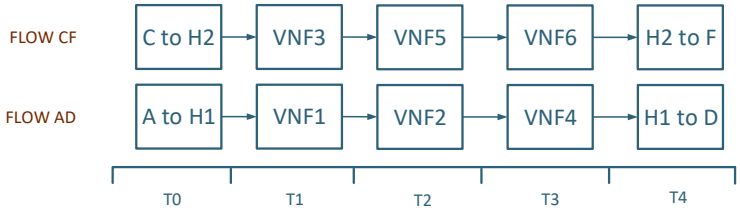

Figure 7.9. Second Scenario Flow Scheduling Results

7.3.3 Third Scenario

In this third scenario, a flow that in the virtual topology have intersections with the already declared ones have been introduced. The module gives a new placement solution. It allows flows to be handled in parallel, although their mapping in physical topology is not the optimal one. However, only one more timeslot than the previous scenarios is needed to address all time-sensitive flows in this topology.

Table 7.5: Third Scenario Placement Results

Physical Host	VNFs Deployed
H1	VNF2, VNF4
H2	VNF1
H3	VNF3, VNF5, VNF6

Table 7.6: Third Scenario Flow Scheduling Results

Timeslots	Flow AD	Flow CF	Flow BE
T0	A to H2	C to H3	B to H1
T1	VNF1	VNF3	VNF2
T2	H2 to H1	VNF5	VNF4
T3	VNF2	VNF6	H1 to H3
T4	VNF4	H3 to F	VNF5
T5	H1 to D	-	H3 to E

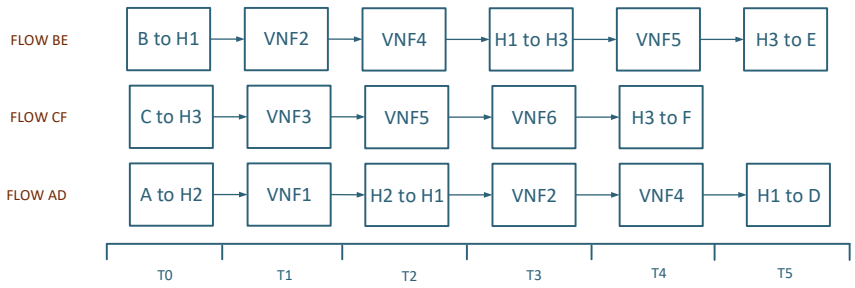

Figure 7.10. Third Scenario Flow Scheduling Results

7.3.4 Fourth Scenario

In this scenario, DE time-sensitive flow has been added to the already declared ones. To be noticed that, accordingly to VNFs configuration, two possible paths with the same cost are available for already declared flow BE:

- the one used which involves VNF2, VNF4, and VNF5. This path avoids a possible overlap with flow CF, since VNF3 is not crossed;

- the one used in the current solution, which involves the use of VNF3, generating an additional overlap to be handled with flow CF.

Solutions may be different accordingly to the path chosen. In the solution shown below, seven timeslots are needed to handle all declared time-sensitive flows, since path which generate a one more intersection in the virtual topology regarding VNFs utilization has been chosen.

Table 7.7: Fourth Scenario Placement Results

Physical Host	VNFs Deployed
H1	VNF1
H2	VNF2, VNF3, VNF6
H3	VNF4, VNF5

Table 7.8: Fourth Scenario Flow Scheduling Results

Timeslots	Flow AD	Flow CF	Flow BE	Flow DE
T0	-	C to H2	-	D to H3
T1	A to H1	VNF3	B to H2	VNF4
T2	VNF1	H2 to H3	VNF2	VNF5
T3	H1 to H2	VNF5	H2 to H3	H3 to E
T4	VNF2	H3 to H2	VNF4	-
T5	H2 to H3	VNF6	VNF5	-
T6	VNF4	H2 to F	H3 to E	-
T7	H3 to D	-	-	-

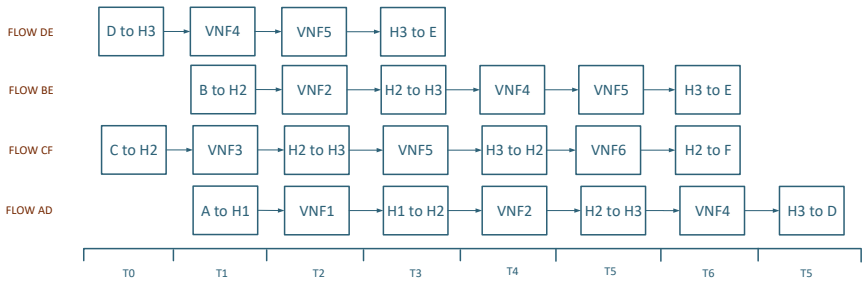

Figure 7.11. Fourth Scenario Flow Scheduling Results

7.3.5 Fifth Scenario

In this scenario, one more flow has been added. But, as can be seen clearly from the tabular timeslots allocation representation, only six timeslots have been used. This could be misleading since, in the previous scenario with one less flow than the current scenario, seven timeslots were needed to handle all declared time-sensitive flows. It happens because no constraints on virtual flow computation are put. In fact, network operator in flow definition phase, gives source and destination of

time-sensitive flow. Virtual path is computed automatically by module, accordingly to VNFs configuration. Since multiple paths are available for flow BE, algorithm choose arbitrarily between possible ones, without considering all possible solutions. This is a heuristic introduced to prune solution tree.

Table 7.9: Fifth Scenario Placement Results

Physical Host	VNFs Deployed
H1	VNF3, VNF5
H2	VNF1, VNF2
H3	VNF4, VNF6

Table 7.10: Fifth Scenario Flow Scheduling Results

Timeslots	Flow AD	Flow CF	Flow BE	Flow DE	Flow FB
T0	A to H2	C to H1	-	D to H3	-
T1	VNF1	VNF3	B to H2	VNF4	F to H3
T2	VNF2	VNF5	VNF2	H3 to H1	VNF6
T3	H2 to H3	H1 to H3	H2 to H1	VNF5	VNF4
T4	VNF4	VNF6	VNF3	H1 to E	H3 to H2
T5	H3 to D	H3 to F	VNF5	-	VNF2
T6	-	-	H1 to E	-	H2 to B

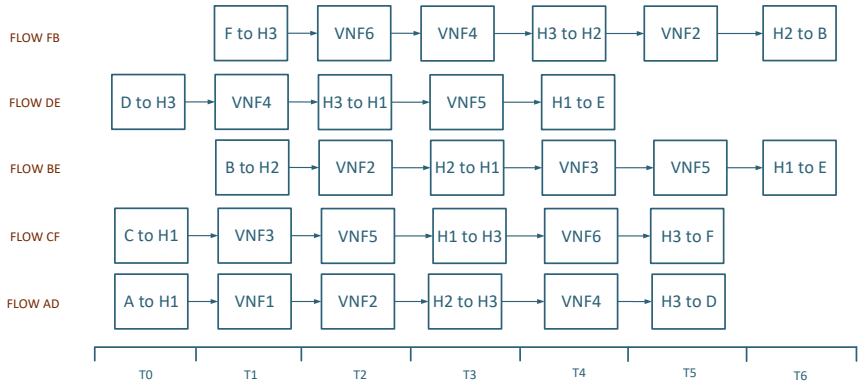

Figure 7.12. Fifth Scenario Flow Scheduling Results

7.4 Performance Summary

Below is shown a histogram that sums up performances reached by the developed module. Execution time has been split into two categories:

- constraint generation time; this is the time required by module to generate placement and flow-scheduling constraints investigated in the previous chapter.

- z3 validation time; this is the time required by z3 engine looking for an optimal solution which satisfies all constrains pushed on the z3 module.

The first one represents the time needed by module to generate constraints to be pushed to z3 model. The second one, instead, is referred to the time required by z3 engine to search an optimal solution that satisfy all constraints pushed in the previous phase.

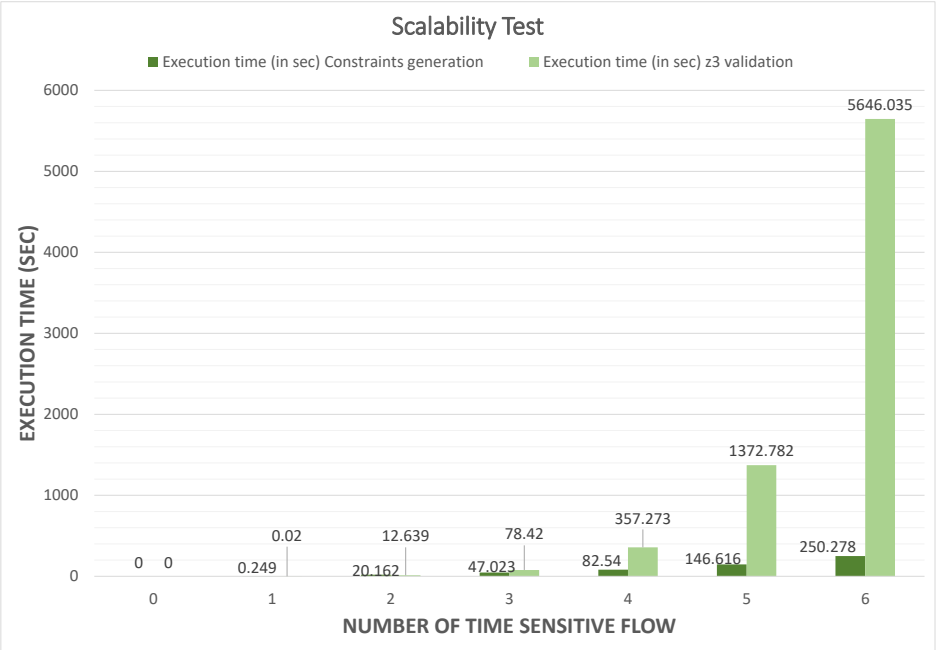

Figure 7.13. Performance Results

Readers could see that execution times indicated in the figure are not acceptable in a network environment which have a strict demand on real-time requirements. Times grows exponentially up due to the number of constraints to be pushed to z3 engine. Most time consuming operations are:

- overlapping constraints generation;
- order constaints generaration;
- physical mapping of time-sensitive declared flow.

These are very complex operations and require too many efforts in terms of computational resources to be properly executed. Algorithmic complexity depends on the number of hosts involved in the studied topology, number of declared time-sensitive flows and length of each time-sensitive flow. Future work could be organized to develop module in order to face these scalability issues as explained in the following chapter.

Chapter 8

Conclusion and Future Works

The world of industry is facing a great revolution that is changing its unique aspects. For many years, the industrial systems architecture and their interconnections have evolved following guidelines that are different from those that have marked the progress of a general-purpose interconnection system architecture. Industrial systems often play a fundamental role in critical security operations. This is why it is often required to achieve specific requirements that can guarantee a high degree of reliability of the systems involved and their interconnections.
Increasingly numerous are the industrial networks that involve a large number of devices located throughout the world; it is, therefore, necessary to provide them a high degree of flexibility and dynamism far from the canons that have characterized traditional industrial networks. It is, at the same time, fundamental respecting safety and reliability requirements to guarantee their correct functioning.
The critical issues emerging in such an environment have, therefore, been analyzed in parallel. There were two major ones: security and low latency end-to-end achievement. Both must be addressed to ensure the overall functioning of the system. The thesis work focused on the formal verification of the real-time achievement of hosts in the network. Making improvements in terms of security to the global network environment implies the introduction of middlebox that can analyze traffic, keep track of it and possibly block flows considered malicious. The logic of the blocks can vary depending on the middlebox referred to, but they all have in common the fact of introducing non-negligible latency. We, therefore, focused on the need to guarantee real-time achievement requirements for networked hosts that in a critical environment such as the industry remain of primary importance. Given the developments that have characterized the world of networking in recent years, there have been several solutions proposed to manage the problem of real-time achievement of hosts on the network, taking full advantage of the potential offered by the new SDN and NFV technologies. These allowed to obtain a more abstract and globalized view of the controlled network environment and therefore, the development of more flexible and efficient solutions. Two solutions were analyzed. The first proposal in [14] focuses on the development of a framework able to guarantee a formal verification of the real-time achievement of hosts in the network as a result of a placement of smart VNFs that takes into account the latency of end-to-end flows. The second proposal in [15] has a completely different approach to the problem. In fact, it seeks to address the requirement of real-time achievement by proposing

a time schedule of the flows in the network. Both solutions were taken into consideration during the thesis work. The research activity started precisely from the analysis of the limitations of the solutions held as a reference point, proposing an extension aimed at allowing the placement of the VNFs while ensuring a limit on the end-to-end latency of the flows declared of type time-sensitive network. The first phase of project development was focused on the design of the problem, formalizing it as a research or optimization problem since we want to find a solution to the VNF placement problem by proposing a time schedule of time-sensitive flows. Therefore the constraints that must be taken into consideration during the placement of VNFs and scheduling have been analyzed. Finally, thanks to the use of the z3 optimizer, some simple test cases have been solved to verify their correct functioning. Excellent results have been achieved. With the developed module we were able to solve some simple network topologies, proposing an optimal solution in terms of VNF positioning on the physical substrate and scheduling of network flows in order to minimize end-to-end latency of time-flows sensitive. Therefore the first validation phase can be considered successfully passed. However, the performances that have been obtained do not yet allow the deployment of the module within the common management and orchestration (MANO) software of the VNF on the net. In fact, it is essential to keep the algorithm execution times extremely low, so as to be as transparent as possible for network operations. Due to the very nature of the problem, it was necessary to insert a considerable number of constraints in the model. The generation of constraints is already a burdensome operation in itself. Experimentally we have seen that the execution time increases linearly with the number of time-sensitive flows declared in the network. Furthermore, the greater the number of constraints introduced in the optimizer model, the longer the time required for finding the optimal solution. From the performance analysis conducted, we have seen how the execution time due to the validation by the optimizer used increases exponentially as the number of time-sensitive flows declared on the network increases. Furthermore, in the current version of the module, the user has the possibility to declare the communication endpoints. The module automatically calculates the path in the virtual topology as the shortest path existing between those endpoints. However, this implies that the proposed placement and the flow-scheduling solution is a suboptimal solution since it may exist some other mapping of the flows that does not foresee the use of the shortest paths between the declared endpoints but globally uses fewer timeslots, so it is to be considered as a better solution.

It is, therefore, possible to intervene by making improvements regarding the generation of constraints which are then inserted into the optimizer. There are data structures provided by the various optimizers that can already generate internally or such constraints that some constraints are met implicitly without the need to generate them. This could lead to general improvements in the performance of the developed module.

Finally, we can plan to address the problem of finding the optimal solution, taking into account the various paths available for each flow on the virtual network topology rather than considering the best one in terms of length. This could lead to better solutions and therefore, to obtain lower latencies on the network. This work has allowed us to demonstrate how it is possible to combine the problem of VNF placement and network flow scheduling in a single instance. In a world where

networking is increasingly oriented towards virtualization, adding a formal verification of achievement requirements can lead to a significant acceleration in the implementation of these new technologies in a world that for years has advanced much more slowly due to nature criticism of the devices involved. In the wave of the revolution that is involving in recent years, industry requires greater flexibility and dynamism. Integrating NFV / SDN-type technologies within this environment can undoubtedly bring benefits, but it must never lose sight that certain requirements must necessarily be satisfied with the design of a safe and reliable network.

Appendix A

RestAPI Developer's Guide

In this chapter, an overview of API design strategy will be presented. APIs have been developed to interact easily with VerIns module that includes both placement and flow scheduling optimization engine, core of previous chapters. The project also includes Swagger documentation that shows how to interact with APIs correctly.

A.1 New XML Features

New elements have been introduced to interact properly with the VerINS module:

- **FlowElement**: it has been added to address flow scheduling module implementation.

  In particular, they are used to report results from VerINS module execution to the end-user. Below the XML description of Flow Element is reported:

```
<xsd:element name="FlowElement">
<xsd:complexType>
<xsd:attribute name="graph" type="xsd:long" />
<xsd:attribute name="flow" type="xsd:string" />
<xsd:attribute name="type" type="xsd:string" />
<xsd:attribute name="description" type="xsd:string" />
<xsd:attribute name="startOnTimeslot" type="xsd:int" />
</xsd:complexType>
</xsd:element>

```

Listing A.1. Flow Element XML schema

  A flow element could be of two types:

  - **Physical connection crossing**: it is intended as physical link crossing in physical topology. Connections declared in the virtual environment are mapped in physical link between two or more hosts.
  - **VNF execution**: it is intended as the VNF execution in the considered environment. The VNF is deployed in a physical host. This step involves the VNF execution. Its execution time is linked to host capabilities.

- **RealTimeProperty**: it has been introduced to allow end-user to declare properties that must be satisfied during VerINS module execution. It includes:
  - **Source**: host from which flow has to be started
  - **Destination**: host to which flow has to be ended
  - **Maximum time allowed**: it is expressed in number of timeslots (length of timeslot is an environment property)

In addition, new XML schemas to address Neo4J interaction have been added. Among them, the ones used to map HTTP requests and responses.

To help in build operation, an **ANT script** has been used to generate Java classes automatically through XJC tool (XML to Java Compiler). They are used to make module implementation easier. They are generated starting from an XML schema.

A.2 Resource Description

The main resource handled through APIs consists in the union of elements belonging physical topology and definition of more virtual topologies defined on the physical one. This is the schema that sums up the structure of the main resource.

```
<xsd:element name="NFV">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:long" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="Graphs" minOccurs="1" maxOccurs="1" />
      <xsd:element ref="Constraints" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="PropertyDefinition" minOccurs="1" maxOccurs="1" />
      <xsd:element ref="Hosts" minOccurs="1" maxOccurs="1" />
      <xsd:element ref="Connections" minOccurs="1" maxOccurs="1" />
      <xsd:element ref="FlowElementsResult" minOccurs="0"/>
      <xsd:element name="ResourceID" type="xsd:string" minOccurs="0"
    maxOccurs="1" />
      <xsd:element name="VerInsConstraintsGenerationTimeRequired"
    type="xsd:long" minOccurs="0" maxOccurs="1" />
      <xsd:element name="VerInsZ3TimeRequired" type="xsd:long" minOccurs="0"
    maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>

...

<xsd:element name="NFV"/>
```

Listing A.2. Main resouce XML schema example

The developed web service interacts with NFV resource allowing the end-user to perform various operations on it in order to facilitate his experience of use as regards the interaction with VerIns module. Operations will be described in the next sections. To further simplify the interaction with VerIns module, some operations have been added, allowing CRUD operations (creation, removal, updating, deletion)

Resources	Relative URLs
industrial	/
— environments	/environments
— — {id}	/environments/{id}
— — — — result	/environments/{id}/result
— — — — — — properties	/environments/{id}/properties
— — — — — — — — — {pid}	/environments/{id}/properties/{pid}
— — — — — — connections	/environments/{id}/connections
— — — — — — — — — {cid}	/environments/{id}/connections/{cid}
— — — — — — hosts	/environments/{id}/hosts
— — — — — — — — — {hid}	/environments/{id}/connections/{hid}
— — — — — — graphs	/environments/{id}/graphs
— — — — — — — — — {gid}	/environments/{id}/connections/{gid}

Table A.1. Resources mapping for VerIns module

on the elements that compose the main resource.
Mapping resources for VerIns module (note that base url "/industrial"is omitted):

As can be seen from the table, the main resource is the network topology (understood as the physical topology and the virtual ones declared on it). Within this resource you can find other resources that represent:

- **Result**: results are coming from VerIns module. They are shown to end user in two categories: one is concerning "placement results", and one is concerning "scheduling operations result". The first one category includes the placement of VNF is the physical topology, and in particular, they report where a VNF is deployed. While the second category reports where each element of flow declared through properties (managed with the next resource) is located in terms of timeslot, avoid overlapping between different flows in the network environment.
  Scheduling result must follow this XML schema:

```
1
2 <xsd:element name="FlowElement">
3   <xsd:complexType>
4   <xsd:attribute name="graph" type="xsd:long" />
5   <xsd:attribute name="flow" type="xsd:string" />
6   <xsd:attribute name="type" type="xsd:string" />
7   <xsd:attribute name="description" type="xsd:string" />
8   <xsd:attribute name="startOnTimeslot" type="xsd:int" />
9   </xsd:complexType>
10 </xsd:element>
```

Listing A.3. Flow Element XML schema example

Verins results also include placement suggestion about VNFs orchestration inside studied network environment. Physical host definition contains this other part of results through node reference.

- **Properties**: they are used to declare property that have to be satisfied during placement and scheduling operation done by Verins module. Properties with "RealTimeProperty" as a name are the ones handled through VerIns engine are. You have to specify source and destination of flow. As it is considered as time-sensitive flow, you have to declare max number of timeslots that you are willing to wait until communication is brought to an end.
  Properties must satisfy rule imposed by this XML schema:

```
<xsd:complexType name="Property">
  <xsd:attribute name="name" type="P-Name" use="required" />
  <xsd:attribute name="graph" type="xsd:long" use="required" />
  <xsd:attribute name="src" type="xsd:string" use="required" />
  <xsd:attribute name="dst" type="xsd:string" use="required" />
  <xsd:attribute name="maxNumTimeslots" type="xsd:long" use="optional"
    />
  <xsd:attribute name="resourceID" type="xsd:string" use="optional" />
  <xsd:attribute name="isSat" type="xsd:boolean" />
</xsd:complexType>
```

Listing A.4. Property XML schema example

- **Connections**: they represent the physical interconnections between the various hosts on the network. To each of them is associated with latency and bandwidth, besides the source and destination of involved hosts. The XML schema will be represented in the next section.

- **Hosts**: represents a host on the network, making clear the set of all the features that distinguish it. The XML schema will be represented in the next section.

- **Graphs**: represent the virtual topologies declared on a physical topology. In particular, it contains the definition of all the virtual network functions declared in the environment taken into consideration together with their interconnections in the Service Graph. The XML schema will be represented in the next section.

A.3 Physical and Virtual Topology Management

They represent the environment with which the VerIns module interacts. The physical topology includes the set of all hosts intended as physical machines available and therefore also the set of all connections between them. Both physical devices and connections are characterized by parameters that define them. These parameters are taken into account by the VerIns module during the operations of placement and flow scheduling.
Below is shown how hosts and connections are defined using an XML schema:

```
<xsd:element name="Host">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="SupportedVNF" type="SupportedVNFType"
    maxOccurs="unbounded" minOccurs="0" />
```

```
      <xsd:element name="NodeRef" type="NodeRefType" maxOccurs="unbounded"
    minOccurs="0" />
    </xsd:sequence>

    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="cpu" type="xsd:int" use="required" />
    <xsd:attribute name="cores" type="xsd:int" use="required" />
    <xsd:attribute name="diskStorage" type="xsd:int" use="required" />
    <xsd:attribute name="memory" type="xsd:int" use="required" />
    <xsd:attribute name="maxVNF" type="xsd:int" use="optional" />
    <xsd:attribute name="type" type="TypeOfHost" use="optional" />
    <xsd:attribute name="fixedEndpoint" type="xsd:string" use="optional" />
    <xsd:attribute name="active" type="xsd:boolean" use="optional"
    default="false" />
    <xsd:attribute name="resourceID" type="xsd:string" use="optional" />
  </xsd:complexType>
</xsd:element>
```

Listing A.5. Host XML schema example

```
<xsd:element name="Connection">
  <xsd:complexType>
    <xsd:attribute name="sourceHost" type="xsd:string" use="required" />
    <xsd:attribute name="destHost" type="xsd:string" use="required" />
    <xsd:attribute name="avgLatency" type="xsd:int" />
    <xsd:attribute name="bandwidth" type="xsd:int" use="optional" />
    <xsd:attribute name="resourceID" type="xsd:string" use="optional" />
  </xsd:complexType>
</xsd:element>
```

Listing A.6. Connection XML schema example

Once the physical topology is defined, one or more virtual topologies can be declared on this. These topologies, called Service Graphs in our context, include some nodes. Some of these nodes represent the virtualized network functions. Others instead represent endpoints of the communication taken into account. For instance, in a typical industrial network, endpoints can be a slave that has collected data to be returned to a master collector. While a virtualized network function can be a Modbus firewall.

Below is shown how it is represented in our environment:

```
<xsd:element name="graph">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element ref="node" maxOccurs="unbounded"></xsd:element>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:long" use="optional" />
  <xsd:attribute name="resourceId" type="xsd:string" use="optional" />
  </xsd:complexType>
</xsd:element>
```

Listing A.7. Graph XML schema example

The VerIns module interacts with both types of topologies (physical and virtual) through the Neo4j service. Both topologies are loaded into the Neo4j environment to exploit algorithms made available through this service, such as the minimum

paths research given a topology.
Below is shown how both topologies are represented on the graph-oriented database Neo4j.

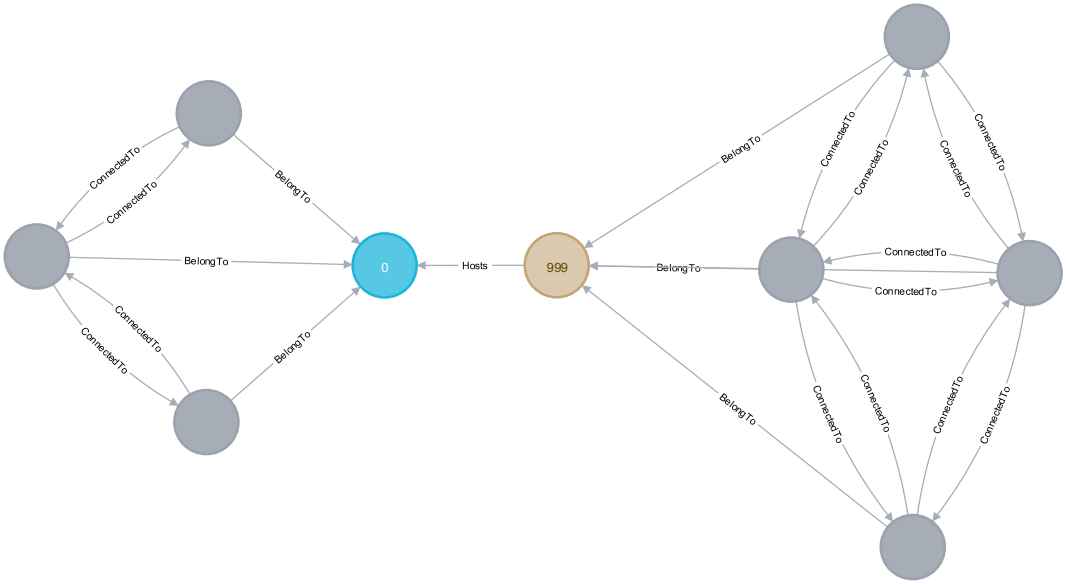

Figure A.1. Virtual and physical graphs mapping

In blue, the Service Graphs (virtual topologies) while in brown, the physical topology hosting the Service Graphs are represented. Neo4j service works separately on these two kinds of topologies to retrieve information about the reachability of two hosts and paths between two or more hosts. Interaction with Neo4j service is possible through available APIs. For this purpose, a client has been developed in a Java environment capable of interfacing with the available APIs. It produces useful results to facilitate the implementation of VerIns module for the network flow scheduling management. Through the client, it is, therefore, possible to load an environment on Neo4j and customize it by creating, modifying, eliminating physical and virtual connections. Here are reported available operations that you could exploit through Neo4j client:

- **addNFVSchema**(NFV nfv, Logger logger)
- **addGraph**(Graph graph, Logger logger)
- **findShortestPathsInNFV**(Long graph, String source, String destination, int maxlength, Neo4jNFVMappingResult mappings, Logger logger)
- **findShortestPathsInPhysicalGraph**(String source, String destination, int maxlength, Neo4jNFVMappingResult mappings, Logger logger)
- **findShortestPathsInGraph**(String source, String destination, int maxlength, Neo4jGraphMappingResult mappings, Logger logger)

- **findAllPathsInNFV**(Long graph, String source, String destination, int maxlength, Neo4jNFVMappingResult mappings, Logger logger)
- **findAllPathsInPhysicalGraph**(String source, String destination, int maxlength, Neo4jNFVMappingResult mappings, Logger logger)
- **findAllPathsInGraph**(String source, String destination, int maxlength, Neo4jGraphMappingResult mappings, Logger logger)
- **getGraph**(Long id, Logger logger, String type)
- **deleteGraph**(Long graphId, Logger logger, String type)
- **getGraphs**(Long id, Logger logger)
- **deleteNFVSchema**(Long nfvId, Logger logger)
- **addNodeToGraph**(String nodeId, Long graphId, Neo4jGraphMappingResult mappings, Logger logger)
- **addConnection**(String source, String destination, Neo4jGraphMappingResult neo4jGraphMappingResult, Logger logger)
- **addOrGetPhysicalGraph**(NFV nfv, Logger logger)

Therefore it is possible to exploit the algorithms already implemented in Neo4j for the search of minimum paths between two nodes in the network.

A.4 Web Service Configuration and Packages

In this section, the design of the web service developed to allow smooth interaction with the VerIns module will be better understood. The main packages are:

- **it.polito.verins.rest.resources**: it contains the definition of all available operation. Jersey-annotated Java Classes that provides mapping between requested URL and method (GET, POST, DELETE, ...) on the one hand and Java method on the other hand;
- **it.polito.verins.rest.db**: it includes all classes that interact with data structure needed for the web service implementation. In particular, have been added classes to manage the main resource of web service and handle concurrent operation on data structure through ConcurrentHashMap provided in Java language;
- **it.polito.verins.rest.webservice**: it contains classes that make more natural interaction between operation requested by user and data structure handled inside it.polito.verins.rest.db package;
- **it.polito.verin.rest.common**: it contains classes that implement the core of VerIns engine.

A.5 Resource Mapping Operations

Resource	Method	Status		Operation
/	POST	201	Created	Upload a new environment
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id	GET	200	OK	Get an existing environment
		404	Not Found	
/id	PUT	200	OK	Update an existing environment
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id	DELETE	200	OK	Delete an existing environment
		404	Not Found	
/id/result	GET	200	OK	Get an existing environment result
		404	Not Found	
/id/connections	GET	200	OK	Get all connections declared in the physical topology
		404	Not Found	
/id/connections	POST	200	OK	Post a new connection in the physical topology
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id/connections/cid	GET	200	OK	Get an existing connection declared in a given environment
		404	Not Found	
/id/connections/cid	PUT	200	OK	Update an existing connection declared in a given environment
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id/connections/cid	DELETE	200	OK	Delete an existing connection declared in a given environment
		404	Not Found	
/id/properties	GET	200	OK	Get all properties declared in a given environment
		404	Not Found	
/id/properties	POST	200	OK	Post a new property in a given enviroment
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id/properties/pid	GET	200	OK	Get an existing property declared in a given environment
		404	Not Found	
/id/properties/pid	PUT	200	OK	Update an existing property declared in a given environment
		400	Bad Request	
		415	Invalid Media Type	
		500	Something wrong with server	
		503	Service temporarily unavailable	
/id/properties/pid	DELETE	200	OK	Delete an existing property declared in a given environment
		404	Not Found	

Table A.2. Resources mapping operations

To be noticed that all POST and PUT requests handle body in APPLICATION/XML content type. Other body types are not supported by web service, returning code 415 Invalid Media Type in response.

A.6 API Interaction Example

- **Upload a new topology environment to web service**: elaborate a post request where in body a topology is described, like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="nfvSchema.xsd">
  <name>999</name>
  <graphs>
    <graph id="0">

    <node functional_type="ENDHOST" name="nodeA">
      <neighbour name="node1" />
      <configuration description="A simple description" name="confA">
        <webclient nameWebServer="nodeB" />
      </configuration>
    </node>

    <node functional_type="ENDHOST" name="nodeB">
      <neighbour name="node1" />
      <configuration description="A simple description" name="confB">

...


```

Listing A.8. Post request body example

URL used to make this POST request is: http://localhost:8080/verifoo/rest/industrial/environments/

- **Get topology result from a declared environment**: elaborate get request to main resource, obtaining result like the following one:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
  <Hosts>
    <Host name="hostA" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeA" active="true" resourceID="1"/>
    <Host name="hostB" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeB" active="true" resourceID="2"/>
    <Host name="hostC" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeC" active="true" resourceID="3"/>
    <Host name="hostD" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeD" active="true" resourceID="4"/>
    <Host name="hostE" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeE" active="true" resourceID="5"/>
    <Host name="hostF" cpu="2" cores="2" diskStorage="10" memory="4"
  type="CLIENT" fixedEndpoint="nodeF" active="true" resourceID="6"/>
    <Host name="host1" cpu="1" cores="4" diskStorage="50" memory="16"
  maxVNF="4" type="MIDDLEBOX" active="true" resourceID="7">
    <SupportedVNF functional_type="FIREWALL"/>
    <SupportedVNF functional_type="CACHE"/>
    <SupportedVNF functional_type="FIELDMODIFIER"/>
    <NodeRef node="node4"/>
    <NodeRef node="node5"/>
    <NodeRef node="node2"/>
```

```
17      <NodeRef node="node3"/>
18      <NodeRef node="node1"/>
19      </Host>
20      <Host name="host2" cpu="1" cores="4" diskStorage="50" memory="16"
      maxVNF="4" type="MIDDLEBOX" active="false" resourceID="8">
21      <SupportedVNF functional_type="FIREWALL"/>
22      <SupportedVNF functional_type="CACHE"/>
23      <SupportedVNF functional_type="FIELDMODIFIER"/>
24      </Host>
25      <Host name="host3" cpu="1" cores="4" diskStorage="50" memory="16"
      maxVNF="4" type="MIDDLEBOX" active="true" resourceID="9">
26      <SupportedVNF functional_type="FIREWALL"/>
27      <SupportedVNF functional_type="CACHE"/>
28      <SupportedVNF functional_type="FIELDMODIFIER"/>
29      <NodeRef node="node6"/>
30      </Host>
31    </Hosts>
32    <FlowElements>
33      <FlowElement graph="0" flow="from_nodeA_to_nodeD"
      type="connection" description="from hostA to host1"
      startOnTimeslot="0"/>
34      <FlowElement graph="0" flow="from_nodeA_to_nodeD"
      type="vnfExecution" description="node1" startOnTimeslot="1"/>
35      <FlowElement graph="0" flow="from_nodeA_to_nodeD"
      type="vnfExecution" description="node2" startOnTimeslot="2"/>
36      <FlowElement graph="0" flow="from_nodeA_to_nodeD"
      type="vnfExecution" description="node4" startOnTimeslot="3"/>
37      <FlowElement graph="0" flow="from_nodeA_to_nodeD"
      type="connection" description="from host1 to hostD"
      startOnTimeslot="4"/>
38    </FlowElements>
39    <VerInsConstraintsGenerationTimeRequired> 249
      </VerInsConstraintsGenerationTimeRequired>
40    <VerInsZ3TimeRequired>20</VerInsZ3TimeRequired>
41  </NFV>
42
```

Listing A.9. Get request to main resource body example

URL used to make this GET request is: http://localhost:8080/verifoo/rest/industrial/environments/1/result Results contains both placement and flow scheduling results coming from VerINS module.

Bibliography

[1] Trend Micro. *Industrial Control System*. Available on line. URL: https://www.trendmicro.com/vinfo/us/security/definition/industrial-control-system.

[2] HEXA Engineers. *The Differences Between DCS and SCADA*. Available on line. 2019. URL: https://www.hexaengineers.us/the-differences-between-dcs-and-scada/.

[3] Luís Almeida. «Flexibility and Timeliness in Fieldbus-based Real-time Systems». In: (Oct. 2019).

[4] Manuel Cheminod et al. «Leveraging SDN to improve security in industrial networks». In: *WFCS*. IEEE, 2017, pp. 1–7.

[5] Antonio López Padilla Miguel Herrero Collantes. *Protocols and network security in ICS infrastructure*. Spanish National Institute for Cyber-security, 2015.

[6] Bruno Dutertre. «Formal Modeling and Analysis of the Modbus Protocol». In: *Critical Infrastructure Protection*. Ed. by Eric Goetz and Sujeet Shenoi. Boston, MA: Springer US, 2008, pp. 189–204. ISBN: 978-0-387-75462-8.

[7] Peter Neumann and Axel Pöschmann. «Ethernet-based real-time communications with PROFINET IO». In: 4 (May 2005).

[8] T. Stock and G. Seliger. «Opportunities of Sustainable Manufacturing in Industry 4.0». In: *Procedia CIRP* 40 (2016). 13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use, pp. 536 –541. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2016.01.129. URL: http://www.sciencedirect.com/science/article/pii/S221282711600144X.

[9] Alasdair Gilchrist. *Industry 4.0: The Industrial Internet of Things*. Apress, 2016. ISBN: 978-1-4842-2046-7.

[10] C. Moeller J. Smit S. Kreutzer and M. Carlberg. *Policy department A: economic and scientific policy*. Parliament's Committee on Industry, Research and Energy (ITRE). 2016.

[11] J. Matias et al. «Toward an SDN-enabled NFV architecture». In: *IEEE Communications Magazine* 53.4 (Apr. 2015), pp. 187–193. DOI: 10.1109/MCOM.2015.7081093.

[12] *Architectural Framework*. ETSI Network Function Virtualisation (NFV) Industry Specification Group (ISG). 2013.

[13] L. Zhou and H. Guo. «Applying NFV/SDN in mitigating DDoS attacks». In: *TENCON 2017 - 2017 IEEE Region 10 Conference*. Nov. 2017, pp. 2061–2066. DOI: 10.1109/TENCON.2017.8228200.

[14] Guido Marchetto et al. «Formally verified latency-aware VNF placement in industrial Internet of things». In: *14th IEEE International Workshop on Factory Communication Systems, WFCS 2018, Imperia, Italy, June 13-15, 2018*. IEEE, 2018, pp. 1–9. ISBN: 978-1-5386-1066-4. DOI: 10.1109/WFCS.2018.8402355. URL: https://doi.org/10.1109/WFCS.2018.8402355.

[15] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. «Time-sensitive Software-defined Network (TSSDN) for Real-time Applications». In: *RTNS*. ACM, 2016, pp. 193–202.

[16] Andrew Loveless. «On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks». In: Aug. 2015. DOI: 10.2514/6.2015-4526.

[17] K. Harris. «An application of IEEE 1588 to Industrial Automation». In: *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. Sept. 2008, pp. 71–76. DOI: 10.1109/ISPCS.2008.4659216.

[18] John Swindall Hunter Harrington Mirko Torrez Contreras. *Isochronous Real-Time (IRT) Communication*. URL: https://profinetuniversity.com/profinet-basics/isochronous-real-time-irt-communication/.

[19] *Network Functions Virtualisation (NFV) Release 3; NFV Evolution and Ecosystem; Hardware Interoperability Requirements Specification*. ETSI Network Function Virtualisation (NFV) Industry Specification Group (ISG). 2017.

[20] P. Iovanna et al. «SDN-based architecture to support Synchronization in a 5G framework». In: *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. Sept. 2016, pp. 1–4. DOI: 10.1109/ISPCS.2016.7579504.

[21] Microsoft Research. *Z3 - guide*. Available on line. URL: https://rise4fun.com/z3/tutorial.

[22] David R. Cok. «The SMT-LIBv2 Language and Tools: A Tutorial». In: 2012.

[23] Inc. Neo Technology. *Neo4j Graph Platform*. Available on line. URL: https://neo4j.com.

[24] Eclipse Foundation. *Jersey*. URL: https://eclipse-ee4j.github.io/jersey/.