QoS and Resource-aware Security Orchestration and Life Cycle Management

Miloud Bagaa *, Tarik Taleb *¶‖, Jorge Bernal Bernabe **, Antonio Skarmeta **
* Aalto University, Finland, {miloud.bagaa, tarik.taleb}@aalto.fi
¶University of Oulu, 90570 Oulu, Finland.
‖ Department of Computer and Information Security, Sejong University, South Korea.
** Department of Communications and Information Engineering, University of Murcia, Spain {jorgebernal, skarmeta}@um.es

***Abstract*—Zero-touch network and Service Management (ZSM) exploits Network Function Virtualization (NFV) and Software-Defined Networking (SDN) to efficiently and dynamically orchestrate different Service Function Chaining (SFC), whereby reducing capital expenditure and operation expenses. The SFC is an optimization problem that shall consider different constraints, such as Quality of Service (QoS), and actual resources, to achieve cost-efficient scheduling and allocation of the service functions. However, the large-scale, complexity and security issues brought by virtualized IoT networks, which embrace different network segments, e.g. Fog, Edge, Core, Cloud, that can also exploit proximity (computation offloading of virtualized IoT functions to the Edge), imposes new challenges for ZSM orchestrators intended to optimize the SFC, thereby achieving seamless user-experience, minimal end-to-end delay at a minimal cost. To cope with these challenges, this paper proposes a cost-efficient optimized orchestration system that addresses the whole life-cycle management of different SFCs, that considers QoS (including end-to-end delay, bandwidth, jitters), actual capacities of Virtual Network Functions (VNFs), potentially deployed across multiple Clouds-Edges, in terms of resources (CPU, RAM, storage) and current network security levels to ensure trusted deployments. The proposed orchestration system has been implemented and evaluated in the scope of H2020 Anastacia EU project[1], showing its feasibility and performance to efficiently manage SFC, optimizing deployment costs, reducing overall end-to-end delay and optimizing VNF instances distribution.**

I. Introduction

Network Function Virtualization (NFV) and Software-Defined Networking (SDN) paradigms will play crucial roles in enabling the 5G system and beyond. While NFV decouples the network function from the hardware to decrease the capital expenditures (CAPEX) and operating expenses (OPEX), SDN enables the network softwarization, and hence it elevates the network flexibility. NFV exploits virtualization techniques to deploy network functions (e.g. router, switches, proxies), and security network functions (e.g. firewalls, proxies, AAA services, Virtual-HoneyNet) on Virtual Machines that run top of general-purpose hardware, separating network functionality from traditional dedicated hardware middle-boxes. From another side, Zero-touch network and Service Management (ZSM) [1] will play a crucial role in enabling closed-loop efficient network slice management systems. The main aim of ZSM is to allow zero-touch automated network and service management in a multi-vendor and multi-tenant environment, including management of programmable, multi-layer and cloud-native networks, as well as traditional virtualized and physical network functions. ZSM leverages cloud-native principles, SDN and NFV for addressing (fully automated) management and operation of network slicing. ZSM allows automatizing all the service chaining processes, thereby minimizing the required provisioning time, optimizing operational and capital costs, and reducing or even eliminating human intervention. These virtual network service functions (VNFs) can be dynamically and effectively connected setting-up different service function chains (SFCs) belonging to different service providers (SPs), that run in the same hardware sharing the available physical resources.

For enabling network slicing, VNFs are leveraged that use either virtual machines (VMs) or light-wight virtualization (Containers) that should run on top of different administrative and technological clouds. The SFC provisioning is a challenging problem that requires placing VNFs in different clouds, interconnect them in proper order in the chain, and then steering the traffic. ZSM requires all the portions, SDN, NFV and slice orchestration, are integrated in a unified manner to ensure the closed loop configurations and orchestration of different network slices that are presented by different SFCs. The orchestration and the management of SFCs are an optimization problem that looks for finding a fair trade-off between the cost and expected QoS offered by each SFC [2]. The optimization problem is even more complicated due to the sharing of various physical resources across SPs. The scheduling and the chaining of VNFs under different administrative domains (i.e., SPs) make the orchestration further difficult. In addition, the optimization problem is exacerbated when dealing with SDN/NFV-based IoT networks, where different virtualized and softwarized edges, fog nodes, and multi-clouds, able to allocate VNFs, and having each one different available flavors, QoS and resources available are involved. This heterogeneity imposes challenges to manage and optimize the end-to-end overall delay as different heterogeneous virtual network infrastructures in the SFC are involved.

To deal with the above challenges, in this paper, we suggest a cost efficient orchestration mechanism that enables ZSM of different SFCs. We have suggested a novel orchestration

[1]H2020 Anastacia EU project website: http://www.anastacia-h2020.eu/

mechanism aimed to optimize the allocation and life cycle management (LCM) of VNFs potentially deployed across IoT domains, inter-clouds, and cross network segments, thereby ensuring efficient management of SFC in virtualized and softwarized IoT networks. We have suggested an optimal and a heuristic solutions for ensuring an efficient LCM. The optimal solution leverages linear integer programming (LIP) for finding optimal configurations. In contrast, the heuristic provides near optimal configuration, however within a short execution time. Unlike the current state of the art, the proposed solutions consider not only quality of service (QoS) of the system (including end-to-end delay, bandwidth, jitters) and actual capacities of VNFs in terms of Resources (CPU, RAM and disk), but also the network security levels (enforced security channels mechanisms across the VNFs in the SFC). Moreover, the suggested solutions provide cost efficient ZSM for orchestrating and scheduling different network services. The algorithms consider a predefined blueprint configuration for the SFC, and targeted KPIs, and optimize the targeted KPIs in terms of link security level, QoS, delay and bandwidth. Thus, depending on the KPIs defined in the SFC blueprint, the VNFs and communication links are used to satisfy the predefined requirements. To the best of our knowledge this is the first proposal intended to deal with ZSM for providing cost efficient network service orchestration mechanism for SDN/NFV-based IoT networks that might demand multiple clouds, fog nodes, virtualized edges to host the VNF instances conforming the SFC. The proposed orchestration solutions have been successfully implemented and evaluated showing their feasibility and performance to deal with life-cycle management of SFC in IoT scenarios [3].

The rest of this paper is structured as follows. Section II presents related works. Section III overviews the orchestration architecture for NFV/SDN-enabled IoT networks. Section IV formulates the optimization problem. The optimization algorithm is described in section V. The heuristic solution is described in section VI. Section VII presents the evaluation and finally section VIII concludes the paper.

II. Related work

The NFV orchestration has been widely discussed in the literature to efficiently and autonomously manage and control software and hardware in NFV Infrastructure (NFVI). To efficiently manage and orchestrate the NFVI, a Management Orchestration (MANO) layer has been proposed that is responsible for managing and orchestrating both VNFs and NFVI. To ensure an efficient orchestration plan, MANO architecture has mainly three functional components. The first component is NFV orchestrator (NFVO) that is responsible for: i) Network resource orchestration of NFVI and VNFs that form SFCs; ii) Validating and authorizing NFVI resource requests from the VNF manager (VNFM); iii) Network service life-cycle management of SFCs and their VNFs. The second component is VNFM, which is responsible for the life-cycle management of VNFs at a specified cloud. Finally, the Virtualized Infrastructure Manager (VIM) manages NFVI associated that usually belongs to the cloud operator. The VIM is also responsible for managing Virtual Machine (VMs) that host different VNFs.

As it has been recently highlighted in different surveys on network service orchestration [4], [5], one of the main open challenges is to consider security aspects when it comes to SFC optimization and VNF management. NFVO is responsible for the dynamic life cycle management to provide end-to-end connectivity and additional services between users and accessed services. The SFC optimization problem has been widely studied recently. The basic formulation for VNF scheduling considering VNFs' processing delays can be found in [6]. Similarly, in [7], authors laid out simple algorithms aimed at function mapping and scheduling (NFMS) of VNFs, considering as objective problems the minimization of flow time, cost, and revenue. However, these works do not consider end-to-end delays in multi-domain scenarios, in which interconnection links can influence delays in the SFC.

In [8] authors analyze the problem of VNFs allocation optimization with optimal algorithms, but they do not actually deal with service chaining and order of VNFs. Likewise, in [9] authors address the service chaining optimization challenge, but they do not consider global delays of network traffic in the SFC path. Similarly, in [10] authors formalize the VNF placement and SFC problem and follow an Integer Linear Programming (ILP) to deal with the optimization problem, considering the distance across clients and VNFs and their cost. However, they do not consider other resources and security aspects as addressed in our model. In contrast to this solution, our model considers cost-efficient life cycle management when allocating computational and network resources. Some other related research papers focus on other kind of QoS metrics for performance improvement, such as bandwidth allocation optimization for VNFs usage [11]. In [12] authors analyze the VNF scheduling and resource allocation with service chain, and provide resource optimization solutions. They define the scheduling decisions and chaining based on service transmission and VNF's processing delays. However, unlike our work they do not consider security aspects in the formulation.

A reliability-aware SFC provisioning of VNFs with delay guarantees is proposed in [13]. They suggest a mixed integer linear program (MILP) for VNFs allocation that maximizes reliability and end-to-end delays. Artificial Intelligence is starting to be exploited for VNF scheduling [14]. However, in their work authors only account on transmission and processing delays to optimize the SFC. Recently Long Qu et al. [15] have proposed a model for SFC orchestration in 5G networks. They formulate the optimization problem using ILP and heuristic algorithms, but considering only the minimization of the path length and cost of computing resources of the VMs.

None of the above research works have considered so far the security aspects, such as security levels in different links interconnecting IoT domains with different clouds. Also, in contrast to these works, the proposed framework considers cost-efficient ZSM of different network slices across heterogeneous technological and administrative clouds domains. The suggested solutions consider the optimization of the SFC of VNFs combining inter-cloud relationships, QoS properties,

and resource features.

III. Security Orchestration in SDN/NFV-based IoT systems

Our proposed Service Function Chaining (SFC) algorithms and life Cycle Management model for SDN/NFV-aware IoT systems (defined in section V) have been implemented as a security Orchestrator component, which in turn, has been integrated in a global Security Architecture [16] for IoT networks devised in the scope of ANASTACIA H2020 EU project. The Anastacia architecture implements a MAPE loop (Monitor, Analyze, Planning, Execute) to counter dynamically cyber-attacks in IoT systems, by deploying dynamically the pertinent chained virtual network security functions according to the actual network and system conditions. The Security Orchestrator is able to interact with different controllers to manage the systems (Fig. 1(a)) by ensuring cost-efficient life cycle management of SFC and ensuring the network security. Namely, the SO interfaces with the IoT controller, SDN controller and NFV orchestrator. When an attack is detected, thanks to the mitigation action service (MAS) component, a mitigation request (security policy) is sent to our security orchestrator (SO) (Fig. 1(b), Step 3) that runs the optimization algorithms for selecting the best SFCs, according to the actual resources, QoS and security aspects.

The SO oversees orchestrating the virtual security enablers according to the security policies generated and forwarded from other architectural components. In order to mitigate the various attacks without affecting the QoS in different verticals, the SO takes into account the policies requirements and the available resources in the underlying infrastructure. The latter refers to the available amount of resources in terms of CPU, RAM, and storage in different cloud providers, as well as the communication network resources including the end-to-end bandwidth, jitters and delay. The quality of the link varies according to the locations of different communicating peers (i.e., intra and inter cloud communication), as well as the use of secure channels with different levels including IPsec, SSL and TLS. As aforementioned, the SO is able to interface with the IoT controller, SDN controller and NFV controller.

- **IoT controller:** It provides IoT command and control at high-level of abstraction in independent way of the underlying technologies. That is, it is able to carry out the IoT management request through different IoT constraint protocols like CoAP or MQTT. It also maintains a registry of relevant information of the deployed IoT devices like the IoT device properties and available operations. The SO interacts with the IoT controller to mitigate the attacks at the level of the IoT domain and prevent the propagation of the attack to other networks (Figure 1(b): 4). The IoT controller enforces different security rules at the IoT router (data plane) to mitigate the attack (Figure 1(b): 5).
- **NFV orchestrator:** to ensure efficient management of SFC, we have integrated SDN controller (ONOS) with the used Virtual Infrastructure Manager (VIM), in our case OpenStack. The integration of SDN with the VIM enables the smooth communication between different VNFs that form the same SFC. After receiving the MSPL message from the MAS, the security orchestrator identifies the right mitigation plane should be implemented. If the mitigation plan requires the instantiation of new VNFs, the security orchestrator (SO) instructs the NFV orchestrator to instantiate and configure the required VNFs. To instantiate the required VNFs, the NFV orchestrator interacts with the VIM (Figure 1(b): 6). After the successful instantiation of a security VNF, the security orchestrator configures that VNF with the security policy (Figure 1(b): 6).
- **SDN controller:** This component helps in rerouting the traffic between the VNFs in different SFCs. As depicted in Fig. 1, when the mitigation action service notifies the orchestrator about an attack, the SFC would be updated by adding/inserting new security VNFs in the SFCs. The security orchestrator should push the adequate SDN rules to reroute the traffic between different VNFs in the SFC and the IoT domain (Fig. 1(b): 7). Also, according to the different situations, the security orchestrator can choose the SDN as security enabler. In this case, it can be the attack mitigated by pushing exploring the strength of the SDN technology. If so, the security orchestrator can instruct the SDN controller to push some SDN rules to prevent, allow or limit the communication on specified protocols and ports between different communication peers (Fig. 1(b): 7).

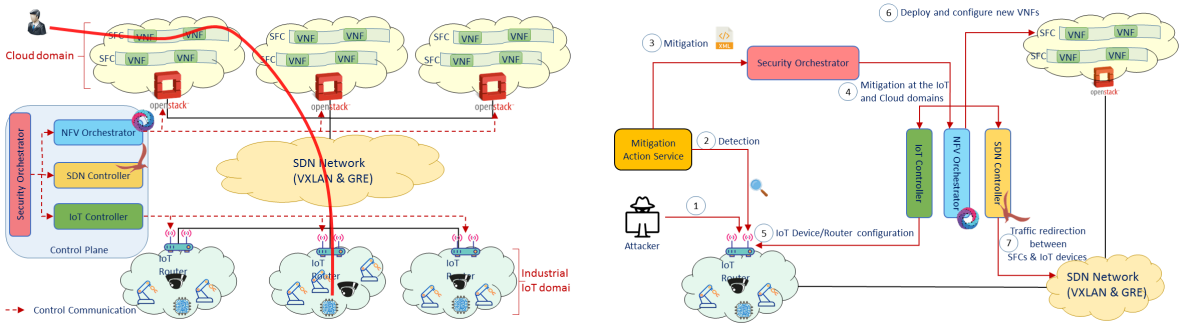
Formally, in our solution, the communication between a user and an IoT domain happens through a list of chains of VNFs and PNFs named service function chaining (SFCs). The latter consists of three parts:

- The ingress point, which is the first VNF in the SFC. The user initially attaches to the ingress point;
- The intermediate VNFs;
- the egress point, which is the last VNF in the SFC. The egress point should be connected to the IoT controller.

As depicted in Fig. 1(b), the order of the communications between the VNFs is defined according to the different SDN rules enforced thanks to the SDN controller. The nature and the size of the SFCs would be defined according to the nature of the user (a normal or a suspicious).

Thus, the framework can mitigate the threats either at the PNFs, VNFs or IoT devices by directly interacting at these components and/or the network interconnecting them. The communication between different peers (End-user and IoT devices) happens thanks to VNFs and PNFs installed in the network. When the anomalies are detected, an alert should be generated and forwarded to the SO. The latter mitigates the attacks by dint of SE plane by:

- Pushing different rules at the network by leveraging software defined networking (SDN);
- Interacting at the IoT devices and PNFs by pushing different rules that mitigate the attacks, such as shutting off the malicious IoT devices;
- Creating different VNFs (e.g., virtual firewall) and enforce the traffic to pass through these VNFs, such that



(a) Network topology

(b) Three UEs are interested to connect to the server

Figure 1: Orchestration system functional architecture

the exploit of different vulnerabilities is mitigated through ports communication restriction.

By relying in the aforementioned orchestration properties and features, as well as NFV, SDN and IoT controllers, the ANASTACIA framework aims to cope with the research challenges related with Orchestration of SDN/NFV-based security solutions for IoT environments and currently several experiments have been carried out in different security areas. For instance, we have developed a new VNF firewall based on SDN-enabled switch and OpenFlow [17], or a VNF for AAA and channel protection solutions [18]. Besides, several experiments have been successfully carried out regarding virtual IoT-honeynets [19]. This kind of VNF allows replicating a real IoT environment in a virtual one by simulating the IoT devices with their real deployed firmware, as well as the physical location. The IoT-honeynet can be represented by an IoT-honeynet security policy, and the final configuration can be deployed transparently on demand with the support of the SDN network.

The following sections focus on defining the Orchestration Component of Anastacia Framework, including the SFC model and life cycle management of the VNFs.

IV. Network Model and Problem Formulation

A. Main idea

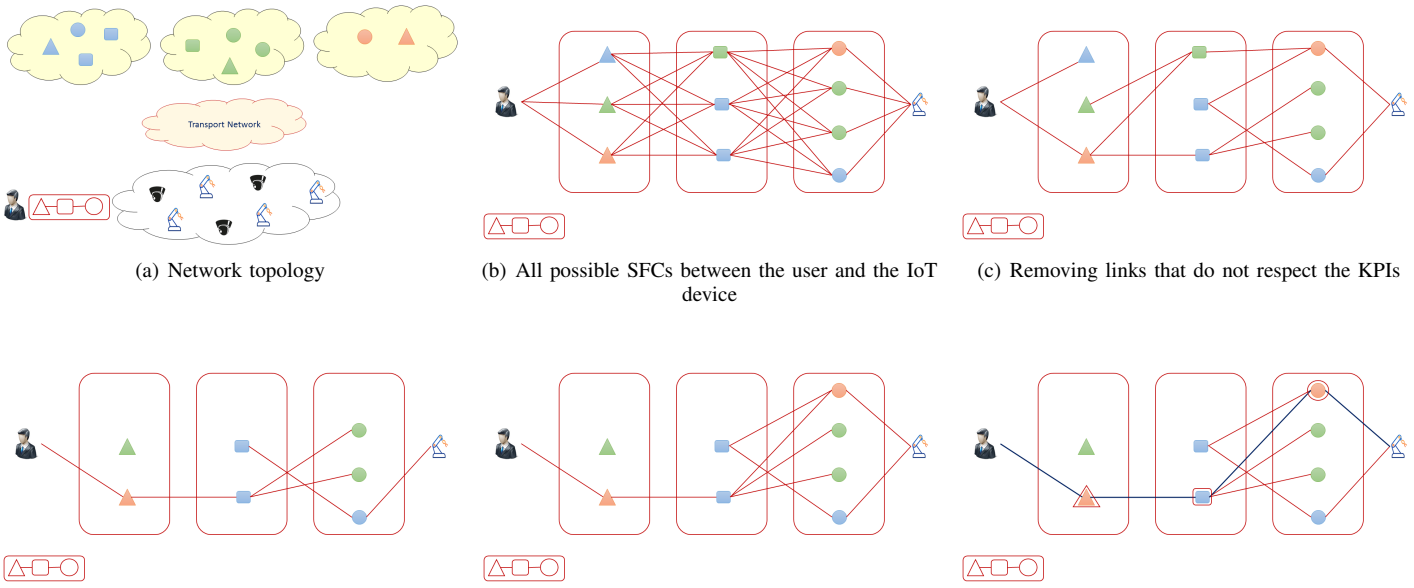
In this paper, we are interested to provide a framework that serves a user $u \in \mathcal{U}$ to access to an IoT devices $d \in \mathcal{D}$ deployed in a distributed IoT infrastructure by leveraging SDN and NFV technology as depicted Fig. $2(a)$ for enabling Zero-touch network Service Management (ZSM) of different network slices. The user u would access to the IoT device d through a specified SFC that consists of a set of VNFs and PNFs, respectively. For instance, the user u can access to the IoT devices using an SFC that consists of: i) A load balancer (a triangle in Fig. $2(a)$); ii) A firewall that is presented as rectangle in Fig. $2(a)$; iii) Finally, an MQTT broker whereby the IoT device publish its message.

The communication, between different clouds, VNFs, users and devices, is characterized by different delay, bandwidth, jitters and security levels. For instance, the communication link within the same cloud has a higher security level while the one between users, devices and inter clouds that do not use security mechanisms, such as IPSEC and SSL, has the lowest security level. Each SFC is created from a predefined blueprint that specifies its characteristics and targeted KPIs that should be respected in terms of link security level, QoS, delay and bandwidth. According to the KPIs defined in the SFC blueprint, the VNFs and communication links should be used to satisfy the predefined requirements. Fig. $2(b)$ depicts a rearrangement of candidate VNFs, in different clouds, and their possible links between them to fulfil the requested SFC.

As we have explained before, some links could not satisfy the KPIs specified in the SFC blueprint, and hence they should be considered when instantiating the SFC as shown in Fig. $2(c)$. As depicted in Fig. 1 of [20], the information about the links, such as bandwidth, jitters and delay, and the resources of already deployed VNFs in terms of CPU, RAM and disk would be retrieved from the module "Resource and QoS Monitoring". The latter would give predict values about different metrics by exploring the "Data Analytics" module. Using the latter module, we can predict the resources should be consumed from each VNFs if it is included in the SFC. Thus, each VNFs that would use resources higher than a predefined threshold should be not considered as depicted in Fig. $2(d)$. However, we could have a situation whereby some VNFs are missing to complete the SFC as depicted in Fig. $2(d)$. In this case, the security orchestrator needs to instantiate a new VNF with the required resources and at the appropriate cloud (i.e., Links that ensure the KPIs) as depicted in Fig. $2(e)$. Then, the SFC would be instantiated using both NFV orchestrator and SDN controller as depicted in Fig. $2(f)$. Note that in a real scenario, we can receive many requests to create different SFCs with different size and KPIs. The security optimizer component should consider all the requests once to prevent to generate sub-optimal configuration (i.e., local optimal situation).

B. Problem formulation

We denote by $\mathcal{U}$ a set of users that interested with different services offered by IoT devices $\mathcal{D}$ deployed in a distributed IoT infrastructure. For the sake of clarity, a summary of the



(a) Network topology

(b) All possible SFCs between the user and the IoT device

(c) Removing links that do not respect the KPIs

(d) Removing VNFs that do not have enough resources

(e) Creating a new VFN at the third cloud to generate a valid SFC

(f) Generating the final SFC between the user and the IoT device

Figure 2: Main idea of the proposed framework

notations employed in this formulation is included in Table I. Let $\mathcal{G}(V, E, W)$ denote the underlying network that consists of a set of IoT devices and clouds. V consists of: i) Clouds or edges (V_C); ii) Distributed IoT infrastructure (V_I); iii) End-users (V_U) that represent the (radio) access nodes ((R)AN) from whereby the users access. Each cloud/edge $u \in V_C$ is characterized by a limited storage and computation resources including CPU, RAM and storage. Let $\Delta_r(u)$ be a vector that shows the resources of the cloud $u \in V_C$ in terms of resources r that could be CPU, RAM and storage, respectively. For the sake of simplicity, E represents the end-to-end connections between $V_U \cup V_I \cup V_C$ by considering an overlay network and by making an abstraction on the intermediate routers and switches (backbone). However, the suggested model is orthogonal, and can be easily adapted and extended to consider the intermediate network components if needed. Meanwhile, W denotes the characteristics of the links that include the bandwidth, delay and security level. Let $\mathcal{W}^{\mathcal{B}}$, $\mathcal{W}^{\mathcal{L}}$ and $\mathcal{W}^{\mathcal{S}}$ denote the characteristics of the links $\mathcal{E}$ in terms of bandwidth capacity, propagation delay and security level, respectively.

We denote by Θ the set of SFCs in the network, such that each SFC $\theta \in \Theta$ serves a set of users. In this paper, we propose a "Security Orchestrator Optimizer" (SOO) component that ensures the life cycle management of SFCs Θ. Formally, at a given time, we have three set of SFCs: i) SFCs $\dot{\Theta}$ that will not be used in the future as all its users left the communication or shifted to new SFCs; ii) SFCs $\ddot{\Theta}$ that already deployed and would be kept as its users should be stay connected in the future; iii) New SFCs $\hat{\Theta}$ that would be instantiated to serve new connected users. While the first SFCs $\dot{\Theta}$ will not be considered in the optimization, the other SFCs are considered as their users will be connected in the future. Thus, the resource used by $\dot{\Theta}$ should be released before executing the optimization. Let $\bar{\Theta}$ denote the SFCs that would be considered in the future to interconnect the new and existing users to their IoT devices. Formally, $\bar{\Theta} = \ddot{\Theta} \cup \hat{\Theta}$. In the proposed model, we aim to provide ZSM for enabling cost-efficient LCM of network slicing that consists of Θ. For this reason, either periodically or when the KPIs are not respected, the "Security Orchestrator Engine" (SOE) will call the SOO for delivering the optimal configurations. For each SFC $\theta \in \bar{\Theta}$, we have the QoS and security level should be respected. Let $\xi_\theta^{\mathcal{L}}$ and $\xi_\theta^{\mathcal{B}}$ denote the end-to-end delay and bandwidth should be respected, respectively. Meanwhile, $\xi_\theta^{\mathcal{S}}$ represents the minimum security level that should be considered when interconnecting different VNFs of the SFC θ. The SOE detects the violation in the KPIs thanks to the "Resource and QoS Monitoring" (RQM) or Mitigation Action Services (MAS) components. While RQM detects the violation in the KPIs that have relationship with the network and computation resources utilization, the MAS detects the violation in the security that has relationship with the security. Let Φ_θ denote the set of users of the SFC $\theta \in \bar{\Theta}$. The SOO would have the information about the expected traffic that would be exchanged between the users Φ_θ and the IoT devices. Let λ_θ denote the expected traffic that will be exchanged between the users Φ_θ and the IoT devices.

In the proposed model, we leverage the strength of SDN technology to interconnect the users with their IoT devices. Formally, the communication between a user and its IoT devices happens through a specified SFC $\theta \in \bar{\Theta}$ that consists of a list of VNFs Υ_θ. As we have mentioned in the previous section, each VNF has a specified type, such as firewall, load balancer,...etc. We denote by Π the set of VNFs types

Table I: Summary of Notations.

Notation	Description
$\mathcal{U}$	The set of users in the network.
$\mathcal{D}$	The IoT devices deployed in a distributed IoT infrastructure.
V_C	The set of clouds or edges existing in the network.
$\mathcal{R}$	The set of all the resources. $\mathcal{R}$ could equal to $\{\text{RAM}, \text{CPU}, \text{DISK}\}$.
$\Delta_r(c)$	A parameter that shows the amount of resource $r \in \mathcal{R}$ available at the cloud/edge $c \in V_C$.
$\mathcal{W}^{\mathcal{B}}$	The characteristics of the links in terms of bandwidth.
$\mathcal{W}^{\mathcal{L}}$	The characteristics of the links in terms of delay.
$\mathcal{W}^{\mathcal{S}}$	The characteristics of the links in terms of security.
Θ	The set of service function chaining (SFCs) that serves the users.
$\dot{\Theta}$	The SFCs that will not be used in the future.
$\ddot{\Theta}$	The already deployed SFCs and would be kept in the future.
$\hat{\Theta}$	The set of new SFCs that should be deployed in the future for serving the users.
$\bar{\Theta}$	The set of SFCs that would be considered in the future. Formally, $\bar{\Theta}$ is defined as follows: $\bar{\Theta} = \ddot{\Theta} \cup \hat{\Theta}$
Φ_θ	The users of the SFC $\theta \in \bar{\Theta}$.
λ_θ	The expected traffic that will be exchanged between the users Φ_θ and the IoT devices $\mathcal{D}$.
Υ_θ	The list of VNFs forming the SFC $\theta \in \bar{\Theta}$.
Π	The set of VNFs' types supported in the system, such as firewall, load balancer,...etc.
π_υ	The type of the VNF $\upsilon \in \Upsilon_\theta$, such that $\pi_\upsilon \in \Pi$.
$\dot{\mathcal{V}}$	The already deployed VNF instances that serve already existing VNFs $\bigcup_{\theta \in \ddot{\Theta}} \Upsilon_\theta$.
$\hat{\mathcal{V}}$	The new VNF instances that should be instantiated to serve the new VNFs $\bigcup_{\theta \in \hat{\Theta}} \Upsilon_\theta$.
$\mathcal{V}$	The VNF instances (VNFI) that would be used by the VNFs $\bar{\Theta}$ in the future. Formally $\mathcal{V}$ is defined as follows: $\mathcal{V} = \ddot{\mathcal{V}} \cup \hat{\mathcal{V}}$
$\zeta(\upsilon)$	All the VNFs that have the same type like the VNF $\upsilon \in \bigcup_{\theta \in \bar{\Theta}} \Upsilon_\theta$. $\zeta(\upsilon)$ is defined formally as follow: $\zeta(\upsilon) = \{\acute{\upsilon} : \forall \acute{\upsilon} \in \bigcup_{\acute{\theta} \in \bar{\Theta}} \Upsilon_{\acute{\theta}} \wedge \pi_\theta = \pi_{\acute{\theta}}\}$.
$\Psi(\upsilon)$	All the VNFs that have conflicts with the VNF $\upsilon \in \Upsilon_\theta$.
$\mathcal{F}$	The list of flavors that can be used by different VNFs. Each flavor $f \in \mathcal{F}$ specifies the amount of all resources $r \in \mathcal{R}$ that should be used by a VNF instance.
$\delta_r(f)$	The amount of resources $r \in \mathcal{R}$ used by the flavor $f \in \mathcal{F}$.
$\delta_p(f)$	The price of using the flavor $f \in \mathcal{F}$.
$\Omega(u)$	The set of VNFs that would be hosted at the VNFI $u \in \mathcal{V}$.
$\Gamma(.)$	$\Gamma(\pi_u, f, \sum_{\theta \in \bar{\Theta}, \upsilon \in \Upsilon_\theta \cap \Omega(\upsilon)} \lambda_\theta)$ denotes the average processing delay would be expected from the VNFI u to process one packet.
$\xi_\theta^{\mathcal{L}}$	It denotes the end-to-end delay requirement of the SFC θ that should be respected.
$\xi_\theta^{\mathcal{B}}$	It denotes the end-to-end bandwidth of the SFC θ that should be respected.
$\xi_\theta^{\mathcal{S}}$	It denotes the end-to-end security level of the SFC θ that should be respected.

supported in the system. We denote by π_υ the type of the VNF $\upsilon \in \Upsilon_\theta$, such that $\pi_\upsilon \in \Pi$. Each SFC θ consists of three parts:

- The ingress point ($\upsilon_I \in \Upsilon_\theta$), which is the first VNF in Υ_θ. The user initially attaches to (R)AN, and then from the (R)AN to the ingress point (υ_I);
- The egress point ($\upsilon_E \in \Upsilon_\theta$), which is the last VNF in Υ_θ;
- The intermediate VNFs ($\upsilon \in \Upsilon_\theta \setminus \{\upsilon_I, \upsilon_E\}$);

We denote by $(\upsilon_i, \upsilon_j) \in \Upsilon_\theta$ a two consecutive VNFs in the SFC $\theta \in \bar{\Theta}$. Note that the SFCs $\bar{\Theta}$ are only logical presentation that should be specified in the blueprint as depicted in Fig. 2 with white colored shapes. $\bar{\Theta}$ would be instantiated in the clouds as shown in the same figure as colored shape. In the blueprint, the KPIs of each SFC $\theta \in \bar{\Theta}$ are also defined in terms of end-to-end delay and bandwidth, as well as the security levels of the links used for interconnecting θ's components. While the two former metrics target the quality of experience (QoE), the latter targets the security level of θ. Let $\sigma_\theta^{\mathcal{L}}$, $\sigma_\theta^{\mathcal{B}}$ and $\sigma_\theta^{\mathcal{S}}$, denote the θ's end-to-end delay, bandwidth and security level, respectively.

Let $\mathcal{V}$ represent the instantiated VNFs in different clouds (i.e, colored shapes in Fig. 2). $\mathcal{V}$ represents the VNF instances (VNFI) that would be used by the VNFs of SFCs $\bar{\Theta}$. In the network, we would have two types of VNFIs, $\mathcal{V} = \ddot{\mathcal{V}} \cup \hat{\mathcal{V}}$. Already deployed VNFIs $\ddot{\mathcal{V}}$ that serve already existing VNFs $\bigcup_{\theta \in \ddot{\Theta}} \Upsilon_\theta$. Meanwhile, the Algorithms will also instantiate new VNFIs for the new SFCs $\hat{\Theta}$. Unfortunately, $\ddot{\mathcal{V}}$ could not serve all the SFCs $\bar{\Theta} = \ddot{\Theta} \cup \hat{\Theta}$. For this reason, new VNFIs could be instantiated to serve some new VNFs of $\hat{\Theta}$ as depicted in Fig. $2(f)$. We denote by $\hat{\mathcal{V}}$ the set of VNFIs that could be instantiated to serve the new VNFs $\bigcup_{\theta \in \hat{\Theta}} \Upsilon_\theta$. Note that more than one VNF in $(\bar{\Theta})$ could use the same VNFI in $(\mathcal{V})$. Based on the observation that each new VNF would be maximum instantiated at one VNFI, we have $|\hat{\mathcal{V}}| \leq |\bigcup_{\theta \in \hat{\Theta}} \Upsilon_\theta|$.

In the proposed model, one VNFI can be shared with multiple SFCs, and hence two VNFs $(\upsilon_1, \upsilon_2) \in \Upsilon_{\theta_1} \times \Upsilon_{\theta_2}$, that belong to two different SFCs $\theta_1, \theta_2 \in \bar{\Theta}$, could use the same VNFI $u \in \mathcal{V}$ iff the following conditions hold: i) They have the same type $\pi_{\upsilon_1} = \pi_{\upsilon_2}$; ii) There are no conflicts in the configuration of the two VNFs υ_1 and υ_2. The conflicts happen between the already enforced and the newly added security policies. Many types of security configuration conflicts should be prevented during the network orchestration ([21]: sub-section 5.3.2). For instance, friction happens when a deep packet inspector is deployed on a link in which the network channel is already encrypted. Conflict also occurs when the same firewall has been configured to accept and reject the traffic from the same subnet with the same priority level. Let $\zeta(\upsilon)$ denote all the VNFs that have the same type like υ. Formally $\zeta(\upsilon)$ is defined as follows: $\zeta(\upsilon) = \{\acute{\upsilon} : \forall \acute{\upsilon} \in \bigcup_{\acute{\theta} \in \bar{\Theta}} \Upsilon_{\acute{\theta}} \wedge \pi_\upsilon = \pi_{\acute{\upsilon}}\}$. Meanwhile, $\Psi(\upsilon)$ denotes all the VNFs that have conflicts with the VNF $\upsilon \in \Upsilon_\theta$. The SOO receives this list from the "Policy Interpreter" component. Formally, the candidate list of VNFs that can be hosted in the same VNFI with the VNF $\upsilon \in \Upsilon_\theta$ is $\zeta(\upsilon) - \Psi(\upsilon)$.

Let $\mathcal{R}$ denote the resources that are considered in this paper. $\mathcal{R}$ can be defined as follows: $\mathcal{R} = \{\text{RAM}, \text{CPU}, \text{Storage}\}$. Also, we denote by $\mathcal{F}$ the list of available flavors in the network. Each flavor $f \in \mathcal{F}$ has a specified amount of resources should be used by a VNF instance. Let $\delta_r(f)$ denote the amount of resources $r \in \mathcal{R}$ should be used by the flavor f. We denote by $\Omega(u)$ the set of VNFs that would be hosted at the VNFI $u \in \mathcal{V}$. Formally, there is a correlation between the resources used by a VNFI $u \in \mathcal{V}$, the amount of traffic

treated by that VNFI, and the expected services offered by that VNFI in terms of computation and QoS. While the amount of traffic negatively affects the QoS, the amount of resources has a positive impact on the QoS [22], [23]. We denote by π_u the VNF instance type. Note that a VNFI u whose type is similar to the VNFs that it hosts. Let $\Gamma(\pi_u, f, \sum_{\theta\in\bar{\Theta}, \upsilon\in\Upsilon_\theta\cap\Omega(\upsilon)} \lambda_\theta)$ a function that returns the average expected processing delay for handling the data traffic $\sum_{\theta\in\bar{\Theta}, \upsilon\in\Upsilon_\theta\cap\Omega(\upsilon)} \lambda_\theta$ by a VNFI u that uses the flavor $f \in \mathcal{F}$ and has a type $\pi_u \in \Pi$. For the sake of simplicity, we consider that Γ is a linear function in respect to the amount of data λ_θ. Therefore,

$$\Gamma(\pi_u, f, \sum_{\theta\in\bar{\Theta}, \upsilon\in\Upsilon_\theta\cap\Omega(u)} \lambda_\theta) = \sum_{\theta\in\bar{\Theta}, \upsilon\in\Upsilon_\theta\cap\Omega(u)} \Gamma(\pi_u, f, \lambda_\theta)$$

In this paper, a best effort communication model is considered within the same VNFI, and hence if two SFCs share the same VNFI, they will share both the same processing delay. This means that the processing delay will be increased by twice for each SFC.

V. Optimal security orchestration solution

In this section, we present the Algorithm that should be executed at the SOO component. First of all, we start by defining the different variables, then we will define the various constraints. The different variables used in this paper are summarized in the table II.

Table II: Summary of Variables Notations.

Variable	Description
$\mathcal{X}_{\upsilon,u}$	A Boolean decision variable that shows if the VNF $\upsilon \in \Upsilon_\theta$ of the SFC $\theta \in \Theta$ would use the VNFI $u \in \mathcal{V}$.
$\mathcal{D}_u^V$	The expected average time needed to process all the traffic by the VNFI u.
$\phi_{u,f}$	A decision variable that shows if the VNFI $u \in \mathcal{V}$ will use the flavor $f \in \mathcal{F}$.
$\mathcal{D}_u^V$	The expected time to handle the traffic by VNFI $u \in \mathcal{V}$.
$\delta_r(\upsilon)$	The amount of resources $r \in \mathcal{R}$ used by the VNF $\upsilon \in \Upsilon_\theta$.
$\mathcal{T}_{u,v}^V$	The amount of traffic should be transferred from VNFI $u \in \mathcal{V}$ to VNFI $v \in \mathcal{V}$.
$\mathcal{T}_c^C$	The amount of traffic should be handled by the cloud $c \in V_C$.
$\mathcal{Y}_{u,c}$	A Boolean decision variable that shows if the VNFI $u \in \mathcal{V}$ would be deployed at the edge/cloud $c \in V_C$.
$\mathcal{A}_{u,\pi}$	A Boolean decision variable that shows if a VNFI u has the type π.
$\mathcal{B}_{\theta,u}$	A Boolean decision variable that shows if an SFC $\theta \in \hat{\Theta}$ would use VNFI $u \in \mathcal{V}$.

As aforementioned, mainly the SOO has the following inputs:

- The set of users $\mathcal{U}$ in the network;
- The set of IoT devices $\mathcal{D}$ existing in the network;
- Underlying network $\mathcal{G}(V, E, W)$:
  - V consists of i) Clouds or edges (V_C); ii) Distributed IoT infrastructure ($V_I = \mathcal{D}$); iii) End-users ($V_U = \mathcal{U}$);
  - E represents the end-to-end connections between $V_U \cup V_I \cup V_C$;
  - W denotes the characteristics of the links that include the bandwidth, delay and security level. W could contain either actual characteristics that should be received from RQM module or future predicted characteristics that should be received from the data analytic module. W consists of bandwidth capacity ($\mathcal{W}^{\mathcal{B}}$), propagation delay ($\mathcal{W}^{\mathcal{L}}$) and security level ($\mathcal{W}^{\mathcal{S}}$);
- The set of already deployed VNFIs $\ddot{\mathcal{V}}$ that should be kept used in the next period;
- $\ddot{\mathcal{U}}_{u,c}$ is a parameter that specifies if a VNFI $u \in \ddot{\mathcal{V}}$ is deployed at a cloud $c \in V_C$ or not. $\ddot{\mathcal{U}}_{u,c} = 1$ if a VNFI u is already deployed in a cloud c, otherwise $\ddot{\mathcal{U}}_{u,c} = 0$. Note that a VNFI in $\ddot{\mathcal{V}}$ should be deployed only on one cloud, and hence $\sum_{c\in V_C} \ddot{\mathcal{U}}_{u,c} = 1$;
- The set of SFCs that should be considered in the future $\bar{\Theta} = \ddot{\Theta} \cup \hat{\Theta}$;
- For each SFC $\theta \in \bar{\Theta}$, we have the QoS and security level should be respected that are presented by $\xi_\theta^{\mathcal{L}}$, $\xi_\theta^{\mathcal{B}}$ and $\xi_\theta^{\mathcal{S}}$;
- For already deployed VNF $\upsilon \in \Upsilon_\theta$ for $\theta \in \ddot{\Theta}$, we have the following parameters that are included as input in the Algorithm:
  - $\ddot{\mathcal{X}}_{\upsilon,u}$ equals to 1 if the VNF υ uses the VNFI $u \in \ddot{\mathcal{V}}$. Note that a VNF υ should use only one VNFI, thus $\sum_{u\in\ddot{\mathcal{V}}} \ddot{\mathcal{X}}_{\upsilon,u} = 1$. Moreover, each already deployed SFC $\theta \in \ddot{\Theta}$ should use maximum one VNFI $u \in \ddot{\mathcal{V}}$ to prevent the loops. Formally, $u \in \ddot{\mathcal{V}}, \forall\theta \in \ddot{\Theta} : \sum_{\upsilon\in\Upsilon_\theta} \ddot{\mathcal{X}}_{\upsilon,u} \leq 1$;
  - Each VNF υ has a specified type π_υ;
  - Each VNFI $u \in \ddot{\mathcal{V}}$ has a specified type. We denote by $\ddot{\mathcal{A}}_{u,\pi}$ a parameter that equals to 1 if VNFI u uses has the type $\pi \in \Pi$. Moreover, VNFI u should have the same type as all the VNFs that use it. Formally, $\forall u \in \ddot{\mathcal{V}} : \ddot{\mathcal{X}}_{\upsilon,u} = 1 \implies \pi_\upsilon = \sum_{\pi\in\Pi} \pi \times \ddot{\mathcal{A}}_{u,\pi}$;
  - We also denote by $\ddot{\mathcal{Y}}_{\upsilon,c}$ a decision parameter that shows if the VNF υ is hosted at the cloud $c \in V_C$ or not. Note that the VNF υ is hosted at the cloud $c \in V_C$ if its VNFI is running on top of that cloud. $\sum_{c\in V_C} \ddot{\mathcal{Y}}_{\upsilon,c} = 1$. Formally, $\ddot{\mathcal{Y}}_{\upsilon,c}$ is defined as follow: $\forall c \in V_C : \ddot{\mathcal{Y}}_{\upsilon,c} = \sum_{u\in\ddot{\mathcal{V}}} \ddot{\mathcal{X}}_{\upsilon,u} \times \ddot{\mathcal{U}}_{u,c}$.

In what follows, we will present the different variables and constraints that should be used by the Algorithm of SOO.

A. VNF and VNFI relationship constraints

We define by $\mathcal{X}_{\upsilon,u}$ a decision Boolean variable that shows if the VNF $\upsilon \in \Upsilon_\theta$ of the SFC $\theta \in \Theta$ will use the VNFI $u \in \mathcal{V}$.

$$\forall\theta \in \hat{\Theta}, \forall\upsilon \in \Upsilon_\theta, \forall u \in \mathcal{V} : \quad \mathcal{X}_{\upsilon,u} = \begin{cases} 1 & \text{Iff } \upsilon \text{ uses VNFI } u \\ 0 & \text{Otherwise} \end{cases}$$

$\mathcal{X}_{\upsilon,u}$ should be subject to the following constraints:

Similar to the existing VNFs $\upsilon \in \Upsilon_\theta$ for $\theta \in \ddot{\Theta}$, new VNFs should use only one VNFI. Each new VNF, $\upsilon \in \Upsilon_\theta$ for $\theta \in \hat{\Theta}$, should use one and only one VNFI u in the network.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \sum_{u \in \mathcal{V}} \mathcal{X}_{\upsilon,u} = 1 \quad (1)$$

Let $\mathcal{B}_{\theta,u}$ denote a Boolean decision variable that shows if an SFC $\theta \in \hat{\Theta}$ would use VNFI $u \in \mathcal{V}$.

$\forall \theta \in \hat{\Theta}, \forall u \in \mathcal{V}$:

$$\mathcal{B}_{\theta,u} = \begin{cases} 1 & \text{Iff } \theta \text{ uses VNFI } u \\ 0 & \text{Otherwise} \end{cases}$$

In order to prevent the loop in the SFC θ, VNFs ($\forall \upsilon \in \Upsilon_\theta$) of that SFC should use the VNFI u maximum once. The constraint (2) prevents the loop in the SFC, as well as it ensures that $\mathcal{B}_{\theta,u} = 1$ if one of θ's VNFs uses u.

$$\forall u \in \mathcal{V}, \forall \theta \in \hat{\Theta} : \sum_{\upsilon \in \Upsilon_\theta} \mathcal{X}_{\upsilon,u} = \mathcal{B}_{\theta,u} \quad (2)$$

The following constraints ensure that a new VNF υ should not share a VNFI $u \in \mathcal{V}$ with another VNF with whom has a conflict or does not have the same type. Let we define two sets of conflicts $\Psi(\upsilon)$ for the VNF υ. These two sets are: $i)$ $\ddot{\Psi}(\upsilon) = \Psi(\upsilon) \cap (\bigcup_{\theta \in \ddot{\Theta}} \Upsilon_\theta)$; $ii)$ $\hat{\Psi}(\upsilon) = \Psi(\upsilon) \cap (\bigcup_{\theta \in \hat{\Theta}} \Upsilon_\theta)$. Formally, $\ddot{\Psi}(\upsilon)$ denotes all deployed VNFs that have security conflict with υ while $\hat{\Psi}(\upsilon)$ denotes all the VNFs that should be deployed and have a conflict with υ.

We define by $\mathcal{Y}_{\upsilon,c}$ a decision Boolean variable that shows if a VNF $\upsilon \in \Upsilon_\theta$ for $\theta \in \bar{\Theta}$ is hosted at the cloud $c \in V_C$ or not. Formally a VNF υ is hosted at a cloud $c \in V_C$ if its VNFI is running on top of that cloud.

$\forall \theta \in \bar{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall c \in V_C$:

$$\mathcal{Y}_{\upsilon,c} = \begin{cases} 1 & \text{Iff } \upsilon \text{ is hosted at cloud } c \\ 0 & \text{Otherwise} \end{cases}$$

Each VNF should run on top only one cloud at a given time. Constraint (3) ensures that each VNF should be deployed only on one cloud.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \sum_{c \in V_C} \mathcal{Y}_{\upsilon,c} = 1 \quad (3)$$

We define also by $\mathcal{A}_{u,\pi}$ a decision Boolean variable that shows if the VNFI $u \in \hat{\mathcal{V}}$ has the type $\pi \in \Pi$.

$\forall u \in \hat{\mathcal{V}}, \forall \pi \in \Pi$:

$$\mathcal{A}_{u,\pi} = \begin{cases} 1 & \text{Iff } u \text{ has type } \pi \\ 0 & \text{Otherwise} \end{cases}$$

Each new VNF $\upsilon \in \Upsilon_\theta$ for $\forall \theta \in \hat{\Theta}$ that uses existing VNFIs $u \in \ddot{\mathcal{V}}$ should use the same VNF type as mentioned in constraint 4.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \ddot{\mathcal{V}}, \forall \pi \in \Pi : \pi_\upsilon \times \mathcal{X}_{\upsilon,u} = \pi \times \ddot{\mathcal{A}}_{u,\pi} \times \mathcal{X}_{\upsilon,u} \quad (4)$$

From another side, constraint 5 ensures that each new VNFI $u \in \hat{\mathcal{V}}$ should have the same type as the VNFs that use it.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \hat{\mathcal{V}}, \forall \pi \in \Pi : \pi_\upsilon \times \mathcal{X}_{\upsilon,u} = \pi \times \mathcal{A}_{u,\pi} \times \mathcal{X}_{\upsilon,u} \quad (5)$$

Unfortunately, the constraint (5) is not linear due to the part $\mathcal{A}_{u,\pi} \times \mathcal{X}_{\upsilon,u}$. In order to make this constraint linear, we replace the constraint (5) with the constraints (6 and (7).

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \hat{\mathcal{V}}, \forall \pi \in \Pi : \pi_\upsilon \leq \pi + (2 - \mathcal{X}_{\upsilon,u} - \mathcal{A}_{u,\pi}) \times \mathcal{M} \quad (6)$$

, where $\mathcal{M}$ is a big number ($\mathcal{M} \approx \infty$).

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \hat{\mathcal{V}}, \forall \pi \in \Pi : \pi \leq \pi_\upsilon + (2 - \mathcal{X}_{\upsilon,u} - \mathcal{A}_{u,\pi}) \times \mathcal{M} \quad (7)$$

The following constraint ensures that each VNFI $u \in \hat{\mathcal{V}}$ should have only one type if and only of it is deployed, otherwise it should not have any type.

$$\forall u \in \hat{\mathcal{V}} : \sum_{\pi \in \Pi} \mathcal{A}_{u,\pi} = \sum_{c \in V_C} \mathcal{U}_{u,c} \quad (8)$$

Constraints (6), (7) and (8) ensure VNFs should have the same type of their VNFIs, which prevent the deployment of the VNFs that have different types on the same VNFI.

B. VNFI and cloud relationship constraints

We also define by $\mathcal{U}_{u,c}$ a Boolean variable that shows if the VNFI $u \in \hat{\mathcal{V}}$ would be deployed in the cloud/edge $c \in V_C$

$\forall u \in \hat{\mathcal{V}}, \forall c \in V_C$:

$$\mathcal{U}_{u,c} = \begin{cases} 1 & \text{Iff VNFI } u \text{ would be deployed in} \\ & \text{the cloud/edge c} \\ 0 & \text{Otherwise} \end{cases}$$

A new VNFI $u \in \hat{\mathcal{V}}$ should be deployed if and only if it is used at least by one VNF as presented by the constraint (9).

$$\forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \sum_{c \in V_C} \mathcal{U}_{u,c} \geq \mathcal{X}_{\upsilon,u} \quad (9)$$

Also, a VNFI $u \in \hat{\mathcal{V}}$ should be not deployed if it is not used by any VNF in the network as show in the constraint (10).

$$\forall u \in \hat{\mathcal{V}} : \sum_{c \in V_C} \mathcal{U}_{u,c} \leq \sum_{\theta \in \bar{\Theta}, \upsilon \in \Upsilon_\theta} \mathcal{X}_{\upsilon,u} \quad (10)$$

Finally, each VNFI $u \in \hat{\mathcal{V}}$ should be deployed at most at one cloud as shown in the constraint (11).

$$\forall u \in \hat{\mathcal{V}} : \sum_{c \in V_C} \mathcal{U}_{u,c} \leq 1 \tag{11}$$

A VNF should use the same cloud where its VNFI is deployed. Constraint (12) ensures that each VNF that uses already existing VNFI, then it should use the same cloud as that VNFI.

$$\forall c \in V_C, \forall u \in \ddot{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \ddot{\mathcal{U}}_{u,c} = 1 \implies \mathcal{Y}_{\upsilon,c} \geq \mathcal{X}_{\upsilon,u} \tag{12}$$

Meanwhile, Constraint (13) ensures that each VNF that uses a new VNFI, then it should use the same cloud as that VNFI. While constraint (14) ensures that a VNF υ should not use a cloud c if its VNFI is not running on that cloud.

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,c} \geq \mathcal{X}_{\upsilon,u} \times \mathcal{U}_{u,c} \tag{13}$$

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,c} \leq \sum_{u \in \hat{\mathcal{V}}} \mathcal{X}_{\upsilon,u} \times \mathcal{U}_{u,c} + \sum_{u \in \ddot{\mathcal{V}}} \mathcal{X}_{\upsilon,u} \times \ddot{\mathcal{U}}_{u,c} \tag{14}$$

Unfortunately, the constraint (13) and (14) are not linear. To make the optimization linear, we have replaced the constraints (13) and (14) with the following variables and constraints:

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \hat{\mathcal{V}}, \forall c \in V_C : \mathcal{Y}_{\upsilon,u,c} = \begin{cases} 1 & \text{Iff VNF } \upsilon \text{ would use VNFI } u \text{ that} \\ & \text{should be deployed at cloud/edge c} \\ 0 & \text{Otherwise} \end{cases}$$

We replace the constraint (13) and (14) with the following constraints:

Constraints (15) and (16) ensure that $\mathcal{Y}_{\upsilon,u,c} = 0$ if υ does not use u (i.e., $\mathcal{X}_{\upsilon,u} = 0$) or u is not deployed at cloud c (i.e., $\mathcal{U}_{u,c} = 0$).

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,u,c} \leq \mathcal{U}_{u,c} \tag{15}$$

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,u,c} \leq \mathcal{X}_{\upsilon,u} \tag{16}$$

Constraint (17) ensures that $\mathcal{Y}_{\upsilon,u,c} = 1$ iff both variables $\mathcal{X}_{\upsilon,u}$ and $\mathcal{U}_{u,c}$ equal to 1.

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,u,c} \geq \mathcal{X}_{\upsilon,u} + \mathcal{U}_{u,c} - 1 \tag{17}$$

Meanwhile, constraint (18) ensures that the VNF υ is deployed in cloud c (i.e., $\mathcal{Y}_{\upsilon,c} = 1$) if $\mathcal{Y}_{\upsilon,u,c} = 1$.

$$\forall c \in V_C, \forall u \in \hat{\mathcal{V}}, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,c} \geq \mathcal{Y}_{\upsilon,u,c} \tag{18}$$

Constraint (19) ensures that a VNF υ is not deployed in a cloud c (i.e., $\mathcal{Y}_{\upsilon,c} = 0$) if it does not use any VNFI deployed at cloud c.

$$\forall c \in V_C, \forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \mathcal{Y}_{\upsilon,c} \leq \sum_{u \in \hat{\mathcal{V}}} \mathcal{Y}_{\upsilon,u,c} + \sum_{u \in \ddot{\mathcal{V}}} \mathcal{X}_{\upsilon,u} \times \ddot{\mathcal{U}}_{u,c} \tag{19}$$

Meanwhile, the constraint (20) ensures that each VNF υ should be hosted at a cloud.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta : \sum_{c \in V_C} \mathcal{Y}_{\upsilon,c} = 1 \tag{20}$$

C. Resource aware constraints

We define by $\phi_{u,f}$ a decision Boolean variable that shows if VNFI $u \in \mathcal{V}$ would use a flavor $f \in \mathcal{F}$.

$$\forall u \in \hat{\mathcal{V}}, \forall f \in \mathcal{F} : \phi_{u,f} = \begin{cases} 1 & \text{Iff VNFI } u \text{ uses flavor } f \\ 0 & \text{Otherwise} \end{cases}$$

First of all, each VNFI should use only one flavor if it is deployed as shown in the constraint (21).

$$\forall u \in \mathcal{V} : \sum_{f \in \mathcal{F}} \phi_{u,f} = \sum_{c \in V_C} \mathcal{U}_{u,c} \tag{21}$$

The constraint (22) ensures that each cloud should not overloaded.

$$\forall r \in \mathcal{R}, \forall c \in V_C : \sum_{f \in \mathcal{F}} \delta_r(f) \times \Big(\sum_{u \in \ddot{\mathcal{V}}} \ddot{\phi}_{u,f} \times \ddot{\mathcal{U}}_{u,c} + \sum_{u \in \hat{\mathcal{V}}} \phi_{u,f} \times \mathcal{U}_{u,c} \Big) \leq \Delta_r(c) \tag{22}$$

Unfortunately, the constraint (22) is not linear due to the part $\phi_{u,f} \times \mathcal{Y}_{u,c}$. To convert the optimization to linear optimization. we replace the constraint (22) by the following constraints and variables:

First, we define the variable $\mathcal{U}_{u,c,f}$ that shows if the VNFI u uses the flavor f in the cloud c.

$$\forall u \in \mathcal{V}, \forall f \in \mathcal{F}, \forall c \in V_C : \mathcal{U}_{u,c,f} = \begin{cases} 1 & \text{Iff VNFI } u \text{ uses flavor } f \text{ in the cloud } c \\ 0 & \text{Otherwise} \end{cases}$$

Also, we replace the constraint 22 with the following constraints.

$$\forall c \in V_C, \forall f \in \mathcal{F}, u \in \hat{\mathcal{V}} : \mathcal{U}_{u,c,f} \leq \mathcal{U}_{u,c} \tag{23}$$

$$\forall c \in V_C, \forall f \in \mathcal{F}, u \in \hat{\mathcal{V}} : \mathcal{U}_{u,c,f} \leq \phi_{u,f} \tag{24}$$

$$\forall c \in V_C, \forall f \in \mathcal{F}, u \in \hat{\mathcal{V}} : \mathcal{U}_{u,c,f} \geq \mathcal{U}_{u,c} + \phi_{u,f} - 1 \tag{25}$$

$$\forall r \in \mathcal{R}, \forall c \in V_C : \sum_{f \in \mathcal{F}} \delta_r(f) \times \big(\sum_{u \in \ddot{\mathcal{V}}} \ddot{\phi}_{u,f} \times \ddot{\mathcal{U}}_{u,c} + \sum_{u \in \hat{\mathcal{V}}} \mathcal{U}_{u,c,f} \big) \leq \Delta_r(c) \tag{26}$$

D. QoS aware constraints

Let $\mathcal{T}_u$ denote processing delay expected by the VNFI $u \in \mathcal{V}$. The variable $\mathcal{T}_u$ should be subject to the following constraints:

The processing delay for already deployed VNFI $\forall u \in \ddot{\mathcal{V}}$ is computed using the constraint (27):

$$\forall u \in \ddot{\mathcal{V}} : \mathcal{T}_u = \sum_{f \in \mathcal{F}} \sum_{\pi \in \Pi} \ddot{\mathcal{A}}_{u,\pi} \times \ddot{\phi}_{u,f} \times \big(\Gamma(\pi, f, \sum_{\theta \in \ddot{\Theta}, \upsilon \in \Upsilon_\theta} \lambda_\theta \times \ddot{\mathcal{X}}_{\upsilon,u}) + \sum_{\theta \in \hat{\Theta}} \Gamma(\pi, f, \lambda_\theta) \times \mathcal{B}_{\theta,u} \big) \tag{27}$$

Note that $\sum_{\theta \in \ddot{\Theta}, \upsilon \in \Upsilon_\theta} \ddot{\mathcal{X}}_{\upsilon,u} \leq 1$ as two VNFs at the same SFC θ does not use the same VNFI u.

Meanwhile, the processing delay of a new VFNI $\forall u \in \hat{\mathcal{V}}$ is computed as follow:

$$\forall u \in \hat{\mathcal{V}} : \mathcal{T}_u = \sum_{f \in \mathcal{F}} \sum_{\pi \in \Pi} \sum_{\theta \in \hat{\Theta}} \Gamma(\pi, f, \lambda_\theta) \times \mathcal{B}_{\theta,u} \times \mathcal{A}_{u,\pi} \times \phi_{u,f} \tag{28}$$

However, the constraint (28) is not linear due to the term $\mathcal{B}_{\theta,u} \times \mathcal{A}_{u.\pi} \times \phi_{u,f}$. In order to convert the optimization to linear optimization, we replace the constraint 28 with the following constraints and variables:

First of all we add the following variable $\mathcal{C}_{\theta,u,\pi,f}$ that equals to 1 if the SFC $\theta \in \hat{\Theta}$ uses the VNFI $u \in \hat{\mathcal{V}}$ that has the flavor f and the type π.

$$\forall \theta \in \hat{\Theta}, \forall u \in \hat{\mathcal{V}}, \forall f \in \mathcal{F}, \forall \pi \in \Pi : \mathcal{C}_{\theta,u,\pi,f} \leq \mathcal{B}_{\theta,u} \tag{29}$$

$$\forall \theta \in \hat{\Theta}, \forall u \in \hat{\mathcal{V}}, \forall f \in \mathcal{F}, \forall \pi \in \Pi : \mathcal{C}_{\theta,u,\pi,f} \leq \mathcal{A}_{u,\pi} \tag{30}$$

$$\forall \theta \in \hat{\Theta}, \forall u \in \hat{\mathcal{V}}, \forall f \in \mathcal{F}, \forall \pi \in \Pi : \mathcal{C}_{\theta,u,\pi,f} \leq \phi_{u,f} \tag{31}$$

$$\forall \theta \in \hat{\Theta}, \forall u \in \hat{\mathcal{V}}, \forall f \in \mathcal{F}, \forall \pi \in \Pi : \mathcal{C}_{\theta,u,\pi,f} \geq \mathcal{B}_{\theta,u} + \mathcal{A}_{u,\pi} + \phi_{u,f} - 2 \tag{32}$$

$$\forall u \in \hat{\mathcal{V}} : \mathcal{T}_u = \sum_{f \in \mathcal{F}} \sum_{\pi \in \Pi} \sum_{\theta \in \hat{\Theta}} \Gamma(\pi, f, \lambda_\theta) \times \mathcal{C}_{\theta,u,\pi,f} \tag{33}$$

Let $\mathcal{T}_\upsilon^V$ denote processing delay expected by the VNF $\upsilon \in \Upsilon_\theta$ for $\theta \in \Theta$. The variable $\mathcal{T}_\upsilon^V$ should be subject to the following constraints:

First, we define the constraint that has relationship with the already exiting VNFs $\upsilon \in \Upsilon_\theta$ for $\theta \in \ddot{\Theta}$.

$$\forall \theta \in \ddot{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \ddot{\mathcal{V}} : \mathcal{T}_\upsilon^V \leq \mathcal{T}_u + (1 - \ddot{\mathcal{X}}_{\upsilon,u}) \times \mathcal{M} \tag{34}$$

$$\forall \theta \in \ddot{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \ddot{\mathcal{V}} : \mathcal{T}_u \leq \mathcal{T}_\upsilon^V + (1 - \ddot{\mathcal{X}}_{\upsilon,u}) \times \mathcal{M} \tag{35}$$

Second, we define the constraint that has relationship with the new VNFs $\upsilon \in \Upsilon_\theta$ for $\theta \in \hat{\Theta}$.

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \mathcal{V} : \mathcal{T}_\upsilon^V \leq \mathcal{T}_u + (1 - \mathcal{X}_{\upsilon,u}) \times \mathcal{M} \tag{36}$$

$$\forall \theta \in \hat{\Theta}, \forall \upsilon \in \Upsilon_\theta, \forall u \in \mathcal{V} : \mathcal{T}_u \leq \mathcal{T}_\upsilon^V + (1 - \mathcal{X}_{\upsilon,u}) \times \mathcal{M} \tag{37}$$

The following constraint ensures that the end-to-end delay in the SFCs is respected.

First, we define $\mathcal{T}_{c_1,c_2}^{\mathcal{C}}$ that denotes the expected delay between two different clouds $c_1, c_2 \in V_C, c_1 \neq c_2$ after binding and deploying the SFCs.

$$\forall c_1, c_2 \in V_C, c_1 \neq c_2 : \mathcal{T}_{c_1,c_2}^{\mathcal{C}} = \frac{1}{\mathcal{W}_{c_1,c_2}} \big[\sum_{\theta \in \ddot{\Theta}} \sum_{(\upsilon_1,\upsilon_2) \in \Upsilon_\theta} \lambda_\theta \times \ddot{\mathcal{Y}}_{\upsilon_1,c_1} \times \ddot{\mathcal{Y}}_{\upsilon_2,c_2} + \sum_{\theta \in \hat{\Theta}} \sum_{(\upsilon_1,\upsilon_2) \in \Upsilon_\theta} \lambda_\theta \times \mathcal{Y}_{\upsilon_1,c_1} \times \mathcal{Y}_{\upsilon_2,c_2} \big] \tag{38}$$

However, the constraint (38) is not linear due to the part $\mathcal{Y}_{\upsilon_1,c_1} \times \mathcal{Y}_{\upsilon_2,c_2}$. In order to make the optimization linear, we replace the constraint (38) by the following variables and constraints:

$$\forall \theta \in \bar{\Theta}, \forall (\upsilon_1, \upsilon_2) \in \Upsilon_\theta, \forall c_1, c_2 \in V_C, c_1 \neq c_2 : \mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2} = \begin{cases} 1 & \text{Iff } \upsilon_1 \text{ and } \upsilon_2 \text{ are hosted at} \\ & \text{cloud } c_1 \text{ and } c_2, \text{ respectively} \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall \theta \in \bar{\Theta}, \forall (\upsilon_1, \upsilon_2) \in \Upsilon_\theta, \forall c_1, c_2 \in V_C, c_1 \neq c_2 : \mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2} \leq \mathcal{Y}_{\upsilon_1,c_1} \tag{39}$$

$$\forall\theta\in\bar{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta,\forall c_1,c_2\in V_C,c_1\neq c_2:\\ \mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2}\leq\mathcal{Y}_{\upsilon_2,c_2} \quad (40)$$

$$\forall\theta\in\bar{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta,\forall c_1,c_2\in V_C,c_1\neq c_2:\\ \mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2}\geq\mathcal{Y}_{\upsilon_1,c_1}+\mathcal{Y}_{\upsilon_2,c_2}-1 \quad (41)$$

$$\forall c_1,c_2\in V_C,c_1\neq c_2:\\ \mathcal{T}^{\mathcal{C}}_{c_1,c_2}=\frac{1}{\mathcal{W}_{c_1,c_2}}\Big[\sum_{\theta\in\ddot{\Theta}}\sum_{(\upsilon_1,\upsilon_2)\in\Upsilon_\theta}\lambda_\theta\times\ddot{\mathcal{Y}}_{\upsilon_1,c_1}\times\ddot{\mathcal{Y}}_{\upsilon_2,c_2}\\ +\sum_{\theta\in\hat{\Theta}}\sum_{(\upsilon_1,\upsilon_2)\in\Upsilon_\theta}\lambda_\theta\times\mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2}\Big] \quad (42)$$

Then, we will define the propagation delay between two consecutive VNFs in an SFC. Let $\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}$ denote a propagation delay between two consecutive VNFs $(\upsilon_1,\upsilon_2)\in\Upsilon_\theta$ for $\theta\in\bar{\Theta}$. We have two sets of constraints that define the propagation delay between two consecutive VNFs υ_1 and υ_2 according to the state of their SFCs either is already deployed or not yet.

In what follow, we define $\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}$ for already scheduled SFCs $\theta\in\ddot{\Theta}$.

$$\forall c_1,c_2\in V_C,c_1\neq c_2,\forall\theta\in\ddot{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta:\\ \mathcal{T}^{V}_{\upsilon_1,\upsilon_2}\leq\mathcal{T}^{\mathcal{C}}_{c_1,c_2}+(1-\ddot{\mathcal{Y}}_{\upsilon_1,c_1}\times\ddot{\mathcal{Y}}_{\upsilon_2,c_2})\times\mathcal{M} \quad (43)$$

$$\forall c_1,c_2\in V_C,c_1\neq c_2,\forall\theta\in\ddot{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta:\\ \mathcal{T}^{\mathcal{C}}_{c_1,c_2}\leq\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}+(1-\ddot{\mathcal{Y}}_{\upsilon_1,c_1}\times\ddot{\mathcal{Y}}_{\upsilon_2,c_2})\times\mathcal{M} \quad (44)$$

Then, we define $\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}$ for unscheduled SFCs $\theta\in\hat{\Theta}$.

$$\forall c_1,c_2\in V_C,c_1\neq c_2,\forall\theta\in\hat{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta:\\ \mathcal{T}^{V}_{\upsilon_1,\upsilon_2}\leq\mathcal{T}^{\mathcal{C}}_{c_1,c_2}+(1-\mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2})\times\mathcal{M} \quad (45)$$

$$\forall c_1,c_2\in V_C,c_1\neq c_2,\forall\theta\in\hat{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta:\\ \mathcal{T}^{\mathcal{C}}_{c_1,c_2}\leq\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}+(1-\mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2})\times\mathcal{M} \quad (46)$$

In what follow, we will define the constraints that ensure end-to-end delay of each SFC is respected. Thus, the end-to-end delay of each SFC $\theta\in\bar{\Theta}$ does not exceed its threshold $\xi^{\mathcal{L}}_\theta$.

$$\forall\theta\in\bar{\Theta}:\\ \underbrace{\sum_{\upsilon\in\Upsilon_\theta}\mathcal{T}^{V}_{\upsilon}}_{(47.a)}+\underbrace{\sum_{(\upsilon_1,\upsilon_2)\in\Upsilon_\theta}\mathcal{T}^{V}_{\upsilon_1,\upsilon_2}}_{(47.b)}\leq\xi^{\mathcal{L}}_\theta \quad (47)$$

While the part (47.a) presents the computation delay expected by an SFC $\theta\in\bar{\Theta}$, the second part (47.b) presents the propagation delay expected in the SFC θ.

Finally, we need to ensure that all the paths selected by the VNFs of a given SFC respect the constraints of security. Indeed, all the links that interconnect VNFs of the same SFC $\theta\in\bar{\Theta}$ should have a security level higher than the security level $\xi^{\mathcal{S}}_\theta$.

$$\forall c_1,c_2\in V_C,c_1\neq c_2,\forall\theta\in\hat{\Theta},\forall(\upsilon_1,\upsilon_2)\in\Upsilon_\theta:\\ \xi^{\mathcal{S}}_\theta\times\mathcal{Y}_{\upsilon_1,c_1,\upsilon_2,c_2}\leq\mathcal{W}^{\mathcal{S}}_{c_1,c_2} \quad (48)$$

E. Final Optimization of SOO component

In what follow, we present the global optimization solution that aims to reduce the cost while taking into account the QoS during the LCM of different SFCs.

$$\mathbf{min}\sum_{u\in\hat{\mathcal{V}},f\in\mathcal{F}}\delta_p(f)\times\phi_{u,f} \quad (49)$$

S.t,

VNF and VNFI relationship constraints:

(1), (2), (3), (4), (6), (7) and (8).

VNFI and cloud relationship constraints:

(9), (10), (11), (12), (15), (16), (17), (18), (19) and (20).

Resource aware constraints:

(21), (23), (24), (25) and (26).

QoS aware constraints:

(27), (29), (30), (31), (32), (33), (34), (35), (36), (37), (39), (40), (41), (42), (43), (44), (45), (46), (47) and (48).

VI. Efficient QoS and Resource-aware Security Orchestration Solution

In this section, we proposed a heuristic solution that aims for finding near optimal configurations in a reasonable time. The heuristic, dubbed Efficient QoS and Resource-aware Security Orchestration Algorithm (EQRSO), aims to ensure the efficient LCM of VNFs that are potentially deployed across IoT domains, inter-clouds and cross network segments. The different steps of the heuristic are summarised in Algorithm 1. The Algorithm has the following inputs: i) The underlying network graph $\mathcal{G}(V,E,W)$; ii) The already deployed SFCs $\ddot{\Theta}$; iii) The set of new SFCs that should be deployed $\hat{\Theta}$; iv) Φ_θ that denotes the users of the SFC θ; v) The already deployed VNF instances $\ddot{\mathcal{V}}$; vi) $\zeta(\upsilon)$ that denotes all the VNFs that have the same type like the VNF υ; vii) $\Psi(\upsilon)$ that denotes all the VNFs that have a conflicts with the VNF υ. While the outputs of the Algorithms are the new VNF instances $\hat{\mathcal{V}}$ that should be instantiated to serve the new SFCs and their VNFs.

Algorithm 1: Efficient QoS and Resource-aware Security Orchestration Algorithm (EQRSO)

Input :
$\mathcal{G}(V, E, W)$**: The underlying network graph.**
$\ddot{\Theta}$**: The already deployed SFCs and would be kept in the future.**
$\hat{\Theta}$**: The set of new SFCs that should be deployed in the future for serving the users.**
Φ_θ**: The users of the SFC** $\theta \in \ddot{\Theta} \cup \hat{\Theta}$**.**
$\ddot{\mathcal{V}}$**: The already deployed VNF instances that serve already existing VNFs.**
$\zeta(\upsilon)$**: All the VNFs that have the same type like the VNF** υ**.**
$\Psi(\upsilon)$**: All the VNFs that have conflicts with the VNF** υ**.**
Output:
$\hat{\mathcal{V}}$: The new VNF instances that should be instantiated to serve the new VNFs.

1 $\hat{\mathcal{V}} \leftarrow \emptyset$;
2 $\Theta \leftarrow \ddot{\Theta}$;
3 **for** $\theta \in \hat{\Theta}$ **do**
4 **for** $\upsilon \in \Upsilon_\theta$ **do**
5 $c_1 \leftarrow \upsilon.pred.c$;
6 **for** $u \in sorted(\hat{\mathcal{V}} \cup \ddot{\mathcal{V}})$ **do**
7 **if** $Z(u) \cap \zeta(\upsilon) \neq Z(u) \vee Z(u) \cap \Psi(\upsilon) \neq \emptyset$ **then**
8 $continue$;
9 **end**
10 $c_2 \leftarrow u.c$;
11 **if** $\xi_\theta^{\mathcal{S}} > \mathcal{W}_{c_1,c_2}^{\mathcal{S}}$ **then**
12 $continue$;
13 **end**
14 $\upsilon_u \leftarrow u$;
15 $u_\theta \leftarrow u_\theta \cup \{\theta\}$;
16 **if** $QoS\_Satisfied(\Theta, \theta) = True$ **then**
17 $break$;
18 **end**
19 $\upsilon_u \leftarrow \emptyset$;
20 $u_\theta \leftarrow u_\theta \setminus \{\theta\}$;
21 **end**
22 **if** $\upsilon_u = \emptyset$ **then**
23 $c_2, u_f \leftarrow \eta(\Theta, \theta)$;
24 $\upsilon_u \leftarrow u$;
25 $u_\pi \leftarrow \upsilon_\pi$;
26 $u_\theta \leftarrow u_\theta \cup \{\theta\}$;
27 $u_c \leftarrow u_c \cup \{c_2\}$;
28 $\upsilon_u.c \leftarrow c_2$;
29 $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}} \cup \{u\}$;
30 **end**
31 **end**
32 $\Theta \leftarrow \Theta \cup \theta$;
33 **end**

The heuristic firstly starts by initiating the new VNF instances $\hat{\mathcal{V}}$ with an empty set (Algorithm 1: Line 1). The Algorithm should assign VNF instances to the VNFs without creating conflicts with already scheduled SFCs within the Algorithm. In the Algorithm, we denote by Θ the set of scheduled SFCs in the Algorithm. Formally, $\Theta = \ddot{\Theta} \cup \hat{\Theta}$. Initially, Θ should be initialized with already scheduled SFCs $\ddot{\Theta}$ (Algorithm 1: Line 2). Then, the Algorithm schedules the SFCs $\hat{\Theta}$ in a loop (Algorithm 1: Lines $3-33$). At each iteration one SFC should be scheduled. The VNFs of the same SFC $\theta \in \hat{\Theta}$ are also scheduled one by one (Algorithm 1: Lines $5-32$). In the Algorithm, we primarily aiming at the use of already existing VNFIs and the deployment of new VNFIs on the last resort basis, as a "capacitor". The Algorithm initially starts looking for already deployed VNFIs (Algorithm 1: Lines $6-21$), then if it could not use an existing VNFI, the Algorithm should deploy a new VNFI (Algorithm 1: Lines $22-30$).

$$\forall c_1, c_2 \in V_C, c_1 \neq c_2 : \mathcal{T}_{c_1,c_2} = \frac{1}{\mathcal{W}_{c_1,c_2}} \sum_{\theta' \in \Theta \cup \{\theta'\}} \sum_{(\upsilon_1,\upsilon_2) \in \Upsilon_{\theta'} \wedge \upsilon_1.c = c_1 \wedge \upsilon_2.c = c_2} \lambda_{\theta'} \tag{50}$$

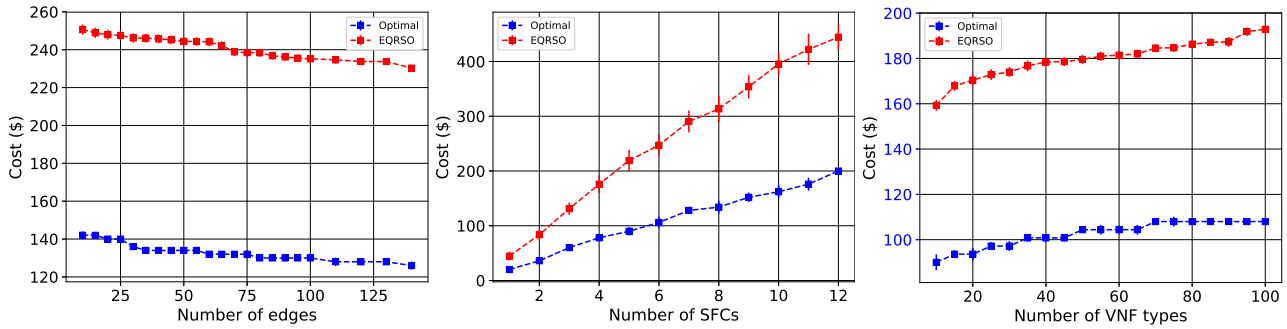
$$\forall u \in \mathcal{V} : \mathcal{T}_u = \sum_{\theta' \in u_\Theta} \Gamma(u_\pi, u_f, \lambda_{\theta'}) \tag{51}$$

The Algorithm goes through the VNFs Υ_θ of the same SFC θ one by one (Algorithm 1: Line 4). For each VNF $\upsilon \in \Upsilon_\theta$, the Algorithm finds the edge c_1 of its predecessor (Algorithm 1: Line 5). Note that $c_1 = \emptyset$ for the first VNF in the SFC θ. The Algorithm first starts looking for the sorted list of already deployed VNFIs. The latter includes previously scheduled VNFIs ($\ddot{\mathcal{V}}$) and the scheduled VNFIs within the Algorithm ($\hat{\mathcal{V}}$). The Algorithm checks the VNFIs one by one starting from the one that have more resources (Algorithm 1: Line 6). The Algorithm schedules the VNF υ in the VNFI $u \in sorted(\hat{\mathcal{V}} \cup \ddot{\mathcal{V}})$ if the following statements hold:

- The VNFI u should host the VNFs that have the same type like υ. We denote by $Z(u)$ the set of VNFs hosted at the VNFI u. Formally, the VNFI u should be skipped if all the VNFs hosted at it do not have the same type like the VNF υ (i.e., $Z(u) \cap \zeta(\upsilon) \neq Z(u)$). (Algorithm 1: Line 7);;
- The VNFI u should not host a VNF that hast a conflict with the VNF υ ($Z(u) \cap \Psi(\upsilon) \neq \emptyset$);
- The security level of the links should be respected. The Algorithm should deploy the VNF υ on the edge $c2$ of the VNFI u that ensures the required security level (Algorithm 1: Line 11). In other words, the security requirement should not be violated. The Algorithm checks if the security link level between the previous VNF's cloud/edge and $c2$ is not lower than the required threshold ($\xi_\theta^{\mathcal{S}}$) (Algorithm 1: Line 12);
- Finally, the Algorithm checks if the schedule of VNF ϵ at the VNFI u should not have a negative impact on the expected QoS should be perceived by the SFC θ and already scheduled SFC Θ (Algorithm 1: Lines $16-18$).

$$\forall \theta^{'} \in \Theta \cup \{\theta^{'}\} : \sum_{\upsilon \in \Upsilon_{\theta'}} \mathcal{T} + \sum_{(\upsilon_1,\upsilon_2) \in \Upsilon_\theta} \mathcal{T}_{\upsilon_1.c,\upsilon_2.c} \leq \xi_\theta^{\mathcal{L}} \tag{52}$$

In order to ensure the QoS, the Algorithm first assigns the VNF υ to the VNFI u (Algorithm 1: Lines $16-18$). Then, it checks if the QoS of all the SFCs $\Theta \cup \{\theta\}$ is respected using the function $QoS\_Satisfied(\Theta, \theta)$. If the QoS is satisfied, the Algorithm schedules the VNF υ by exiting the loop and going to the next VNF (Algorithm 1: Lines $16-18$). Otherwise, the VNFI u should not be considered and the Algorithm goes to the next VNFI (Algorithm 1: Lines $19-20$). The function



(a) Varying the numbers of clouds (b) Varying the numbers of SFCs (c) Varying the numbers of VNF types

Figure 3: Performances of the proposed solutions in terms of cost

$QoS\_Satisfied(\Theta, \theta)$ ensures that the end-to-end delay of all the SFCs $\Theta \cup \theta$ does not exceed the threshold $\xi_{\theta' \mathcal{L}}$. This function checks if the inequality (52) is verified for all the SFCs $\cup\theta$. It returns true if all the end-to-end delays of the SFCs $\theta^{'} \in \Theta \cup \theta$ do not exceed the threshold $\xi_{\theta' \mathcal{L}}$. Otherwise, the function returns false. Similarly to the inequality (47), this inequality has two main parts, which are the computational and propagation delay. While the propagation is computed using the equality (50), the computation delay is computed in the equation (51).

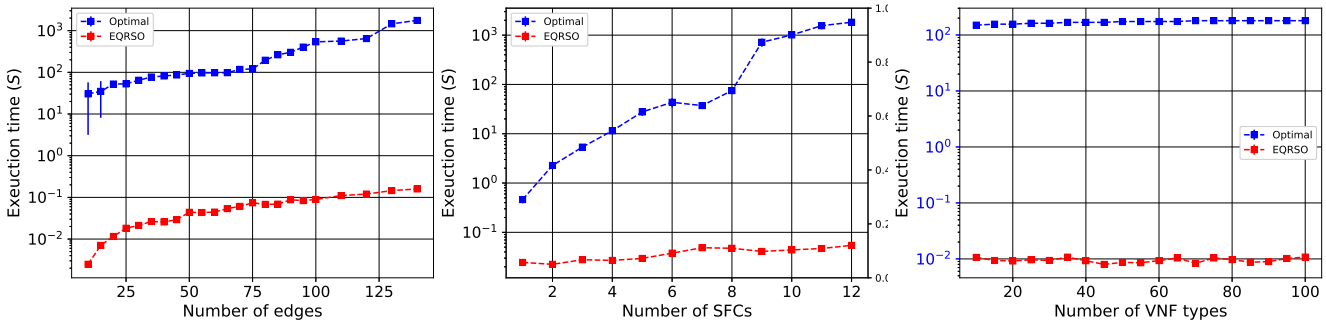
In case that the already scheduled VNFIs cannot serve the VNF v, then a new VNFI should be instantiated and scheduled (Algorithm 1: Lines $22-30$). The Algorithm calls the function $\eta(\Theta, \theta)$ that selects the adequate cloud/edge c_2 and the flavor u_f. This function $\eta(\Theta, \theta)$ leverages the inequalities (50), (51) and (52) and ensures the security level. While the cloud/edge c_2 is selected to ensure better propagation delay and security level, the flavor is selected to ensure better computational delay. Then, the new VNFI u will be assigned to the VNF v and added to $\hat{\mathcal{V}}$ (Algorithm 1: Lines $23-29$). When all the VNF of an SFC θ has been scheduled, the Algorithm includes it to Θ to be considered when scheduling the remaining SFCs.

VII. Performance evaluation

In this section, we evaluate the performances of our solutions. While the heuristic EQRSO has been developed using Python language, the optimal solution uses Python language and Gurobi Optimizer software. All the execution time measurements are based on a Dual Intel Xeon E5-2680 v3 @ 2.5 GHz, with 256 GB of RAM, and running Ubuntu 16.04. In order to ensure the life cycle management (LCM) of different slices, we ran the simulation in 20 epochs, in each of which a set of SFCs arrive and leave the environment. When a SFC leaves the simulation, then all its VNFs will not considered in the next epoch. As aforementioned, the communication link between clouds, VNFs, users, and devices, are characterized by different security levels. While the links within the same cloud have higher security levels, those that do not use any security mechanisms (i.e., IPSEC and SSL) and interconnect clouds, users, and devices have lower security levels. Usually, the use of security mechanisms comes with an inevitable cost regarding the degradation of network bandwidth and end-to-end delay. According to the cryptography type and network layer, the security mechanism is used, we may observe different security levels. In all the simulations, we have considered 15 security levels, where we have uniformly distributed them among the connection links that interconnect clouds, edges and IoT domains. Moreover, we have also considered 15 possible end-to-end security levels, $\xi_{\theta}^{\mathcal{S}}$, that are randomly distributed among different SFCs θ. The suggested solution hosts the VNFs of new arrival SFCs in either existing VNFIs or by instantiating new ones if needed. We have evaluated the suggested solution in different scenarios by running 35 repetitions. The plotted results present the mean and 95% confidence interval. In the evaluation, we have considered the following metrics:

- **LCM cost:** is defined as the average cost needed for deploying VNFIs in public clouds for hosting the VNFs of different SFCs during the simulation epochs.
- **Runtime execution:** This metric is defined as the average time needed to execute the solution and provides configuration for each epoch of the LCM. In fact, at each epoch, we measure the time difference between the finishing and staring time needed for providing the configuration LCM of SFCs.
- **End-to-end delay:** is defined as the average end-to-end delay, which includes the propagation and processing delays, perceived by different SFCs in different epochs.
- **Number of deployed VNFIs:** is defined as the total number of deployed VNFIs in all the periods for serving different SFCs.

Our proposed solutions are evaluated by varying the number of edges, the number of deployed SFCs, and the number of VNF types. In the first scenario, we have varied the number of edges/clouds while fixing the number of SFCs by 6 and the VNF types by 15. Meanwhile, in the second scenario, we have varied the number of SFCs while fixing the number of clouds/edges by 20 and the VNF types by 15. Finally, in the last scenario, we have the number of VNF types while fixing the number of edges/clouds by 20 and the number of SFCs also by 6.



(a) Varying the numbers of clouds (b) Varying the numbers of SFCs (c) Varying the numbers of VNF types

Figure 4: Performances of the proposed solutions in terms of execution time

A. Cost performance evaluation

In this subsection, we show the performances of the suggested solutions in terms of cost while varying the number of edges/clouds, the number of SFCs and the number of types as depicted in Fig. 3. While the optimal solution is shown with a blue curve, the heuristic *EQRSO* is presented with a red color curve.

Fig. 3(a) shows the impact of the number of clouds/edges on the suggested solutions in terms of cost. The first observation that we can draw from this figure is that the number of edges/clouds has a positive impact on the cost. This can be explained as follow, increasing the number of edges/clouds gives more flexibility to the solutions for finding a better position for deploying the VNFIs that serve more VNFs, and hence serving more SFCs. This leads to reducing the number of VNFIs should be deployed, and hence reduce the cost. Moreover, we observe that the optimal solution has almost twice times better performances comparing to the heuristic *EQRSO*. The obtained results show the efficiency of the optimal solution in terms of cost when varying the number of edges.

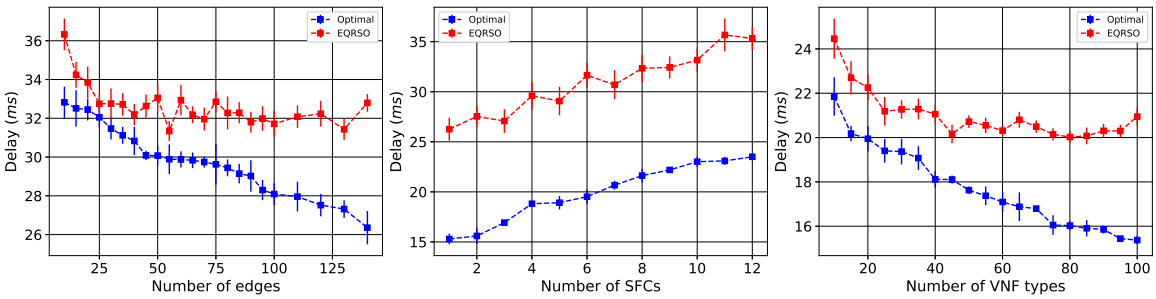
Meanwhile, Fig. 3(b) shows the impact of the number of SFCs should be deployed at each epoch on the cost. From this figure, we observe that the number of SFCs hurts the cost in both solutions. In fact, increasing the number of SFCs leads to an increase in the number of VNFs needed to be deployed, and hence it increases the number of VNFIs that should set up at different clouds/edges. This has a negative impact on the cost. We also observe from this figure that the optimal solution has twice times better performances comparing to the *EQRSO*.

Last but not the least, Fig. 3(c) shows the impact of the number of VNF types on the cost. We observe from this figure that the number of VNF types harms the cost in both solutions. Increasing the number of VNF types leads to increase the likelihood to picks up VNFs with different types in diverse SFCs, and hence they cannot use the same VNFIs. This leads to an increase in the probability of deploying more VNFIs and hence leads to an increase in the cost. We also observe that at a given threshold the cost stabilizes for the optimal solution. This can be explained as follow, at that threshold the most or all the VNFs would have different types. Indeed, the optimal solution successes to assign many VNFs with the same type into the same VNFI. We also observe that the optimal solution has better performances comparing to the heuristic *EQRSO*.

B. Time complexity

Fig. 4 shows the performances of proposed solutions in terms of time complexity while varying the number of edges/clouds, the number of SFCs and the number of types. Fig. 4(a) shows the impact of the number of clouds/edges on the suggested solutions in terms of time complexity. The first observation that we can draw from this figure is that the number of edges/clouds harms the complexity of the optimal solution. Increasing the number of edges/clouds leads to an increase in the number of variables and constraints in the optimization problem of the suggested solution, and hence more time is required for solving the optimization due to the use of branch and bound technique. In contrast to the optimal solution, the number of clouds/edges has only slightly impact the time complexity of *EQRSO*. Whatever the number of the edges/clouds, the execution time does exceed $10^{-1}sec$.

Additionally, Fig. 4(b) shows the impact of the number of SFCs should be deployed at each epoch on the execution time. From this figure, we observe that the number of SFCs has a negative impact on the time complexity of the optimal solution. In fact, increasing the number of SFCs leads to an increase in the number of variables and constraints in the optimization model, and hence it has a negative impact on the execution time. In contrast, the number of SFCs does not have a high impact on the execution time. These results show the efficiency of the heuristic in terms of time complexity. Last but not the least, Fig. 4(c) shows the impact of the number of VNF types could be considered in different SFCs on the execution time. We observe from this figure that the number of VNF types harms both the execution time for the optimal solution. Increasing the number of VNF types leads to an increase in the number of variables and constraints in the optimization problem of the proposed solution, which hurts the execution time of the suggested solution. In contrast, the number of types does not have any impact on the time complexity of *EQRSO*. The time complexity of *EQRSO* does not exceed $10^{-2}sec$ whatever the number of types.



(a) Varying the numbers of clouds (b) Varying the numbers of SFCs (c) Varying the numbers of VNF types

Figure 5: Performances of the proposed solutions in terms of average delay

C. End-to-end delay

In this subsection, we show the performances of the suggested solutions in terms of end-to-end delay, as shown in Fig. 5. Like the previous subsection, we evaluate the end-to-end delay by varying the number of edges/clouds, the number of SFCs, and the number of VNF types. The first observation that we can draw from Fig. 5(a) is the increase in the number of edges/clouds has a positive impact on the delay on *EQRSO* and the optimal solutions. The delay proportionally decreases with the number of edges/clouds in the network. Increasing the number of edges/clouds leads to increasing the likelihood of finding better locations for deploying different VNFIs that could offer better end-to-end delay for different SFCs. We observe that the optimal solution overcomes the heuristic *EQRSO* in terms of end-to-end delay. The difference between the two solutions sometimes exceeds $6ms$.

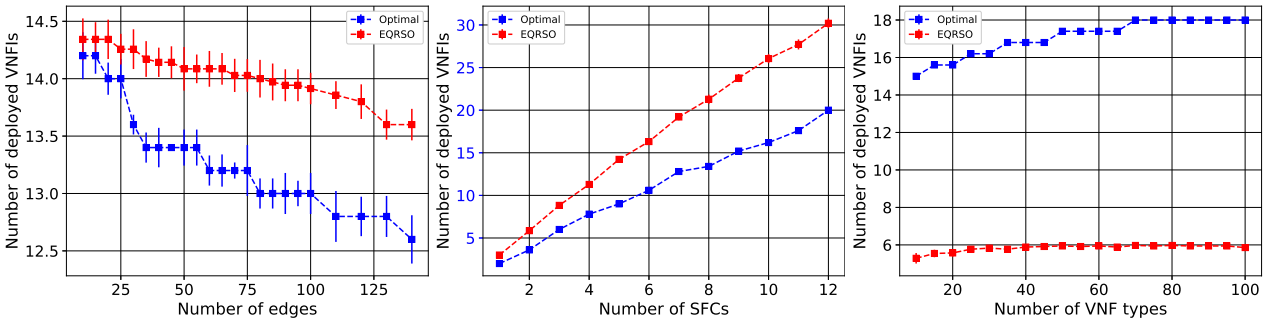
Meanwhile, Fig. 5(b) shows the impact of the number of SFCs on the end-to-end delay. We observe from this figure that the increase in the number of SFCs harms the end-to-end delay for both solutions. This can be explained as follow: Increasing the number of SFCs leads to increase the probability for getting VNFs that have more resources' requirements. The resources demanding of these VNFs will limit their options to be deployed in clouds/edges that offer better delays. Also, increasing the number of SFCs leads to raising the likelihood to get SFCs with sparsely distributed users, which hurts the end-to-end delay. We observe also that the optimal solution has better performances comparing to the heuristic *EQRSO*. The optimal solution overcomes the heuristic *EQRSO* in terms of end-to-end delay by more than $10ms$.

Finally, Fig. 5(c) depicts the impact of VNF types on the delay on both solutions. The first observation that we can draw from this figure is the number of VNF types has a positive impact on the end-to-end delay. In the simulation, we randomly assign types (e.g., firewall and load balancer) to the VNFs of SFCs from a pool named Π. Increasing the size of the pool Π will increase the likelihood that two VNFs that belong to two different SFC do not have the same type. This increases the number of VNFIs used to host the VNFs, which hurts the cost, as depicted in Fig. 3(c). However, from another side has a positive impact on the delay. Increasing the number of VNF types reduces the VNF per VNFI ration, which increases the flexibility of deploying VNFIs in better positions that reduce the delay at each SFC. Let υ_1 and υ_2 two VNFs belong into two different SFC θ_1 and θ_2, respectively. If these two VNFs have the same type and use the same VNFI, then the best location to deploy that VNFI is the one that reduces the delay between each VNF and its successor and predecessor in each chain. Thus, the placement of that VNFI considers four delays; Two delays (i.e., successor and predecessor) per VNF. However, if these two VNFs have different types, they should be assigned to two VNFIs. In this case, the best position of each VNFI is the one that reduces only two delays (i.e., successor and predecessor) of its VNF. This will increase the likelihood of finding better positions that reduce the delay between the VNF and its successor and predecessor, which positively impacts the end-to-end delay. Moreover, the reduction of VNF per VNFI ratio would positively impact the processing delay, which leads to reduce further the end-to-end delay. Also, we observe that the optimal solution has better end-to-end delay comparing to the heuristic *EQRSO*. The optimal solution overcomes the heuristic *EQRSO* by $6ms$ in terms of end-to-end delay.

D. VNFIs distribution

In this subsection, we assist the performances of the suggested solutions in terms of the number of deployed VNFIs while varying the number of edges/clouds, the number of SFCs and the number of VNF types. Fig. 6 shows the impact of the number of edges, the number of SFCs and the number of VNF types on the number of deployed VNFIs. Fig. 6(a) depicts the impact of the number of edges/clouds on the number of deployed VNFIs. The first observation that we can draw from this figure is that the optimal solution outperforms the heuristic *EQRSO* in terms of number of deployed VNFIs. Also, we observe that the number of edges/clouds has a positive impact on the number of deployed VNFIs for both solutions. This can be explained as follows: Increasing the number of edges/clouds leads to increase the possibility to get locations (i.e., clouds) that satisfy many SFCs, hence reduces the number of needed VNFIs to be deployed.

Meanwhile, Fig. 6(b) shows the impact of the number of SFCs on the number of deployed VNFIs. The first observation



(a) Varying the numbers of clouds (b) Varying the numbers of SFCs (c) Varying the numbers of VNF types

Figure 6: Performances of the proposed solutions in terms of average number of deployed VNFIs

that we can draw from this figure is that the number of SFCs harms the number of deployed VNFIs for both solutions. Increasing the number of SFCs leads to an increase in the number of VNFs in the network, and hence we need more VNFIs to be deployed for serving these VNFs. We also observe that the optimal solution has better performances comparing to the heuristic *EQRSO*. The gap between the two solution increases to reach 10 VNFIs. Finally, Fig. 6(c) depicts the impact of the number of VNF types on the number of deployed VNFIs. We observe from this figure that the increase in the number of VNF types hurts the number of deployed VNFIs. Increasing the number of VNF types leads to increase the likelihood to get VNFs that do not have the same type, and hence they cannot use the same VNFI.

VIII. Conclusion

This paper has presented a cost-efficient ZSM orchestration model, aimed to optimize the allocation and scheduling of network services and efficiently deal with the life cycle management (LCM) of different SFC. The proposed solutions consider different aspects during the LCM of different SFCs, encompassing QoS (including end-to-end delay, bandwidth, jitters), actual capacities of VNFs in terms of resources (CPU, RAM, and storage) and current network security levels. The paper has extensively evaluated the efficiently of the proposed solutions and their performances in terms of deploying cost, time complexity, end-to-end delay, and VNF instances distribution, varying the numbers of SFCs, edges/clouds available and VNF types, as demanded in softwarized and complex virtualized IoT scenarios (up to 125 clouds/edges, 100 VNF types). The obtained results show the benefits brought by the proposed solutions to efficiently schedule the network services as the number available edges/clouds increases, reducing overall end-to-end delay, minimizing the number of VNF instances to be deployed, and optimizing the average cost. The obtained results demonstrate the efficiency of the heuristic for achieving good performances in a reasonable time. As future work, we envisage leveraging our orchestration model to dynamically orchestrate network services, even in more complex scenarios, when considering mobile edges nodes (MEC) (e.g. drones as MEC), in which additional constrains and contextual information are need to be considered for cost-efficient zero-touch SFC across networks edges.

Acknowledgment

This work was partially supported by the European research project H2020 ANASTACIA GA 731558, MonB5G GA 871780 and INSPIRE-5Gplus GA 871808. It has been also partially funded by AXA Postdoctoral Scholarship awarded by the AXA Research Fund (Cyber-SecIoT project). This work was partially supported by the Academy of Finland 6Genesis project under Grant No. 318927, and by the Academy of Finland CSN project under Grant No. 311654.

References

[1] ETSI GS ZSM 002, "Zero-touch Network and Service Management (ZSM), Reference Architecture," ETSI, France, Specification V1.1.1, 2019.

[2] N. Huin, B. Jaumard, and F. Giroire, "Optimization of network service chain provisioning," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[3] "Anastacia framework demo," https://www.youtube.com/watch?v=eEQNAcGiMFE, accessed: 2020-03-08.

[4] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. Santos, and C. E. Rothenberg, "Network service orchestration: A survey," *Computer Communications*, 2019.

[5] L. M. Vaquero, F. Cuadrado, Y. Elkhatib, J. Bernal-Bernabe, S. N. Srirama, and M. F. Zhani, "Research challenges in nextgen service orchestration," *Future Generation Computer Systems*, vol. 90, pp. 20–38, 2019.

[6] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín, "Virtual network function scheduling: Concept and challenges," in *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, June 2014, pp. 1–5.

[7] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–9.

[8] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354.

[9] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[10] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 98–106.

[11] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in iaas datacenters via a cooperative game approach," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 873–886, 2016.

[12] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, 2016.

[13] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 554–568, Sep. 2017.

[14] D. Liao, Y. Wu, Z. Wu, Z. Zhu, W. Zhang, G. Sun, and V. Chang, "Ai-based software-defined virtual network function scheduling with delay optimization," *Cluster Computing*, vol. 22, no. 6, pp. 13 897–13 909, 2019.

[15] D. Zhao, J. Ren, R. Lin, S. Xu, and V. Chang, "On orchestrating service function chains in 5g mobile network," *IEEE Access*, vol. 7, pp. 39 402–39 416, 2019.

[16] A. Molina Zarca, J. B. Bernabe, R. Trapero, D. Rivera, J. Villalobos, A. Skarmeta, S. Bianchi, A. Zafeiropoulos, and P. Gouvas, "Security management architecture for nfv/sdn-aware iot systems," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8005–8020, Oct 2019.

[17] A. Molina Zarca, J. Bernal Bernabe, I. Farris, Y. Khettab, T. Taleb, and A. Skarmeta, "Enhancing iot security through network softwarization and virtual security appliances," *International Journal of Network Management*, vol. 28, no. 5, p. e2038, 2018, e2038 nem.2038. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2038

[18] M. Z. Alejandro, D. Garcia-Carrillo, J. Bernal Bernabe, J. Ortiz, R. Marin-Perez, and A. Skarmeta, "Enabling virtual aaa management in sdn-based iot networks †," *Sensors*, vol. 19, no. 2, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/2/295

[19] A. M. Zarca, J. B. Bernabe, A. Skarmeta, and J. M. A. Calero, "Virtual iot honeynets to mitigate cyberattacks in sdn/nfv-enabled iot networks," *IEEE Journal on Selected Areas in Communications*, vol. In press, 2020.

[20] "D3.5: Final security orchestratorreport," http://www.anastacia-h2020.eu/deliverables/ANASTACIA-WP3-AALTO-T3.2-D3.5-FinalSecurityOrchestratorReport-v1.8.pdf, accessed: 2020-03-15.

[21] A. M. Zarca, M. Bagaa, J. B. Bernabé, T. Taleb, and A. F. Skarmeta, "Semantic-aware security orchestration in sdn/nfv-enabled iot systems," *Sensors*, vol. 20, no. 13, p. 3622, 2020. [Online]. Available: https://doi.org/10.3390/s20133622

[22] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[23] M. D. McCool, A. D. Robison, and J. Reinders, *Structured parallel programming: patterns for efficient computation*, 1st ed. Elsevier, 2012.