D3.1: Optimisation of Micro-services Placement and Chaining for Low Latency Services

1 Hichem Magnouche
2 Guillaume Doyen
1 Caroline Prodhon

1 LIST3N, University of Technology of Troyes, Troyes, France, {first.last}@utt.fr
2 SOTERN - IRISA (UMR CNRS 6074), IMT Atlantique, Rennes, France, guillaume.doyen@imt-atlantique.fr

Abstract

This document addresses the major challenges of modern networks in terms of flexibility, scalability, and, above all, low latency in the era of network function virtualization and the emergence of low latency applications. It presents an optimization model for the placement and chaining of micro-services, particularly in the context of low latency Service Function Chains (LL SFCs). This model follows technological considerations of the MOSAICO project regarding packet forwarding strategy, adopting the L4S approach. The document also explores cohabitation strategies for LL and BE (Best Effort) SFCs, proposes a heuristic approach as a fast and realistic alternative solution, and presents specific use cases illustrating practical applications of these concepts. Finally, it analyzes the performance and efficiency of these models through various evaluation scenarios, offering valuable insights for future developments in this field.

Contents

Chapter 1

Introduction

In the age of digitization and rapid IT evolution, modern networks face major challenges in terms of flexibility, scalability and, above all, low latency. Part of the answer to these challenges lies in optimal placement and micro-service chain management, particularly in the context of low-latency (LL) Service Function Chains (SFCs). While these concepts are essential for ensuring optimal network performance, their practical implementation requires a systematic, well-designed approach. This delivrable of the MOSAICO project delves into this issue, presenting an optimisation model for micro-service placement and chaining, as well as an in-depth exploration of cohabitation strategies for LL and BE SFCs which is a straight follow-up of technological consideration of the project regarding the packet forwarding strategy which considers the L4S [1] approach. In addition, a heuristic approach is proposed, offering an alternative, rapid and realistic solution to the problem. We also present specific use cases for placement and the SFC chain within the framework of the MOSAICO project, illustrating practical applications of these concepts. Through a series of evaluation scenarios, we analyze the performance and efficiency of these models, offering valuable insights for future developments in this field.

Chapter 2

Related Work

The rapid transformation of network architectures and the emergence of new performance requirements have led to a significant evolution in approaches to managing virtual network functions (VNFs). These developments have been marked by the growing adoption VNF parallelization and the micro-services approach among others. Each of these architectural optimisations aims to optimize network latency, flexibility and efficiency. In this section, we explore the relevant previous work that laid the foundations for these advances. We will look in detail at the different strategies adopted for VNF placement and routing, the exact and approximate methods used to solve these problems, and how parallelization and micro-services have been integrated to further improve performance.

2.1 VNF Placement and Routing

2.1.1 Exact resolution

Placement and Routing of VNF (VNF-PR) is a problem that has attracted a lot of attention for a decade [2]. It covers two objectives which are: (1) the placement of VNF on network nodes, and (2) their chaining to satisfy the service requests while respecting various associated constraints. In [3], Sun *et al.* assess the richness of the field by leveraging a taxonomy which classifies existing approaches according to the type of orchestration problem, the objective function and finally the resolution method. In most of the related literature, the formulation of the VNF-PR problem is achieved through an Integer Linear Program (ILP) or a Mixed ILP (MILP). Concerning the resolution of the problem, some proposals use a solver such as CPLEX [4, 5], while some others develop a heuristic algorithm [6, 7] or a meta-heuristic [8]. In [4], [9] the authors consider the placement and routing problem of VNF with only one type of VNF for all services thus making it restricted.

In [10], Luizelli *et al.* deal with the VNF-PR issue by proposing a realistic ILP model. In this work, the latency minimization is translated into a constraint instead of the objective function which is rather formulated with the aim of minimizing the number of VNF instances. The authors break down the problem into sub-problems and develop a Variable Neighborhood Search algorithm that tries to find a near-optimal solution for each sub-problem. Similarly, in [11], the authors keep the same model but they use a meta-heuristic algorithm which is more time-efficient. In [12], the authors minimize the number of activated nodes instead of VNF instances, which shows the relevance of the cost metric choice in the model formulation. Finally, in [13], Askari *et al.* propose an algorithm which deploys SFC in a metropolitan network context with the objective of minimizing the number of nodes used. Unlike previous papers, the authors propose an algorithm that allows deployment in a dynamic rather than static environment.

2.1.2 Approximate resolution

The VNF placement and chaining problem is mainly tackled by two strategies. Exact methods, like the Simplex algorithm [14], offer precise results but require long computation times. By contrast, approximate methods, based on heuristics or metaheuristics, offer near-optimal quicker solutions. Despite potential sub-optimality, the faster results of approximate methods make them often preferred in practice.

Existing heuristics for VNF placement and chaining often break down the problem into more manageable steps. For example, [3] tackles it by modeling the problem as a multi-tiered graph and then utilizing the Viterbi algorithm [15] to deploy VNF. However, these methods face a significant challenge: the dynamic parallelization of VNF, which is unknown before deployment, complicates the creation of a multi-level graph. By contrast, Askari *et al.* present in [13] a shared-use approach. Their heuristic examines each VNF placement of existing instances, chooses the

closest to source and destination, or, if absent, calculates the shortest path and deploys a new one. This method may however lead to non-optimal solutions, increasing the traffic latency.

Besides, most studies propose a per SFC approach to deploy VNF on the shortest path. Notably, [16, 17] follow this idea by using Dijkstra's algorithm to calculate the shortest path. In cases where the deployment is not possible on some nodes, they propose to recalculate another one and attempt the deployment again. Similarly, Hirewe *et al.* [16] suggest to remove the most overloaded node, while Gadre *et al.* [17] suggest to remove the highest latency arc before recalculating the new shortest path. However, the latter may not necessarily be the next shortest path, standing for a limitation of this approach. Several optimal methods for k shortest path calculation exist too: the modified Dijkstra algorithm, A*, Bellman-Ford-Moore, and Yen or Eppstein algorithms [18, 19]. The latter extends the Dijkstra algorithm and stands out by its lower complexity. It identifies the k shortest paths in a non-negative graph using a priority queue to store them. It first calculates the shortest paths from each node to the destination node using Dijkstra, and then, proposes a new representation of the original graph which facilitates efficient traversal either on the shortest path with zero-cost or via an auxiliary node, with the cost corresponding to the forwarding latency.

2.2 VNF Parallelization

In order to optimise the latency of SFC, some contributions propose to parallelize the execution of VNF. [20] develops a pioneer work with an algorithm allowing to carry out a graph parallelizing the whole of the parallelizable VNF. To that aim, the algorithm is based on a table which states whether or not two network functions are parallelizable, given that the processing carried out on the packets are not related to each other. The authors propose to implement parallelism of VNF for those deployed on the same node as well as different nodes, standing for internal and external parallelism. Nevertheless, the parallelism on different nodes can controversially degrade the SFC latency as it requires to copy and merge packets, whereas within the same node, the VNF can use a shared memory, as proposed by DPDK [21]. Subsequently, among the most relevant proposals, [22] proposes to solely implement internal parallelism of NF with the condition that all SFC are placed on the same node. In order to limit the copy/merge time, the authors propose two techniques: (1) only copy the header when some processing is performed on it, and (2) use the shared memory when the processed packet fields are different. In a similar way to the former [20, 22] also proposes a parallelization allowing to state which VNF can be parallelized. In [23] the authors propose a parallelization approach that addresses the weaknesses of [20] and [22] by parallelizing VNF placed on the same node without any condition on the whole SFC thus preventing the penalty latency in copy/merge on different nodes. Besides, it also allows better agility since the parallelism is decided after deployment. Finally in [24] the authors present an approach that manages the internal and external parallelism of NF to minimize the latency of deployed SFC. The authors present a two-step approach to (i) compute the set of possible parallelism configurations for a service chain and select the one that respects the order and parallelism between NF and (ii) choose, according to the infrastructure configuration, the best deployable one. Some tests concluded that full parallelism is not always optimal in terms of latency while partial parallelism allows a latency gain of up to 25% as compared to full deployment.

2.3 Micro-services: Architectures and Orchestration

For a couple of years, the benefits of micro-services have been demonstrated [25] as an alternative to standard monolithic VNF which exhibit three important limits: the overlapping of functionalities, the loss of CPU cycles and the lack of flexibility in scaling. Following this idea, several micro-services architectures have been designed and implemented, such as Microboxes [26] and Openbox [27], all bringing specific performance enhancements which mainly leverage lightweight virtualization with containers and zero-copy of packets with DPDK[21]. In [28], the authors study the impact of deploying micro-services on one or more containers, and they observe a slight increase in latency in the latter situation due to the packet flow between containers. In [29], the authors present the advantages and disadvantages of using micro-services and they assess their interest in terms of execution latency as compared to the monolithic approach by deploying micro-services on different CPUs. The authors propose an AI-based architecture to decide the reuse, creation or duplication of micro-services during deployment.

Concerning the placement and chaining of micro-services, in [30], the authors address the intra- and inter-server connectivity and their impact on the communication between micro-services that should be well-performing to minimize end-to-end latency. They propose an ILP placement model whose objective function is to minimize the latency delay between the different micro-services. They also propose some constraints specific to the end-to-end latency as well as the NFV infrastructure. In [31], the authors also develop a placement solution that limits end-to-end latency and minimizes the exchange of messages between VNF. In order to reduce latency, the authors focus

on minimizing the communication delay between the different VNF and underlying micro-services and therefore, they create network micro-service bundles called Affinity Aggregates which are a set of VNF or micro-services exhibiting heavy communication and should therefore be deployed in a close way. [32] exploits the advantages of micro-services (re-usability, light weightiness, and better scaling) to overcome the performance degradation that NFV can generate with monolithic NF. It proposes *MicroNF*, a framework based on three axes: (1) a reconstruction of SFC to re-factor micro-services when possible; (2) a micro-services placement which tries to consolidate them on the same node; and (3) a scaling approach that attempts to balance the load between the network nodes to avoid duplicating micro-service instances and thus limit communications.

Mutualization consists in grouping two or more identical micro-services belonging to the same SFC in a single one. It makes it possible, first, to shorten the length of the SFC and consequently its latency, and second to save memory and CPU resources. Such an approach has been studied in [32, 33] as part of the functional decomposition of VNFs into micro-services. These research works particularly show that the mutualization of two identical micro-services is not systematic. Indeed, it requires checking the overall set of micro-services from the original SFC to be sure that the processing of packet data will not be affected. This is achieved thanks to a mutualization table as proposed by [32] which focuses on this particular enhancement. Besides, [33] also proposes a framework supporting micro-services that includes an algorithm allowing the mutualization of micro-services. However, it does not propose any model nor algorithm for the placement and chaining of micro-services.

Through this survey of related literature, we observe that a consequent work has been carried out in order to minimize the latency of SFC, whether it relies on the routing and placement modeling [4, 9, 10, 11, 12, 2], VNF parallelism [20, 22, 23] or micro-services decomposition and mutualization [30, 25, 33]. However, these different contributions have been considered in an isolated way, which can lead to sub-optimal latency. Indeed, as highlighted in [32], the micro-services approach, implemented without mutualization, exhibits a negative impact on the overall SFC latency. Similarly, parallelism, when used in a non-optimal way, can increase latency instead of reducing it [23], and to the best of our knowledge, it has only been considered for monolithic VNF. Therefore, as a first contribution of this paper, we propose a comprehensive optimization model that specifically provides a robust and flexible micro-services orchestration using both mutualization and optimized parallelism based on the infrastructure configuration.

Chapter 3

Optimisation Models

In this section, we present the micro-services placement and chaining orchestration problem we address and then, we develop its mathematical formulation. Wishing to optimise the SFC latency and benefit from the agility offered by micro-services, we propose an approach allowing to: (1) mutualize micro-services through a pre-processing algorithm; (2) place and chain micro-services through a MILP that also (3) efficiently manages parallelization of micro-services according to the infrastructure setup (i.e. number of nodes and links as well as their available resources).

3.1 Problem Statement

Definition: The micro-services placement and routing problem we study is defined on a network graph $G = (N, L)$, where N is a set of nodes and L a set of links between nodes. Q is a set of SFC requests, with each request $q \in Q$ being characterized by a source S_q, a destination D_q, a nominal bandwidth B_q statistically representative for request q, a maximum execution latency Λ_q and a set of micro-services of different types to be traversed by an edge flow. The micro-services placement and chaining optimization problem consists in finding:

- The placement of micro-services over network nodes;
- The assignment of requests to micro-services already placed;
- The chaining for each request,

subject to:

- Memory node capacity constraints;
- Micro-services forwarding and execution latency constraints;
- Parallelism execution constraints.

3.1.1 Hypothesis

The optimization objective chosen in our work is, for each SFC to deploy, the minimization of the gap between the actual latency as provided by the placement and routing solution, and that given in the SFC specification. To provide an appropriate solution, we formulate two classes of hypotheses:

3.1.2 Essential hypotheses for the model

h1. The available nodes have no usage cost but the available resources expressed in terms of CPU and memory are limited;

h2. The available links have no usage cost but the flow rate on each one is limited;

h3. The processing time of mutualized micro-services is negligible in comparison to that of non-mutualized micro-services, as the time overhead from mutualization is considered insignificant due to identical executed code in both situations.

3.1.3 Releasable hypotheses without loss of generality

h4. The memory required for the deployment of micro-services is similar for all micro-services;

h5. Sharing the usage of micro-services between different SFC is not allowed;

h6. A shared micro-service occupies the same memory space as a non-mutualized micro-service. In our approach, we assume that the size of micro-services code is much larger than their configuration, so merging the configuration of mutualized micro-services does not impact their memory occupation.

3.2 Overall Approach

Our approach is developed with the aim of taking advantage of the micro-service features in order to reduce the end-to-end latency of SFC. It consists in two parts: (1) a pre-processing algorithm that performs the mutualization, a pre-parallelization on the set of SFC, and provides the Q set and parallelism parameters to be used in our model, and (2) a MILP translating the problem of placement and chaining of micro-services, which is solved by taking into consideration the parallelism parameters among other constraints.

3.2.1 SFC Pre-processing

The pre-processing phase needs two-dimensional parallelism and mutualization tables as inputs, which indicate whether two micro-services are mutualizable or parallelizable as developed in the literature [32, 23]. To understand the pre-processing, let us consider the example depicted in Figure 3.1, where a request q is made up of an original SCF containing five sequential micro-services. Supposing that the tables indicate that micro-services 1 and 5 can be mutualized and micro-services 2, 3 and 4 can be parallelized, our algorithm builds a new SFC made up of four micro-services as depicted in the second part of Figure 3.1. Note that depending on the infrastructure constraints, the MILP of the second phase may not necessarily lead to the placement and execution of micro-services 2, 3 and 4 in parallel, even if this was allowed in the pre-processing phase.

3.2.2 Managing Parallelized Micro-services

The pre-processed SFC are provided to the model with some parallelism parameters to carry out a placement and chaining. The aim here is to minimize the sum of the gaps between the required and achieved latency after deployment, for each SFC while taking into account the infrastructure setup. The key-point stands in the choice of service parallelization which is managed by the model through the generation of forks and mergers of two types:

- internal forking: a packet is duplicated within a node to reach a set of further parallelized micro-services located on the same node. This kind of internal parallelism does not require copying data, nor merging, since, as reviewed in sub-section 2.2, technologies such as DPDK allow shared memory to be used.
- external forking: a packet outgoing from one node is duplicated to subsequent NF located on two or more successor nodes. Such external parallelism implies a copy of the packets to be sent to the different micro-services deployed on different nodes and then a merging of the resulting packets too.

The latter mechanism leads us to add a latency cost for the external fork corresponding to the copy and merge time, unlike the internal fork. The originality of the model lies then in the fact that depending on its configuration, it decides which micro-services to parallelize internally, externally or not at all. However, this complicates the computation of the induced latency. Indeed, when the model defines a placement, it also has to indicate whether the micro-services on the nodes (if parallelizable) are visited consecutively or not. In order to know which micro-services are running in parallel within a node, we use groups that gather them together.

To illustrate this principle, the example in Figure 3.2 shows three different possible deployments for request q, as introduced in Figure 3.1. The first one (A) does not parallelize any micro-services: although two are on the same node (i.e. micro-services 2 and 3), they do not belong to the same group. In this case, the resulting SFC is composed of a single group (g_1) on nodes B and D, and two groups (g_1 and g_2) on node E. This situation is equivalent to a deployment without parallelism as presented in [10, 11, 12]. The second deployment (B) proposes an external parallelization of micro-services 2 and 3 on two different nodes, thus a single group (g_1) per node. This induces an additional latency corresponding to the copying and merging operations. Finally, deployment (C) parallelizes micro-services 2, 3 and 4 by performing internal and external parallelism. This situation is equivalent to the deployment scenarios presented in [22]. This time, a single group (g_1) is assigned to each of nodes B, C and D. In this case also, a fork cost is counted because the latter is external. Beyond, this SFC example and underlying infrastructure may lead to other possibilities, actually depending on the memory capacities of each node.

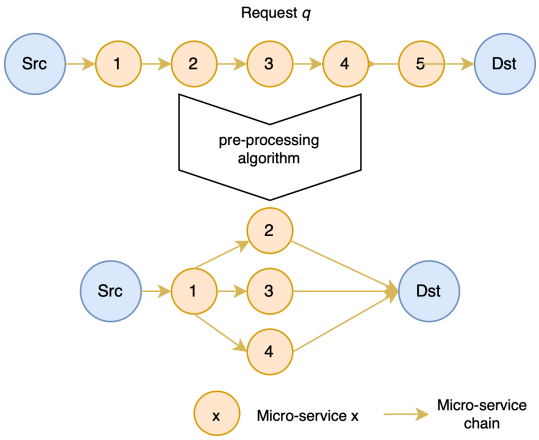


Figure 3.1: Pre-processing example

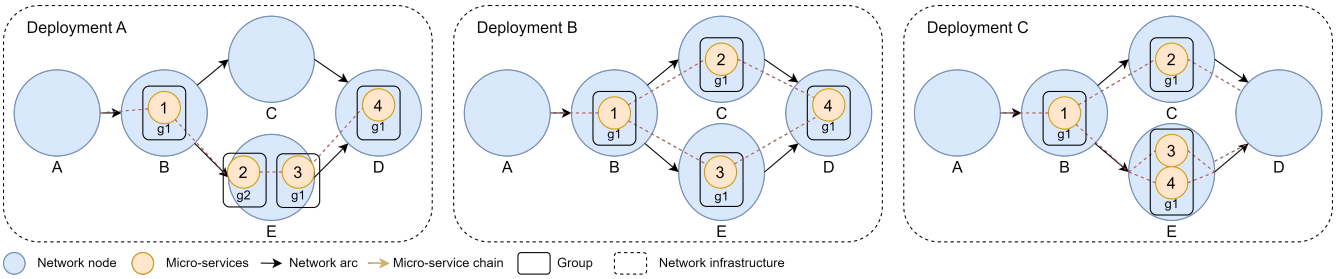


Figure 3.2: A deployment example

3.3 Mathematical Formulation

Table 3.1 provides the set of notations used for the formulation of our MILP formalizing the problem of placement and chaining of micro-services.

In this model, we define seven decision variables. x_{nfq} are used to decide which micro-service $f \in F$ of request $q \in Q$ should be placed on which node $n \in N$. y_{nfq} allow to know whether micro-service $f \in F$ is placed on the nodes preceding $n \in N$ (n included) on the path related to request $q \in Q$. Variables $a_{n_1 n_2 q}$ are used to decide whether the link between nodes n_1 and n_2 from set N is used for service request $q \in Q$, all links being oriented in this model. Having allowed the forks, we introduce variables $l_{n_1 n_2 q h}$ to decide whether the link between nodes n_1 and n_2 from set N is used within path $h \in F_q$ of the chaining of request $q \in Q$. This allows to define the set of possible paths for the chaining of each SFC. Variables bf_{nqh} indicate the existence of a fork on node $n \in N$ relative to a request $q \in Q$ and path $h \in F_q$. Then, the latency of request $q \in Q$ finishing its process at node $n \in N$ is equal to the greatest latency of the different relative paths. This requires to define a longer path, managing the notion of group as introduced in section 3.2.2. A group $g \in F_q$ is composed of a set of a single or several parallelized micro-services assigned to node $n \in N$ for request $q \in Q$ and for path $h \in F_q$. Since all the micro-services belonging to the same group run in parallel, we use variable γ_{nqfg} to indicate whether micro-service $f \in F$ related to request $q \in Q$ is placed in group $g \in F_q$ attached to node $n \in N$, and variable σ_{nqgh} which corresponds to the latency of group $g \in F_q$ relative to request $q \in Q$ and path $h \in F_q$. Finally, in order to minimize the overall gap between the required and actual latency, we introduce variable r_q which represents the latency gap for each request $q \in Q$.

This leads to the following model, consisting, for each SFC, in deploying the minimization of the gap between the actual latency as provided by the placement and routing solution, and that given in the SFC specification, subject to the two dozen of constraints that we describe subsequently. The objective function (3.1) represents in its first term the minimization of the sum of the latencies delay, r_q on the set SFC $q \in Q$. The second term represents the minimization of the decision variables y_{nfq} and bf_{nqh} in order to facilitate the understanding of the result. For example, if y_{nfq} is not constrained, it may hold a random value which may induce a wrong interpretation of the result. Also the variable p_{nfq} is an intermediate variable allowing to facilitate the formulation of the model.

$$\min \sum_{q \in Q} r_q + \sum_{n \in N} \sum_{q \in Q} \sum_{f \in F_q} \sum_{h \in F_q} y_{nfq} + bf_{nqh} + p_{nfq} \tag{3.1}$$

Constraints (3.2) and (3.3) guarantee that if there is an incoming flow in a node for a certain service request q, and the node is neither a source nor a destination node, an outgoing flow must exist and vice-versa. Unlike a

Sets	
N	Nodes
L in $N \times N$	Links
Q	Service requests
F	Micro-services
Micro-service parameters	
λ_f	Execution latency
Rm_f	CPU resource requirements
Demand parameters	
S_q	Source
D_q	Destination
Λ_q	Latency
B_q	Throughput
F_q in Q	Micro-services composition
Infrastructure parameters	
M_n	Node memory
R_n	Node CPU resource
$\Delta_{n_1 n_2}$	Link latency
$Da_{n_1 n_2}$	Link flow rate
CB	Bifurcation cost
Parallelism parameters	
$T_{f_1 f_2}$	Parallelism possibility
$P_{f_1 f_2 q}$	Parallelism possibility into service request
Model parameters	
m	Constant value guaranteeing compliance with constraints
Binary variables	
x_{nfq}	$= 1$ if function f is placed on node n for request q
y_{nfq}	$= 1$ if function f is placed on node n or before for request q
$a_{n_1 n_2 q}$	$= 1$ if link (n_1, n_2) is activated for request q
$l_{n_1 n_2 q h}$	$= 1$ if link (n_1, n_2) is activated for request q for path h
γ_{nqgf}	$= 1$ if function f related to request q placed on node n belongs to group g
bf_{nqh}	$= 1$ if there is a fork on node n for request q and for path h
p_{nfq}	Intermediate variable
Continuous non-negative variables	
r_q	Gaps between the required and achieved latency for LL request q after deployment
σ_{nqgh}	Latency of group g related to request q, node n and path h
o_{nq}	Order of node n in the chaining of request q

Table 3.1: Notation table

standard chaining problem, outgoing flows can be greater or lower than incoming flows and vice-versa, due to the fork-merge managed by the model.

$$\sum_{e \in N} a_{enq} \geq a_{nsq} \qquad \forall q \in Q, n, s \in N \neq S_q, D_q \tag{3.2}$$

$$\sum_{s \in N} a_{nsq} \geq a_{enq} \qquad \forall q \in Q, n, e \in N \neq S_q, D_q \tag{3.3}$$

In order to avoid cycles in the chain and to force it to be elementary, constraints (3.4) ensure that each node has an order in the chain and that this order is respected.

$$o_{n_1 q} \geq o_{n_2 q} + a_{n_2 n_1 q} - m(1 - a_{n_2 n_1 q}) \qquad \forall n_1, n_2 \in N \tag{3.4}$$

The set of constraints (3.5) to (3.10) allow to set variables y_{nfq} to 1 in case function f is placed on node n or before for request q.

$$y_{nfq} \geq x_{nfq} \qquad \forall n \in N, f \in F, q \in Q \tag{3.5}$$

$$a_{n_1 n_2 q} - 1 + x_{n_1 fq} - x_{n_2 fq} \leq y_{n_2 fq} \quad \forall n_1, n_2 \in N, \forall q \in Q, \forall f \in F_q \tag{3.6}$$

$$y_{n_1 fq} - 1 + a_{n_1 n_2 q} \leq y_{n_2 fq} \qquad \forall n_1, n_2 \in N, \forall q \in Q, \forall f \in F_q \tag{3.7}$$

$$\sum_{n_1 \in N} y_{n_1 fq} + a_{n_1 nq} \leq p_{nfq} \forall n \in N, \qquad \forall q \in Q, \forall f \in F_q \tag{3.8}$$

$$x_{nfq} \leq p_{nfq} \qquad \forall n \in N, \forall q \in Q, \forall f \in F_q \tag{3.9}$$

$$y_{nfq} \leq p_{nfq} \qquad \forall n \in N, \forall q \in Q, \forall f \in F_q \tag{3.10}$$

More specifically, constraints (3.5) allow to set variables y_{nfq} to 1 if x_{nfq} equal to 1. Constraints (3.6) aim at setting variables $y_{n_2 fq}$ to 1 if the relative function f is placed on node n_1, preceding node n_2 and concerning request q. Constraints (3.7) allow to set variables $y_{n_2 fq}$ to 1 if $y_{n_1 fq}$ equal to 1 and if n_1 is preceding node n_2 concerning request q. The set of constraints (3.8), (3.9) and (3.10) allow to set variables y_{nfq} to 0 if function f is not placed on node n and it is not placed on any of the preceding nodes of n as well as their respective preceding nodes.

Constraints (3.11) ensure that all micro-services related to each request q are placed on the destination node or before on the chain related to q.

$$y_{nfq} \geq y_{D_q fq} \qquad \forall q \in Q, \forall f \in F_q, \forall n \in N \tag{3.11}$$

Constraints (3.12) and (3.13) guarantee that if two micro-services are placed in the same group, they can be processed in parallel, and this in two ways: the first states that the two micro-services can be technically parallelizable, the second states that, in the SFC, the two micro-services are parallelizable, i.e. none of them is in the precedence list of the other.

$$\gamma_{nqgf_1} + \gamma_{nqgf_2} \leq T_{f_1 f_2} + 1 \qquad \forall n \in N, \forall q \in Q, \forall g, f_1, f_2 \in F_q \tag{3.12}$$

$$\gamma_{nqgf_1} + \gamma_{nqgf_2} \leq P_{f_1 f_2 q} + 1 \qquad \forall n \in N, \forall q \in Q, \forall g, f_1, f_2 \in F_q \tag{3.13}$$

Constraints (3.14) ensure that variables σ_{nqgh} are greater than or equal to the execution latency of all the micro-services belonging to group g on path h.

$$\sigma_{nqgh} \geq \lambda_f(\gamma_{nqgf} + l_{nn_2 qh} - 1) \quad \forall q \in Q, f \in F_q, \forall n, n_2 \in N, \forall h, g \in F_q \tag{3.14}$$

Constraints (3.15) allow variables r_q to be equal to the gaps between the required and achieved latency for request q which do not respect their latency specification.

$$\sum_{n_1 \in N} \sum_{n_2 \in N} l_{n_1 n_2 qh} * \Delta_{n_1 n_2} + \sum_{n \in N} \sum_{g \in Fq} \sigma_{nqgh} + \sum_{n \in N} bf_n * CB - \Lambda_q \leq r_q \quad \forall q \in Q, \forall h \in F_q \tag{3.15}$$

Constraints (3.16) and (3.17) respectively ensure that the CPU resources and the available flow rate on each link are respected for each node.

$$\sum_{q \in Q} \sum_{f \in F_q} x_{nfq} * Rm_f \leq R_n \qquad \forall n \in N \tag{3.16}$$

$$\sum_{q \in Q} \sum_{f \in F_q} a_{n_1 n_2 q} * Db_q \leq Da_{n_1 n_2} \qquad \forall n_1, n_2 \in N \tag{3.17}$$

Finally, the model includes the following types of variables.

$$x_{nfq} \in \{0,1\} \quad \forall n \in N, f \in F, q \in Q \tag{3.18}$$
$$y_{nfq} \in \{0,1\} \quad \forall n \in N, f \in F, q \in Q \tag{3.19}$$
$$a_{n_1n_2q} \in \{0,1\} \quad \forall n_1, n_2 \in N, q \in Q \tag{3.20}$$
$$l_{n_1n_2qh} \in \{0,1\} \quad \forall n_1, n_2 \in N, q \in Q, h \in F_q \tag{3.21}$$
$$\gamma_{nqfg} \in \{0,1\} \quad \forall n \in N, q \in Q, f \in F, g \in F_q \tag{3.22}$$
$$bf_{nqh} \in \{0,1\} \quad \forall n \in N, q \in Q, h \in F_q \tag{3.23}$$
$$p_{nfq} \in \{0,1\} \quad \forall n \in N, q \in Q, f \in F \tag{3.24}$$
$$r_q \geq 0 \quad \forall q \in Q \tag{3.25}$$
$$\sigma_{nqgh} \geq 0 \quad n \in N, q \in Q, g \in F_q, h \in F_q \tag{3.26}$$
$$o_{nq} \geq 0 \quad n \in N, q \in Q \tag{3.27}$$

Besides, the model also includes some additional constraints, not detailed here due to space constraints. However, for reproducibility purposes, a comprehensive description of the model, including all constraints, is available in Deliverable D2.3 of the supporting project, at: https://www.mosaico-project.org/outcomes. Briefly, these constraints ensure that: (a) the placement of all the micro-services related to each request is achieved; (b) the chaining passes through the necessary micro-services; (c) the memory capacity of each node is respected; (d) each chaining starts with the source node and ends with the destination node related to each request; (e) variables bf_{nqh} are activated when an external fork takes place on node n for request q and on the path h ; (f) the links related to each path h are only active if they are active in the request chain; (g) the flows for each path are respected; (h) each path related to a request q passes through at least one micro-service related to its request; (i) each deployed micro-service belongs to a group. One can lastly notice that the constraints of nodes ordering respect and micro-services ordering respect were inspired by the work carried out in [4, 34]. This studied problem is NP-complete. Indeed, by considering the particular case with only one type of VNF, made of a single micro-service without parallelism nor mutualization option, it reduces to the classical VNF placement problem that is already NP-Complete [35].

3.4 Toward Fair Sharing Strategies of LL and BE SFC

Although the expansion of LL applications requires dedicated orchestration models such as the one we proposed in the previous section, there are still BE traffic for current classic applications (e.g. mail, web, etc.) and the Internet stands for the backbone infrastructure which must ensure the coexistence of these two classes regarding both their traffic forwarding by network nodes (i.e. switches and routers) and processing by NF in SFC. In this section, we propose three cohabitation strategies for packet processing in SFC, inspired by the L4S approach for packet forwarding in infrastructure-level networks. These strategies differ in their business objectives and prioritize one class over the other using a dedicated Ω parameter, analogous to the L4S's coupling factor k between respective AQM. While we initially identified sixteen cohabitation strategies for LL and BE SFC, only those presented in this paper are relevant for a realistic cohabitation scenario. One can notice that enforcing a cohabitation strategy is only necessary in the case of an overloaded network infrastructure. When resources are sufficient for deploying all LL and BE SFC, the question of cohabitation is irrelevant, as all SFC can be optimally deployed. The cohabitation strategies we propose here extend our model with important features, including (1) taking into account BE SFC with no strict latency constraints, (2) proposing partial deployment solutions when the infrastructure is insufficient for a complete deployment, and (3) ensuring a strict latency compliance for deployed LL SFC in certain situations.

3.4.1 Related work

SFC can be divided into two classes: LL SFC with strict latency requirements, and BE SFC with no strict latency constraints. The cohabitation of these two SFC classes has emerged as an important research area. From an architectural perspective, two approaches have been proposed: the slicing approach and the L4S one. The slicing approach [36] proposes to leverage network isolation to allocate distinct resources to different service classes, thus preventing an overload in BE traffic from impacting LL ones. However, slicing may not allow the most efficient resource usage of a virtualized infrastructure and a fair sharing of these resources between the different classes. The L4S architecture [1] proposes a solution to the packet forwarding operation that ensures a fair sharing of resources while respecting traffic features. In L4S, the respective AQM (Active Queue Management) of LL and BE traffic are coupled to ensure that both classes adapt their sending rate to mitigate congestion, thus preventing the BE traffic

from being starved while the LL one is preserved [37, 38, 39]. The L4S approach contrasts with the state-of-the-art *diffserv* approach [40] which fully isolates traffic classes for the sole forwarding purpose too.

In the field of placement and chaining, several studies address both BE and LL classes of SFC through mathematical formulations or heuristic algorithms. P. Cappanera *et al.* [41] propose a MILP model for the placement and chaining of SFC, taking into account different levels of prioritization. This priority is defined by a variable introduced in the objective function, allowing the model to balance the requirements of various SFC according to their importance. However, the main limitation of this work relies in its unique approach to coexistence, prioritizing the LL SFC deployment without fully considering the potential latency impact on other services. As such, it could benefit from further investigation especially in the trade-offs between different service classes. Following the same idea and limits, F. Behrooz *et al.* [42] address the issue of optimizing the deployment of dynamic SFC by considering the prioritization of different service requests. Some SFC known as *emergency* are treated as a priority and must respect throughput and latency constraints, while BE SFC, with lower priority, use the remaining resources. The authors develop a resolution heuristic based on a combinatorial optimization resource scheduling algorithm, which considers the network capacity and topology constraints to enhance overall performance. Besides, M. Amir et al. [43] address the management of LL and BE service classes in their placement and chaining algorithm, assigning higher priority to LL SFC and treating them preferentially. By giving precedence to LL SFC, their approach aims to ensure the quality of service for latency-sensitive applications. However, the weakness of this approach is that BE SFC are deployed in the remaining infrastructure capacities, potentially resulting in sub-optimal resource allocation and thus augmenting the degraded performance for traffic forwarding and the overall infrastructure efficiency.

3.4.2 Analysis and Selection of Cohabitation Strategies

The business objective supported by the cohabitation of two service classes can be defined by two criteria abstractly named *deployment* and *latency*. The first one specifies whether or not the deployment of all the SFC of a given class is mandatory. The second one is specific to LL and BE classes of traffic. For LL SFC, it determines whether they must strictly respect the required latency, or whether the gap between the required latency and the actual latency must be minimized. For BE SFC, it determines whether their latency must be minimized still, or not, given that BE traffic is not subject to prescribed latency constraints. The combination of these two criteria for the LL and BE classes generates sixteen possible cohabitation strategies. Among them, we exclude a first set which proves to be infeasible or not plausible, as for example, a strategy setting mandatory deployment of all SFC for both classes which is not feasible in an overloaded infrastructure. Similarly, we also eliminate the strategy where LL are fully deployed and latency constraints are strictly respected, since this approach does not act as an actual cohabitation but rather the sole deployment of BE on the remaining space. At the end, only three plausible strategies remain, representing specific needs and contexts, and for each of them we developed a dedicated objective function and modified some constraints from our initial placement and chaining model. Table 3.2 summarises them, in which: (i) BE latency is always minimized, because even if no prescribed latency bound is specified explicitly for this class of SFC, the lower the latency the better the service quality for consumers, and (ii) BE deployment is always partial. Indeed, considering the case of a total deployment of BE SFC, the infrastructure resources may be insufficient, and, degrading a LL service availability through a partial deployment of its SFC while BE SFC are all deployed, does not appear as a plausible scenario in our context.

Strategy	*Parameters*	*LL*	*BE*
Equitable	Latency	Minimise	Minimise
	Deployment	Partial	Partial
LatRes	Latency	Strict	Minimise
	Deployment	Partial	Partial
LLDep	Latency	Minimise	Minimise
	Deployment	Total	Partial

Table 3.2: Cohabitation strategies for LL and BE SFC

3.4.3 MILP Model Evolutions

Having a mandatory deployment of LL or BE SFC, or constraining the latency compliance for LL SFC requires several modifications in our model which are implemented in the objective function and in some constraints. Besides, considering a novel BE SFC class requires some ground modifications which are common to the three strategies we consider. As such, we add the variables and parameters summarized in Table 3.3. Herein, cl_q is a variable related

Demand parameters	
cl_q	Class of SFC : LL or BE
nLL	Number of LL SFC
nBE	Number of BE SFC
SLR	Sum of prescribed latency for LL SFC
$laBE$	Sum of lengths for BE SFC
Binary variables	
br_q	$= 1$ if SFC q is deployed
Continuous non-negative variables	
rb_q	Latency of BE request q

Table 3.3: Notation table for cohabitations strategies

to each SFC q which indicates the class it belongs to (i.e. BE or LL) and variable br_q indicates if SFC q is deployed or not.

1. **Fair Sharing Strategy (Equitable)**
   This strategy places both classes of services on the same level and the Ω parameter acts as the exclusive way to prioritize one class over the other at the placement stage. Unlike the basic model, it does not require fully deployed SFC and it minimises the latency gap of LL and the total latency of BE. Consequently, the objective function is the sum of four components: the number of deployed LL SFC, the number of deployed BE SFC, the latency gap of LL SFC and the latency of BE SFC. These components exhibiting different scales, a normalization is applied.

$$\min(\sum_{q\in Q} r_q/SLR + (nLL - \sum_{q\in Q}(br_q * cl_q))/nLL) * \Omega + (\sum_{q\in Q} rb_q/laBE + (nBE - \sum_{q\in Q}(br_q * (1-cl_q)))/nBE) * (1-\Omega) + \sum_{n\in N}\sum_{q\in Q}\sum_{f\in F_q}\sum_{h\in F_q} y_{nfq} + bf_{nqh} + p_{nfq} \quad (3.28)$$

   The first line of objective function (3.28) represents the latency gap of LL SFC plus the number of LL SFC that are not deployed multiplied by Ω. The second line exhibits the sum of BE SFC latency plus the number of non-deployed BE SFC multiplied by $(1-\Omega)$. Regarding the constraint modifications of our initial model, we only relax here the constraint which forces all LL SFC to be deployed.

2. **Latency Respect Strategy (LatRes)**
   This second strategy ensures that the latency of deployed LL SFC is respected and that the latency of BE SFC is as low as possible, without any restriction on the deployment. Concerning the use of resources, with equal Ω weight, LL SFC will be prioritized because their deployment requires the full respect of latency. This strategy is interesting when the LL SFC do not support any latency penalty such as for industrial control applications. Consequently, the objective function simply maximises the number of deployed LL and BE SFC and it minimises the latency of BE SFC.

$$\min((nLL - \sum_{q\in Q}(br_q * cl_q))/nLL) * \Omega + (\sum_{q\in Q} rb_q/laBE + (nBE - \sum_{q\in Q}(br_q * (1-cl_q)))/nBE) * (1-\Omega) + \sum_{n\in N}\sum_{q\in Q}\sum_{f\in F_q}\sum_{h\in F_q} y_{nfq} + bf_{nqh} + p_{nfq} \quad (3.29)$$

   The first line of objective function (3.29) represents the number of LL SFC that are not deployed multiplied by Ω. The second line is the sum of the latency of the BE SFC plus the number of not deployed BE SFC multiplied by $(1-\Omega)$. This strategy needs to add a new constraint (3.30) which forces all deployed LL SFC to respect the required latency.

$$r_q * cl_q = 0 \qquad \forall q \in Q \quad (3.30)$$

3. **LL Deployment Strategy (LLDep)**

This last strategy guarantees the full deployment of LL SFC, subject to sufficient infrastructure resources. The latency of LL and BE SFC, and the number of deployed BE SFC depend here on the Ω weight which gives priority to the deployment of all LL SFC, without guaranteeing their latency. This corresponds to the case where the deployment of LL SFC is mandatory. The objective function must therefore minimise the latency gap of LL SFC, the number of deployed BE SFC and their latency.

$$\min(\sum_{q\in Q} r_q/slr) * \Omega + (\sum_{q\in Q} rB_q/loBE + (nBE - \sum_{q\in Q}(br_q * (1 - cl_q)))/nBE) * (1-\Omega) + \sum_{n\in N}\sum_{q\in Q}\sum_{f\in F_q}\sum_{h\in F_q} y_{nfq} + bf_{nqh} + p_{nfq} \quad (3.31)$$

The first line of objective function (3.31) represents the latency gap of LL SFC multiplied by Ω. The second line is the sum of the BE SFC latency plus the number of non-deployed BE SFC multiplied by $(1-\Omega)$. The constraint we need to modify from our initial model is that which forces all SFC to be deployed, so that the model only forces LL SFC as provided in Eq. (3.32).

$$r_q * cl_q \leq 0 \qquad \forall q \in Q \quad (3.32)$$

3.5 Mathematical Model Evaluation

In this section, we present and analyse the evaluation results of our initial model of micro-services placement and chaining and we compare it with concurrent approaches from the literature. We also present the evaluation results of our three cohabitation strategies for LL and BE SFC.

3.5.1 Implementation

In order to validate our model and its performance under various conditions, we have implemented it into CPLEX v20.1.0. The C++ code is 1135 lines long and available at: https://www.mosaico-project.org/outcomes for reproducibility. The correctness checking has consisted in verifying the following bias: (1) absence of circuits, (2) deployment of all micro-services related to each request for the initial version of our model, (3) respect of the ordering between micro-services, (4) respect of memory resource consumption, (5) correctness of latency computation, and finally (6) correctness of parallelism and mutualization in relation to the mutualization and parallelism tables. All experiments were performed on an 11th gen. Intel Core i7-1165g7@2.80GHz 1.69GHz computer with 16GB of RAM and Windows 10 Prof. Educ. as the underlying operating system.

3.5.2 Evaluation Scenarios

We have considered two distinct scenarios whose purpose is to evaluate the performance of (1) our initial orchestration model for LL SFC leveraging micro-services and (2) the different cohabitation strategies of LL SFC with BE ones. All the parameters we considered are summarized in Table 4.2 and motivated subsequently. One can notice that our model considers both internal and external parallelism for the placement of micro-services. However, internal parallelism has been the sole allowed to collect all the results we present here, in order to fit with the most plausible deployment of technology to date.

1. **Placement and Chaining Strategies**

Our evaluation scenario aims at (1) understanding the performance of our initial model for the placement and chaining of LL SFC composed of micro-services in realistic situations and (2) comparing it with current competitors. In particular, we considered the classes of approaches acknowledged in the literature, namely: (a) Monolithic VNF placement (*Mono*) which is similar to the models developed in [4, 5]; (b) Micro-services placement with neither mutualization nor parallelization enhancements (*Micro*) which is similar to approaches presented in [26, 27]; (c) Micro-services placement with mutualization enhancement (*MicroM*) which is similar to the models developed in [32, 33], and finally (d) Micro-services placement with both mutualization and parallelization enhancements (*MicroMP*); the latter standing for our model.

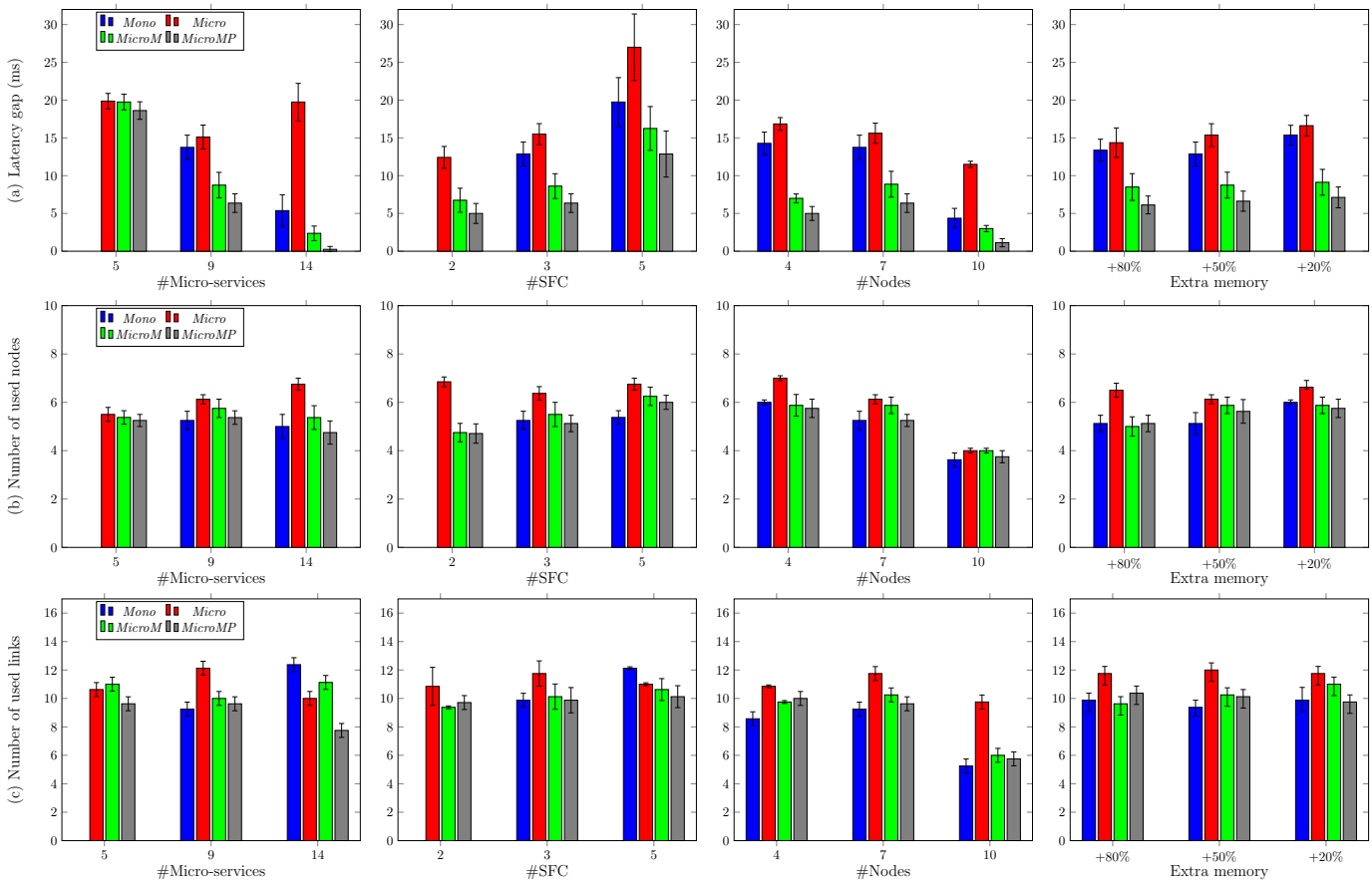


Figure 3.3: Histogram of the different successful measurements featuring the performance of our model (*MicroMP*) against three competitors (*Mono*, *Micro* and *MicroM*). (a) Gap of latency between the SFC requirement and that computed by the placement algorithm; (b) Number of nodes required to place the requested SFC; and (c) Number of links required to place the requested SFC, according to (from left to right sub-figures): The number of micro-services, number of SFC, number of nodes and the extra allocated memory resources. Each result is the average of eight repetitions bounded with 95% confidence intervals.

Criteria	Range or value	
Topology	DFN-Verein European Telco	
VNF	Firewall, NAT, Traffic monitor, IPS	
Micro-services	Read (Rd), Header Classifier (HC), Modifier (Md), Alert (Al), Drop (Dp), Check IP Header (CIH), HTTP Classifier (HC), Count URL (CU), Payload Classifier (PC), Output (Out)	
Link latency	1ms	
Micro-services Proc. latency	1ms	
VNF Proc. latency	4ms	
SFC length	5-14 micro-services	
Node capacity	3-10 instances of micro-services	
Strategy	Placement and chaining	LL and BE SFC cohabitation
#SFC	2-5	3-8
SFC latency	5-10ms	10-15ms
#Nodes	4-10	4

Table 3.4: Evaluation parameters

The implemented topology, extracted from SNDlib[1] library, is that of the DFN-Verein European telco. We have partitioned it by selecting only some Point of Presence (PoP) for a given region. Then, each region has been split into two layers: one node acting as a regional PoP connected to other regional PoPs according to the telco topology, but also acting as an aggregation point for a few local nodes connected to it through a regional loop forming a ring sub-topology. The different SFC we consider are the reflect of those that have been studied in the dedicated literature [34, 11] which are composed of NF (Firewall, NAT, Traffic monitor and IPS) whose decomposition into micro-services is acknowledged. As the core benefits of micro-services, we have considered the mutualization and parallelization tables illustrated in Table 3.5, as defined in [32] and [20], where a "X" in a cell means that the related row and column micro-services can be mutualized or parallelized, respectively. Finally, regarding the varying parameters of our experiments, we have deliberately chosen to limit the computation time, which may be very long in case of exact-resolution, to a realistic order of magnitude to respect what the usage of such a placement algorithm in a real maintenance process of a virtual telco infrastructure could be. As such, the computation time of each experiment has been limited to 600s and, given this limit, the computable instances of placement cover the scales provided in Table 4.2 for the number of nodes, SFC numbers, SFC lengths and nodes capacities. Overall, the results exposed subsequently required more than 30 hours of computation to encompass all cases. Finally, the prescribed latency for SFC ranges from 5 to 10ms.

2. **Cohabitation Strategies**
   Our second scenario aims at analysing the cohabitation of LL and BE SFC deployed under the *MicroMP* approach. Consequently, we reduced the size of the infrastructure as well as that of the SFC in order to obtain the final optimal results of the solver that would allow us to fairly analyse the three cohabitation strategies with no bias due to uncompleted computations. This clearly differs from the first evaluation scenario where the calculation time was limited to 600s. The infrastructure is composed of four interconnected N-PoPs considered as regional nodes. The SFC are composed of one or two NFs (Firewall, Traffic monitor and IPS). Regarding the SFC latencies, they are less tight, varying from 10-15ms as opposed to 5-10ms for the first evaluation, thus better reflecting the needs of current LL services.

3.5.3 Analysis of the Placement and Chaining Results

The different metrics we measure in our performance evaluation campaign are commonly found in the literature related to low latency SFC orchestration [**luizelli15B** , 4, 5, 12]. They aim at first considering the overall latency of service chains as a prerequisite and then the resource consumption to instantiate SFC over the telco infrastructure. More precisely, this stands for (1) the sum of the differences between the latency required by each SFC and the effective latency after deployment, as depicted in first row of Figure 3.3; (2) the number of nodes activated for the deployment of all the SFC, an activated node being a node on which at least one micro-service is instantiated, as depicted in second row of Figure 3.3; and (3) the number of links activated for the deployment of all SFC, as depicted in third row of Figure 3.3.

1. **Latency Benefit**

   The first row of Figure 3.3 shows that *MicroMP* performs better in terms of latency gap in the different configurations under test. Moreover, this latency gain does not require a higher consumption of resources. We also notice that the more the system increases in size and load, the greater the benefit of *MicroMP* is. The reason is that the more micro-services there are, the more chances of parallelism and mutualization there are, these two enhancements allowing the actual latency reduction. One can also see that *Micro* is the approach with the largest latency gap for all configurations. This is due to its basic decomposition into micro-services which generates a latency overhead since there are more entities to deploy. This confirms again that the decomposition into micro-services is relevant in terms of latency gain only when using mutualization and parallelism enhancements. One can also see from the last figure of the first row, that bringing more available resources does not induce much gain in latency. The three configurations +80%, +50% and +20% are indeed almost equivalent regarding that metric.

2. **Usage of Infrastructure Resources**

   The second and third rows of Figure 3.3 represent the nodes and links usage of the four approaches according to the different configurations. One can see that the usage of resources is more important when switching to the micro-services approach and this for all the configurations. The reason is again that with the micro-services approach, more entities have to be deployed, a VNF being composed of four micro-services on average

[1] Survivable fixed telecommunication Network Design – http://sndlib.zib.de/

	Rd	HC	Md	Al	Dp	CIH	HC	CU	PC	Out
Rd	x									
HC		x		x	x	x		x		
Md			x	x	x	x		x		
Al				x	x	x		x		
Dp				x	x	x		x		
CIH				x	x	x		x		
HC							x			
CU				x	x	x		x		
PC									x	
Out										x

	Rd	HC	Md	Al	Dp	CIH	HC	CU	PC	Out
Rd	x									x
HC	x	x	x	x	x	x	x	x	x	x
Md	x			x	x	x	x	x	x	x
Al	x	x		x	x	x	x	x	x	x
Dp	x			x	x	x	x	x		x
CIH				x	x	x		x		
HC	x	x		x		x	x	x		x
CU	x		x	x	x	x	x	x	x	x
PC	x			x	x	x	x	x		x
Out	x									x

Table 3.5: Parallelism (left) and mutualization (right) tables as defined in [32] and [20]

[32]. Nevertheless, mutualization manages to reduce the consumption of resources by reducing the number of micro-services to be deployed, thus exhibiting a consumption equivalent to that of *Mono*. Moreover, parallelism forces some of the micro-services to be deployed on the same node, thus bringing a slight gain for *MicroMP* as compared to *MicroM* and which, in some cases, even exceeds *Mono*.

3. **Deployment Agility**

   The third important aspect to notice concerns the deployment agility. Indeed, it can be seen that for some instances and under some configurations (#Micro-services = 5 and #SFC = 2), the model cannot find a solution for *Mono*, despite a sufficient amount of infrastructure resources for their deployment. Indeed, the breakdown into micro-services allows for better agility as compared to *Mono* because the components are lighter and memory space management is therefore easier to achieve.

4. **Number of Accepted SFC** The first row of Figure

   3.3 indicates that *MicroMP* is the approach which better minimizes the latency gap for the set of SFC it has to place and chain. However, it does not indicate how this latency gap is distributed among the different SFC which would be relevant to understand the fairness of the model between SFC. To that aim, we measured the number of latency compliant SFC in a configuration where the prescribed latency of SFC is relaxed to 10-15ms. The obtained results are summarised in Figure 3.4 which, given our computation time limit, provides an optimality gap of 0 for *Mono*, [2-33%] for *Micro*, [0-20%] for *MicroM* and *MicroMP*. It clearly exhibits that the number of SFC respecting the prescribed latency is the highest for *MicroMP*, *Mono* and *MicroM* having a lower performance and *Micro* being the worst, thus confirming the fair behavior of our model regarding the latency compliance of SFC.

5. **Analysis of Computation Features**

   Table 3.7 summarises the average computation features of CPLEX for the different instances. Each instance represents a configuration setup with the values presented in Table 4.2, and it has been repeated eight times with different SFC to bring some randomness. Regarding the computation time, limited to 600s, we notice that for *Mono*, the optimal solutions were found much earlier (1,5 seconds on average) contrary to *Micro* for which the resolution time is over our limit for almost all instances. In a similar way, the gap with respect to the optimal solution of *Mono* is zero while that of *Micro* is around 45%. This indicates that the solutions obtained for *Micro* can be improved if we allowed more computation time. Nevertheless, with such a limited computation time, *Micro* is penalized. By contrast, regarding *MicroM* and *MicroMP*, the computation time is below the limit for more than 55% of the cases. In comparison to *Mono*, this reveals two insights: (1) with *MicroM* and *MicroMP* the obtained solutions are not necessarily optimal and, in spite of that, they are better

than the optimal solutions of *Mono*. This means that with more computing time, one could have even better solutions for *MicroM* and *MicroMP*; (2) the switch to *MicroM* and *MicroMP* requires more computing time due to the complexity of the approach, but less than *Micro* since there are fewer micro-services to deploy, thus demonstrating the reasonable cost of these approaches.

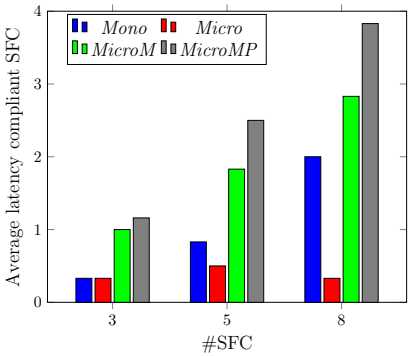


Figure 3.4: Latency compliant LL SFC

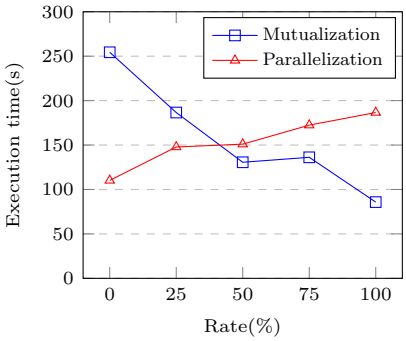


Figure 3.5: Execution time with different parallelization and mutualization rates

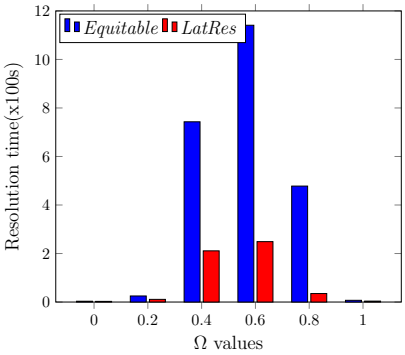


Figure 3.6: Resolution time for *Equitable* and *LatRes* strategies

3.5.4 Evaluating the Impact of Mutualization and Parallelization Rate on Model Performance

Since mutualization and parallelization are the cornerstone which makes *MicroMP* outperforming its competitors with the setup of Table 3.5, we have eventually restricted our study to this sole approach and analysed the impact of mutualization and parallelization rates on the performance of the model. To accomplish this, we have established five levels of micro-services mutualization and parallelization, allowing 0%, 25%, 50%, 75%, and 100% of possible mutualization/parallelization cases, represented by five distinct related tables. For each level, we evaluated the deployment of five SFC consisting of 14 micro-services, with a required latency of 7 ms on a network infrastructure composed of 7 nodes. The results are summarized in Table 3.6.

1. **Analysis of mutualization impact**

   Our analysis shows that as the mutualization rate increases, the latency gap obviously decreases. Transitioning from 0% to 100%, mutualization reduces the latency gap from 5.06 ms to 3.48 ms, illustrating the latency optimization through micro-services mutualization and consequently shorter SFC. The memory usage follows a similar trend, declining from 84% to 65% with an increased mutualization, due to shorter SFC and fewer micro-services to deploy. Regarding the number of links and nodes used, we observe that they remain constant across different mutualization rates, with 8 links and 6 nodes utilized in each case. This observation can be explained by two factors: first, the relatively small scenario size, which does not offer the possibility of a large reduction of the path; and second, the MILP model's formulation, which aims to minimize the latency gap but does not explicitly focus on reducing the number of links and nodes used. In this evaluation, we also sought to measure the number of parallelizations, finding that the number of parallelizations decreases as mutualization increases. This outcome may appear somewhat contradictory, as both mutualization and parallelism share the common goal of reducing overall latency. However, we observe that the reduction of latency remains important even when increasing mutualization induces a decrease in parallelism. Actually, less micro-services in total impacts the possibilities of parallelizing, but it appears to be still better than less mutualization.

   In conclusion, the analysis of Table 3.6 demonstrates that mutualization has a positive impact on the latency gap, computation time, and resource usage. Although the number of micro-services functioning in parallel decreases, the advantages of mutualization out-weight this loss, resulting in overall performance improvements.

2. **Analysis of parallelization impact**

   Our analysis of the impact of parallelization on the model reveals that a decrease in latency is observed as the parallelization rate increases, while the memory usage remains constant at 82%. This result is expected because memory usage does not directly depend on the parallelization rate but rather on the mutualization, which reduces the number of deployed micro-services. As for the number of links and nodes used, we observe

Mut. rate	Gap	Mem. used	#Links	#Nodes	#Par.
0%	5,06	84%	8	6	9
25%	5,05	82%	8	6	8
50%	4,51	80%	8	6	8
75%	4,09	73%	8	6	6
100%	3,48	65%	8	6	4
Par. rate	**Gap**	**Mem. used**	**#Links**	**#Nodes**	**#Par.**
0%	7.23	82%	11	7	0
25%	7.14	82%	11	7	2
50%	6.74	82%	8	6	4
75%	6.21	82%	8	6	5
100%	5.01	82%	8	6	9

Table 3.6: Impact evaluation of the mutualization and parallelization rates on model performance, with Mut. rate(%): Mutualization rate, Par. rate(%): Parallelization rate, Gap(ms): Latency gap, Mem. used(%): Rate of memory used, #Links/Nodes: Number of used links/nodes, #Par.: Number of parallelizations

that it decreases as the parallelization rate increases. Specifically, the number of links decreases from 11 links for 0% parallelization rate to 8 for 50% parallelization rate and then remains constant at 8 for subsequent parallelization rates. Similarly, but to a lower degree, the number of nodes decreases from 7 activated node for 0% parallelization rate to only 6 for higher parallelization rates. This decrease is justified by the fact that internal parallelization forces our MILP to place micro-services in close proximity, which reduces the use of links and nodes. Finally, we observe that the number of parallelizations increases as the parallelization rate increases, from 0 micro-service functioning in parallel for 0% parallelization rate to 9 for 100% parallelization rate. This result seems obviously but it reveals that our model is able to fully exploit the parallelization possibilities offered by the table which is not that trivial since under some configurations one may have expected an under-utilization of this latency reduction means. In conclusion, the analysis of the table reveals that parallelization of micro-services has a positive impact on latency and resource utilization while reducing the complexity of the system.

Finally, we studied the impact of mutualization and parallelization rates on the solver execution time. Figure 3.5 shows the variation of execution time as a function of the parallelization rate (in red) and the mutualization rate (in blue). We observe a linear decrease (between 0-50% and 75-100%) in execution time as the mutualization rate increases, but an increase in execution time as the parallelization rate increases, but with a lesser degree than mutualization. This demonstrates that mutualization leads to a significant improvement in execution time by reducing the number of deployed micro-services, while parallelization, when increasing its rate, increases the number of possible placement combinations, which lead to longer execution times.

3.5.5 Analysis of the Cohabitation Strategies Results

As a second evaluation step of our overall approach, we have quantified and analysed the differences induced by the three strategies we have selected to guide the cohabitation between BE and LL SFC. We have especially measured for each strategy the number of deployed LL and BE SFC, the LL SFC latency compliance rate and the total BE SFC latency as a function of Ω. Table 3.8 summarises the results we have collected and we analyze subsequently.

1. **Comparison of Deployment Results**

   The first relevant outcome revealed by our evaluation in Table 3.8 is the strict equality of the results we collected for the *Equitable* and *LatRes* strategies, although their respective objective function and constraints are different. For these two strategies, the number of deployed LL SFC increases with Ω and inversely for the number of deployed BE SFC. This is a straightforward result which assesses the actual impact of Ω as a priority parameter for one class over the other. An equitable deployment between the BE and the LL SFC can set a value between 0.4 and 0.6, more priority to the BE with a value between 0 and 0.4 and finally, more priority to the LL with a value between 0.6 and 1, the two values $\Omega = \{0,1\}$ being meaningless since they completely prevent the deployment of one class of service. Then, one can notice that the latency compliance of LL SFC of the *Equitable* strategy is always 100% even if not constrained, contrary to the *LatRes* strategy where strict compliance of latency is mandatory for LL SFC. This indicates that deploying more or less BE SFC does not impact the latency of LL SFC because the deployment solution computed by our model, if found, always leads to the optimal latency which respects the prescribed one. This assessment holds for all our scenarios but it may fail if the infrastructure setup does not offer a deployment solution respecting the prescribed latency in LL SFC. This left aside, this observation is the core reason which explains the strict equivalence of the first two strategies.

	Mono			*Micro*			*MicroM*			*MicroMP*		
I	Obj	Dur	Gap	Obj	Dur	Gap	Obj	Dur	Gap	Obj	Dur	Gap
1	-	-	-	19	600	9	19	600	8	18	506	3
2	13	4	0	15	600	53	8	600	27	6	575	18
3	5	4	0	19	600	47	2	584	17	1	111	2
4	-	-	-	12	600	41	6	4	15	5	364	6
5	12	3	0	15	600	53	8	600	27	6	599	18
6	19	2	0	27	600	35	16	600	47	12	600	38
7	14	1	0	16	526	42	7	600	40	5	600	3
8	13	1	0	15	600	48	8	600	30	6	597	1
9	4	1	0	11	600	47	3	600	19	1	395	0
10	13	3	0	14	600	51	8	600	28	6	2	1
11	12	2	0	15	600	52	8	600	27	6	600	2
12	15	4	0	16	600	44	9	600	32	7	600	23

Table 3.7: Features of the different model computations, with I: Instance, Obj (ms): Objective function, Dur (s): Computation duration, Gap (%): Optimality gap.

	Equitable & *LatRes* strategies				*LLDep* strategy			
Ω	LL	BE	LL comp	BE lat avg	LL	BE	LL comp	BE lat avg
0	0	3	0%	7,33	4	1	100%	6
0,2	2	3	100%	7,33	4	1	100%	6
0,4	3	2	100%	7	4	1	100%	6
0,6	4	1	100%	6	4	1	100%	6
0,8	4	1	100%	6	4	1	100%	6
1	4	0	100%	-	4	0	100%	-

Table 3.8: Cohabitation strategies results, with LL: number of deployed LL SFC, BE: number of deployed BE SFC, LL comp (%): rate of LL compliant with the prescribed latency, BE lat avg (ms): BE latency average

The *LLDep* strategy confirms that the deployment of BE SFC does not impact the latency of LL SFC, despite the variation of Ω which leads to an identical number of deployed BE SFC. Given that the *LLDep* strategy requires a strict deployment with a minimization of the latency gap for LL SFC, the results indicate that the latency does not vary either for BE or LL SFC even with the variation of Ω, thus demonstrating a relevant property of our model regarding the latency respect: when an LL SFC is deployed, this is achieved in an optimal way and the deployment of more or less BE has no impact on its latency.

2. **Resolution Time Analysis**

To further assess to what extent the *Equitable* and *LatRes* strategies are equivalent, we compared the resolution time they required, since in spite of the equality of their numerical results, the model does not generate them in the same way. In the *Equitable* strategy, the model tries to minimise the latency gap, whereas in the *LatRes* one, it directly tries to find a solution that allows the prescribed latency to be respected. Figure 3.6 represents the computation time of our model for the two strategies as a function of Ω. It shows that the computation time is equivalent and relatively small for the extreme values of Ω and is significantly larger when the values of Ω are in the $]0.2, 0.6[$ range. In this latter range, the model requires more computation as it attempts to find an equitable solution for the two SFC classes. Besides, Figure 3.4 shows that the *Equitable* strategy requires substantially more computational time than the *LatRes* one, from 4 to 10 times at worst. Although their results are equivalent, the model tries to minimise the latency gap on the *Equitable* strategy which consumes more time whereas it tries to directly find the solution that respects the prescribed time on the *LatRes* one, which is less costly in terms of computational time. It would be tempting to conclude that the *LatRes* strategy is more relevant because it gives the same results as the *Equitable* one and in less time, however if the infrastructure is restricted and there is no solution that allows the latency of deployed LL SFC to be respected, the *LatRes* strategy will not produce any solution whereas the *Equitable* one will produce a solution with an optimal latency of LL SFC even if the LL SFC do not respect the prescribed one.

Chapter 4

Heuristic approach

In this section, we present the micro-services placement and chaining problem we tackle as well as the dedicated heuristic method we propose. The latter aims to (1) place and chain the micro-services and (2) manage an adaptive parallelization while (3) maximizing the number of SFC meeting the prescribed latency and (4) optimizing load balancing. To reach these objectives, our lightweight heuristic integrates several optimization techniques, exploiting the intrinsic characteristics of micro-services, that are introduced subsequently in a step-by-step approach.

4.1 Problem Statement

The micro-services placement and routing problem we investigate is defined by a network graph $G = (N, L)$, where N represents a set of nodes n, characterized by a memory capacity M_n. L represents a set of links between two nodes n_i, $n_j \in N$ characterized by a latency link $\delta_{n_i n_j}$. Q is a set of SFC requests, with each request, $q \in Q$, characterized by a source and destination represented respectively by s_q, $d_q \in N$, a required latency rl_q and a set of micro-services $m \in M$, where M is the set of all types of micro-services that an edge flow must traverse. The objective of this problem is to:

- Place the micro-services for each SFC;
- Chain micro-services together.

Subject to :

- Memory capacity constraints on the nodes;
- Micro-services forwarding and execution latency constraints;
- Micro-services execution order constraints;
- Parallelism execution constraints.

4.2 Algorithm Overview

The heuristic, accepting pre-processed SFC set Q and network infrastructure G as inputs, generates a placement solution for all SFC within the infrastructure capacity. Pre-processing involves mutualizing and identifying parallelizable micro-services, as detailed in our previous work [44]. Composed of three key algorithms, the heuristic utilizes $EppRep()$ and $NeShPat()$ for k shortest path calculation, and $SFCDeploymentAlgorithm()$ for a load-balanced SFC deployment, which itself employs $ServicesPlacementAlgorithm()$ to deploy micro-services while managing parallelization.

This section elaborates the heuristic primary steps, special attributes, shortest path calculation, parallel placement and chaining procedures, and it introduces an online learning approach enhancing the heuristic performance. Table 4.1 provides a comprehensive overview of all parameters and variables we consider in the following.

Functions	**Definition**
$EppRep(s_q, d_q, G)$	Gives *Eppstein* representation for paths from S_q to d_q in G
$NeShPa(ER_q)$	Finds next shortest path using *Eppstein* rep. ER_q
$AvaiSpace(p)$	Computes available memory on a path p
$Place(m, n)$	Assigns micro-service m to node n
$Para(m1, m2)$	Checks if micro-services $m1$, $m2$ can run in parallel
$Cont(n, m)$	Checks if micro-service m is on node n
$IsCritical(q)$	Checks if SFC q length is less than its rl_q
$Move(m, n)$	Migrates micro-service m to node n
$TwiceDecr(q)$	Checks if SFC q score decreased twice consecutively
Sets	**Definition**
$q \in Q$	Set of SFC
$m \in M$	Set of micro-services
$m \in PM_{m_n}$	Set of micro-services operating in parallel to m_n
Parameters	**Definition**
G	Network infrastructure for SFC deployment
ER_q	*Eppstein* representation for SFC q
$NbIt$	Number of iterations for the online learning process
Δ_s	Constant to adjust SFC q score
N	Factor to amplify SFC q score deterioration
s_q	Source node of SFC q
d_q	Destination node of SFC q
Variables	**Definition**
sp_q	Shortest path for SFC q
as	Available space
mr	Space left on p post q deployment
mpn	Space left on p post q deployment per node
rm	Residual space on p post deployment
ca_n	Memory capacity of node n
m^-	Precedent of micro-service m
sc_q	SFC q score
rl_q	Required Latency of SFC q
el_q	Effective latency of SFC q

Table 4.1: Functions, sets, variables, and parameters considered in our heuristic method

4.2.1 Main steps of the heuristic

As shown in Algorithm 1, to optimise the deployment of a set of SFC, our heuristic starts by ordering the SFC in ascending order of the margin between their required latency and the SFC length (line 1). Indeed, the smaller the latency gap, the more critical the SFC. Next, for each SFC $q \in Q$ (loop on line 2–11), the method executes two first operations: it computes a modified *Eppstein* representation which enables the calculation of the k loop-free shortest paths (line 3) and sets the flag dep_q to $false$ indicating that the SFC remains undeployed (line 4). Then, while SFC q is not deployed and a k^{th} shortest path is available (line 5), the heuristic determines the k^{th} shortest path based on the computed *Eppstein* representation (line 6). Subsequently, it verifies whether the memory capacity of the path is adequate for the deployment of SFC q (line 7). If the memory capacity is found to be sufficient, the heuristic attempts the deployment of SFC q by invoking $SFCDeploymentAlgorithm()$ with the SFC q and shortest path sp_q parameters (line 8).

Algorithm 1 Overall heuristic method for micro-service SFC placement and chaining

Input: Set of SFC Q, infrastructure G
Output: Placement and chaining solution for all SFC

1: Order SFC according to latency gap value
2: **for all** SFC $q \in Q$ **do**
3: $ER_q \leftarrow EppRep(s_q, d_q, G)$
4: $dep_q \leftarrow$ **false**
5: **while** $\neg dep_q$ **and** $NeShPa(ER_q).exist()$ **do**
6: $sp_q \leftarrow NeShPa(ER_q)$
7: **if** $Capacity(SP_q) \leq Length(q)$ **then**
8: $dep_q \leftarrow SFCDeploymentAlgorithm(q, sp_q)$
9: **end if**
10: **end while**
11: **end for**

4.2.2 Shortest Path Computation

In a micro-service SFC placement and chaining scenario, determining only one shortest path per SFC may be insufficient due to bounded node memory. This necessitates the identification of the k shortest paths where, if the $(k-1)^{th}$ path lacks sufficient memory capacity, we deploy the SFC on the k^{th} path, maintaining flexibility and robustness. To optimally address this, we leverage the *Eppstein* algorithm [19], an efficient solution for graphs without negative loops, which fits with our context where edge latency cannot be negative. *Eppstein* algorithm complexity is $O(m + n\log(n) + k\log(k))$, with m, n, and k representing the number of edges, nodes, and calculated paths, respectively. The algorithm operates according to the following steps: (1) it starts with the *Dijkstra*'s algorithm calculating the shortest paths between each node and destination; (2) it forms a unique graph, which allows a direct transit to the subsequent node, or a detour to an auxiliary node, with latency variation; (3) it outputs the shortest path, while empty auxiliary node sets represent direct paths. For each node on the shortest path, (4) it forms sets of shortest paths and adds feasible paths to a heap. Finally, (5) it extracts the minimum-cost set from the heap, computes their shortest path, and repeats until all paths are explored.

When graphs contain positive loops, shortest paths can be infinite due to possible loop traversals on each path calculation. Since repeated node traversal does not affect the path's micro-service deployment capacity, we need loop-free paths. Consequently, we modified the *Eppstein* algorithm at two points to exclusively produce k shortest loop-free paths: one at step (2) to remove edges if the auxiliary node creates a loop with the original path, and at step (3) to eliminate looping paths. These modifications ensure paths are loop-free, maximizing *Eppstein* algorithm's efficiency in our context. In our method, the $EppRep()$ function oversees the operation of the modified steps (1-2), while the $NeShPa()$ function manages the execution of the modified steps (3-5).

4.2.3 Load Balanced SFC Deployment

The deployment of SFC produced by Algorithm 2 consists in optimising the placement of micro-services to maximise the number of SFC deployed, while respecting the prescribed latency and balancing the load on the network nodes.

Our load balancing strategy, inspired by the $Waterfilling$ algorithm [45], aims to fairly distribute remaining space for post-SFC deployment on nodes. However, two challenges arise: (1) memory allocation indivisibility, which necessitates a strategy for managing integer division and leftover fractions; (2) the potential resource distribution

imbalance due to micro-services parallelization, which demands an integrated approach for system balance maintenance. To achieve this balanced deployment, we propose to use four metrics: available space as ; the margin mr ; the margin per node mpn and the residual margin (rm) which are described in Table 4.1.

The proposed procedure, described in Algorithm 2, employs an iterative method to traverse two lists: the list of micro-services associated with SFC q, indexed by m, and the list of nodes within the specified path p, indexed by n. Initially, the algorithm considers the first micro-service and node (line 1 and 2). While any SFC remains undeployed and nodes persist within the path, the heuristic executes the ensuing steps (line 7-16). First, it checks whether the current node can host the current micro-service (line 7). This assessment entails verifying if the available space surpasses the mpn metric. If the node proves to be sufficient, the algorithm invokes the $ServicesPlacementAlgorithm()$ (step 10) to deploy the micro-service m onto node n. Subsequently, if the deployment is successful (line 10), it transits to the next micro-service while maintaining its position at the current node (step 11). Conversely, if the node cannot host the micro-service, the algorithm progresses to the next node without transitioning to the next micro-service (step 14). By implementing this procedure, the algorithm facilitates balanced micro-service deployment on path p by evenly distributing the remaining space among the nodes. One can notice that rm will be used by algorithm $ServicesPlacementAlgorithm()$.

Algorithm 2 SFC Deployment Algorithm

Inputs: SFC q, path p
Output: Deployment of SFC q

1: int $n \leftarrow 0$ // *index for iterating over path p*
2: int $m \leftarrow 0$ // *index for iterating over SFC q*
3: int $as \leftarrow AvaiSpace(p)$
4: int $mr \leftarrow as$ - $Length(q)$
5: int $mpn \leftarrow Quotient(marge/Length(p))$
6: int $rm \leftarrow Remainder(margin/Length(p))$
7: **while** $m \leq Length(q)$ **and** $n \leq Length(p)$ **do**
8: **if** $ca_n > mpn$ **then**
9: $dep_q \leftarrow \mu ServicesPlacementAlgorithm(q, p, m, n)$
10: **if** dep_q **then**
11: $m \leftarrow m + 1$
12: **end if**
13: **else**
14: $n \leftarrow n + 1$
15: **end if**
16: **end while**

4.2.4 Micro-services Placement and Internal Parallelisation

To leverage the internal parallelization of micro-services for latency reduction, we propose a strategy, described in Algorithm 3, that begins by placing a micro-service m on node n (line 1), then assess whether the preceding one m^- can feasibly operate in parallel with m. If so and if not already operating in parallel (line 3), the algorithm checks if m^{-1} is deployed on n (line 4) and if so, it integrates m^{-1} to the parallelizable set PM_m (line 5). If m^{-1} is not deployed on n, it verifies if (i) there is enough residual margin rm that could be used to avoid unbalancing the deployment too much, or (ii) if SFC q is critical, meaning that the parallelization is crucial to satisfy the latency constraint. Then the algorithm checks also if path p has enough capacity to move m^- to node n (line 6). Indeed, the parallelizing m with m^{-1} involves the migration to n which leads to imbalances in the load deployment. In cases where the conditions are met, the algorithm moves m^- to n (line 7) and integrates m^- to PM_m (line 8). In the other case when m is parallelizable with m^{-1} and the latter is already operating in parallel with another(s) micro-service(s), the algorithm checks if all micro-services within $PM_{m^{-1}}$ are capable of running in parallel with m (line 10). If so, and m^- has already been deployed on n (line 11), the algorithm integrates m into the existing group $PM_{m^{-1}}$ of parallelizable micro-services of m^- (line 12). If m^- is not placed on the same node (line 13), the algorithm performs the same checks as line 6, but with the set of micro-services operating in parallel with m^-, contained in PM_{m^-}. In cases where the conditions are met, the algorithm moves them to n including m^- (line 16) and integrates m to PM_{m^-} (line 18).

Algorithm 3 µServices Placement Algorithm

Inputs: SFC q, path p, micro-services m, node n
Output: optimized deployment of micro-service m

1: $Place(m, n)$
2: **if** $Para(m, m^{-})$ **then**
3: **if** $PM_{m^{-1}} = \varnothing$ **then**
4: **if** $Contains(n,m^{-})$ **then**
5: $PM_m \leftarrow PM_m \cup m^{-1}$
6: **else if** $rm > 0$ **or** ($IsCritical(q)$ **and** (ca_p - $Length(q)$) > 0 **then**
7: $Move(m^{-},n)$
8: $PM_m \leftarrow PM_m \cup m^{-1}$
9: **end if**
10: **else if** $\forall\ \mu_i \in PM_{m^{-1}}$, $Para(m, \mu_i)$ **then**
11: **if** $Contains(n,m^{-})$ **then**
12: $PM_{m^{-1}} \leftarrow PM_{m^{-1}} \cup m$
13: **else**
14: **if** $rm > Length(PM_{m^{-}})$ **or** ($IsCritical(q)$ **and** (ca_p - $Length(q)$) $> Length(PM_{m^{-}})$ **then**
15: **for all** $\mu_j \in PM_{m^{-}}$ **do**
16: $Move(\mu_j,n)$
17: **end for**
18: $PM_{m^{-1}} \leftarrow PM_{m^{-1}} \cup m$
19: **end if**
20: **end if**
21: **end if**
22: **end if**

4.3 Ordering the SFC Processing

A notable drawback of heuristic methods over the exact one relies in their sequential SFC deployment, which does not optimize the entire SFC set concurrently as in mathematical models. This reveals that the processing order has significant impact on the solution performance. Consequently, we employed an online learning process to optimally order the SFC processing, as described in Algorithm 4.

The heuristic starts by uniformly assigning for each SFC in Q a consistent score "s" and it computes a modified *Eppstein* Representation (line 1-2). Then, at the start of each iteration $it \in NbIt$, whose upper bound has been empirically assessed to half of the number of SFC per instance (line 5), the heuristic deploys them in ascending order of scores unlike latency gap value as detailed in Section 1 (line 6). After deployment, it checks each SFC's latency compliance (line 8). If an SFC meets latency requirements, its score is increased, thus lowering its priority (line 9). If it does not, its score is reduced, thus raising its priority (line 11). Over multiple iterations, the solution progressively improves. However, if an SFC consistently fails to meet latency requirements despite its priority, its score is intentionally increased after two consecutive decreases, downgrading it (line 12-13). This ensures the heuristic does not prioritize SFC that persistently violate latency specifications.

4.4 Evalution

To assess the performance of our heuristic method under diverse scenarios, we implemented it in C++. The implementation consists of 2300 lines of code accessible at: https://www.mosaico-project.org/outcomes for replication purposes. To validate our implementation, we performed comprehensive checks, ensuring that (1) there are no circuits, (2) all micro-services related to each request are deployed, (3) the sequence of micro-services is respected, (4) memory resource consumption is respected, (5) latency computation is accurate, and (6) parallelism and mutualization are correctly implemented in accordance with their respective tables. All our experiments were conducted on an 11th generation Intel Core i7-1165g7@2.80GHz 1.69GHz computer, with 16GB of RAM and operating on Windows 10 Professional Education.

4.4.1 Evaluation Scenarios

Our performance evaluation scenarios aims at (1) understanding the performance of our model in realistic situations and (2) comparing it with the optimal solution ($CPLEX$) generated with the model we proposed in [44]. We also

Algorithm 4 Online learning approach for micro-service SFC placement and chaining

Inputs: set of SFC, infrastructure
Output: deployment of all SFC

for all SFC $q \in Q$ **do**
  $sc_q \leftarrow s$ *// assigns identical score to all SFC*
  $ER_q \leftarrow EppRep(s_q, d_q, G)$
end for
while $it \leq NbIt$ **do**
  // Deploy all SFC as per Heuristic 1 except that the ordering is based on the s score instead of the latency gap value
  for all SFC $q \in Q$ **do**
    if $el_q \leq rl_q$ **then**
      $sc_q \leftarrow sc_q + \Delta_s$ *// score downgrading*
    else
      $sc_q \leftarrow sc_q - \Delta_s$ *// score improvement*
      if $Rank(q) = 1$ and $TwiceDecr(q)$ **then**
        $sc_q \leftarrow sc_q + N * \Delta_s$ *// major deterioration score*
      end if
    end if
  end for
  $it \leftarrow it + 1$
end while

Parameter	Range or value
Topology	DFN-Verein European Telco
VNF	Firewall, NAT, Traffic monitor, IPS
Micro-services	Read (Rd), Header Classifier (HC), Modifier (Md), Alert (Al), Drop (Dp), Check IP Header (CIH), HTTP Classifier (HC), Count URL (CU), Payload Classifier (PC), Output (Out)
SFC latency	5-10ms according to the SFC
Link latency	1ms
Micro-services proc. latency	1ms
SFC length	5-14 micro-services
Node capacity	3-10 instances of micro-services
k	10
sc_q	set to the number of SFC in the scenario
Δ_s	1
N	set to the number of SFC in the scenario

Table 4.2: Evaluation parameters

evaluate the relevance of the different improvements we exposed above, namely: Heuristic without parallelization, online learning approach and load balancing (H); Heuristic managing parallelization (H-P); Heuristic managing parallelization and load balancing (H-PB); (H-PLB) being the last and standing for our selected candidate which manages online learning approach in addition to the version (H-PB).

All the parameters we considered in our evaluation are summarized in Table 4.2 and motivated subsequently. The implemented topology, extracted from the SNDlib[1] library, is that of the DFN-Verein European telco. We have partitioned it by selecting only some Point of Presence (PoP) for a given region. Then, each region has been split into two layers: one node acting as a regional PoP connected to other regional PoP according to the telco topology, but also acting as an aggregation point for a few local nodes connected to it through a regional loop forming a ring sub-topology. The different SFC we consider are the reflect of those that can be found in the dedicated literature[11], each of them being splitable into micro-services. Their split are technically realistic and derived from the literature [32, 33]. As the core benefits of micro-services, we have considered the mutualization and parallelization tables illustrated in [20]. The initial number of shortest paths to compute, denoted as k, is set to 10. In case the algorithm is unable to deploy the SFC on one of these paths, it calculates the next 10. This process continues until a deployment is possible or no more shortest paths are available. As for sc_q, it is defined as the number of instances per scenario. This enables a homogeneous ranking of the SFC. Variable Δ_s is set to 1, which is enough to prioritize one SFC over another. Finally, N is also defined as the number of SFC per scenario. This definition is crucial to cause a significant deterioration in the ranking when necessary. All the results presented subsequently are the mean of eight repetitions bounded with 95% confidence intervals.

4.4.2 Result Analysis

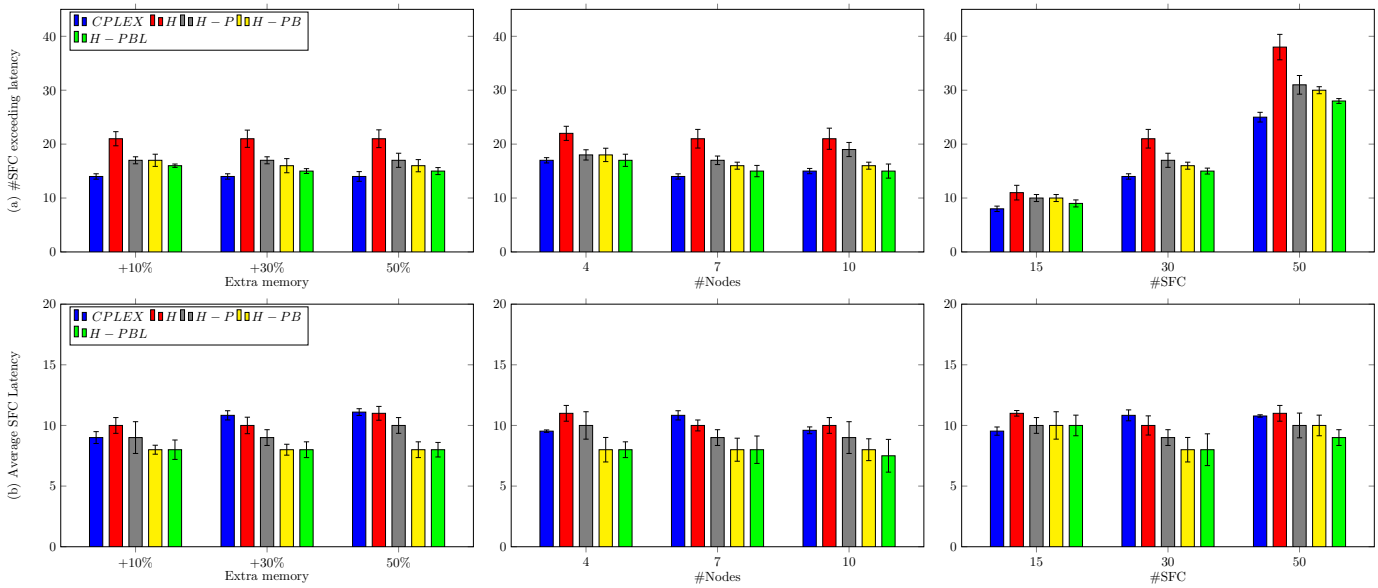


Figure 4.1: Histogram of exceeded SFC latency and average SFC latency under CPLEX and heuristic approaches, with varying extra memory (left), infrastructure nodes (middle), and SFC per instance (right)

In our evaluation, we use several metrics to assess the heuristic's performance and its variants. Primary indicators include the quantity of SFC exceeding the required latency and the mean latency per scenario. We also study the average optimality gap, representing the heuristic solution deviation from the optimal. We estimate the computation duration by modifying the number of SFC instances per scenario and comparing the results to the exact approach's ($CPLEX$) computation duration. Additionally, we scrutinize the computation time by adjusting the number of nodes while keeping the SFC instances constant. Finally, we assess load balancing with the Jain index, an acknowledged metric in the related literature [46, 47]. It quantifies deployment load balance with an index varying from 0 (unbalanced) to 1 (balanced).

1. **Number of SFC Exceeding Latency**

   Reviewing all scenarios of Figure 4.1.a highlights that, as expected, the exact $CPLEX$ method outperforms others, specifically the H approach which lacks optimization and neglects micro-service peculiarities. The

[1] Survivable fixed telecommunication Network Design – http://sndlib.zib.de/

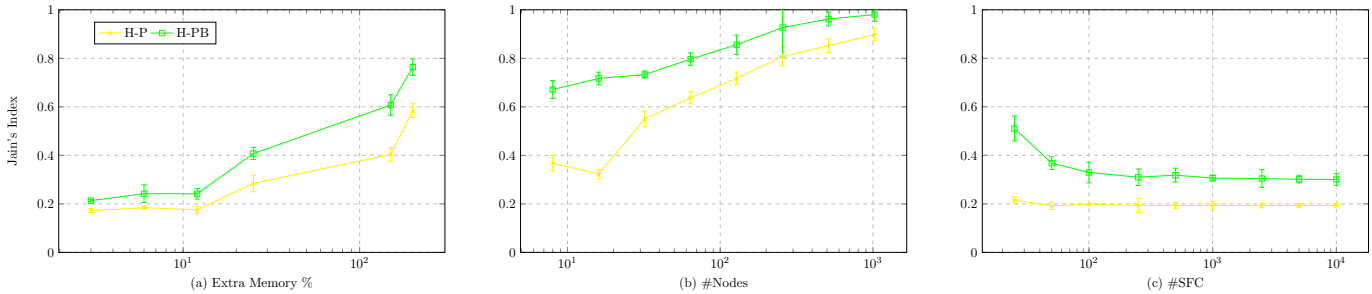


Figure 4.2: Jain index for micro-services deployments as a function of (a) the extra allocated memory, (b) number of nodes and (c) number of SFC

$H-P$ variant shows marked improvement due to effective parallelization during deployment. The load-balancing version, $H-PB$, optimizes performance in 6 out of 9 scenarios. The remaining scenarios, constrained by limited memory infrastructure, maintain performance levels, since the impact of load balancing decreases. However, when the infrastructure is not overloaded, load balancing distributes available space efficiently across the network, facilitating SFC deployment on the shortest paths, hence respecting latency. Finally, online learning approach ($H-PBL$) enhances the heuristic performance by countering its sequential limitations on all scenarios.

In terms of metric variation, with more extra memory, the performance slightly improves due to increased placement and parallelization possibilities. However, when more nodes maintain the same total capacity, the performance of $H-P$ deteriorates as the possibility of parallelization decreases because of fixed memory capacities, which means that less space per node is available. Nevertheless the performance increases for $H-PBL$ thanks to the load balancing approach. Concerning the SFC number variation, we note that the gap between the exact approach $CPLEX$ and heuristic versions (H) and ($H-P$) notably widens when SFC increases. However, our heuristic ($H-PBL$) maintains a consistent difference with $CPLEX$, proving its robustness regardless the number of scenarios.

2. **Average Latency**

   As depicted in Figure 4.1.b, this analysis reveals a paradoxical behavior. Indeed, although the exact approach ($CPLEX$) offers superior performance in terms of the number of SFC exceeding latency as compared to our heuristic approach, it shows the worst performance in terms of average of the execution times of all latencies. This phenomenon is explained by the fact that in the exact approach, when an SFC exceeds the latency, the model does not limit its exceeding value. By contrast, the heuristic optimizes each SFC by trying to minimize its latency, thereby enabling a more uniform deployment in terms of latency compliance. We also observe an improvement in the average latency under the $H-PBL$ approach as compared to the $H-PB$ approach under the scenarios with 10 nodes and the one with 50 SFC, as the benefit of SFC ordering is larger in these cases.

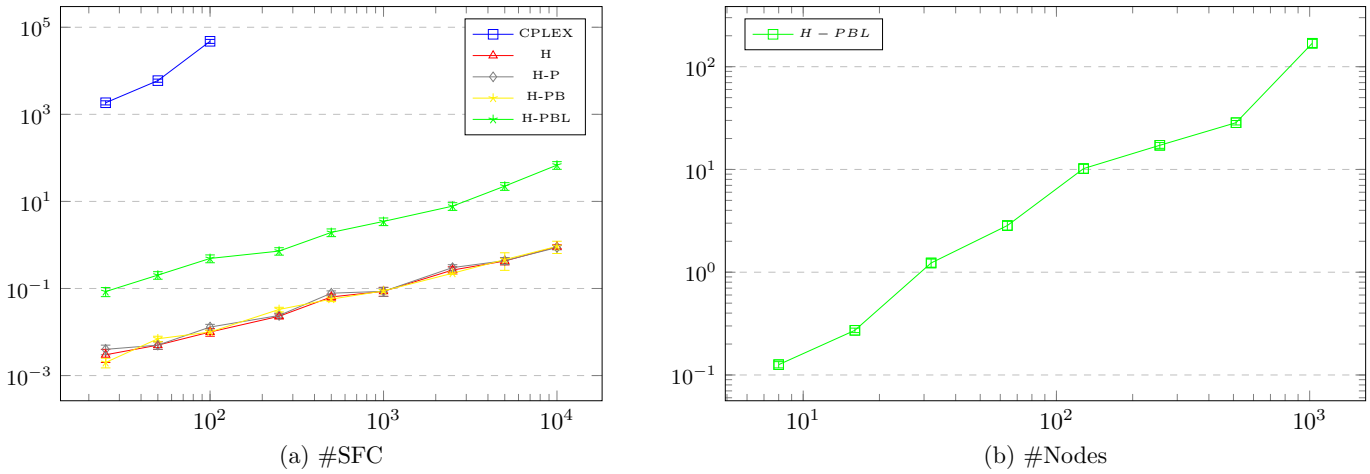
3. **Optimality Gap** To refine the performance analysis of the solutions generated by our heuristic method, we evaluate the optimality gap as summarized in Table 4.3. This metric measures the distance between the solutions generated by our heuristic and the optimal solution produced by the exact method. Our method achieves a latency gap of 11%, which is significantly commendable as compared to the literature, where the latency gap typically ranges between 13% and 14% [48, 49]. Interestingly, each optimization, whether it is parallelization, online learning, or load balancing, contributes to its own margin of improvement. However, parallelization reduces the most the optimality gap (over one half), which is due to its strong tied to micro-services approach.

	H	$H-P$	$H-PB$	$H-PBL$
Optimality gap (%)	43	19	17	11

Table 4.3: Optimality gap for different versions of heuristics

4. **Computation Time as a Function of SFC Number**

   Figure 4.3.a illustrates the evolution of computation time according to the SFC number, ranging from 25 to 10,000, considering different versions of our heuristic as well as the exact method $CPLEX$. For the latter, the results are limited in range due to the exponential increase in computation time. The evolution of the computation time follows a logarithmic shape as the size of the instances increases. It is worth noting that the



(a) #SFC (b) #Nodes

Figure 4.3: Computation time (seconds) as a function of the number of (a) SFC and (b) nodes

heuristic approach $H - PLB$ is on average 20,000 times faster than the exact method $CPLEX$. We can also observe that two distinct categories of computation time also emerge: the H, $H - P$ and $H - PB$ approaches, which present a computation time, on average, 40 times faster than the $H - PBL$ approach. This difference is due to the integration of online learning in $H - PBL$, which requires multiple deployment iterations to optimize the deployment order. As for the impact of parallelization in the $H - P$ version and load balancing in the $H - PB$ version, they seem to have a minimal, if any, impact on the computation time as compared to the H and $H - PL$ versions, respectively.

5. **Computation Time as a Function of Node Number**

   Figure 4.3.b presents the evolution of the computation time as a function of the infrastructure size, defined by the number of nodes, from 8 to 1024, with a fixed number of SFC and available space, fixed to and of 30 and +150% of the number of micro-services, respectively. We observe here a clear increase in computation time which is due to the modified *Eppstein* algorithm used for computing the k shortest paths, whose complexity depends on the number of nodes in the infrastructure. It is worth noting that although this increase is linked to the size of the infrastructure, our approach outperforms that of the exact method of three orders of magnitude, being about 20,000 times faster for the scenario involving 128 nodes. Even when we are on very large infrastructure instances (512 or 1024 nodes) the execution time, roughly a few tens of seconds, proves to be acceptable for an operational deployment and highly scalable.

6. **Load Balancing Quality**

   Figure 4.2 illustrates the Jain's fairness index for the two approaches $H - P$ and $H - PBL$ according to the Extra memory, Nodes number and SFC number. A clear distinction in terms of load balancing is observed in favor of the $H - PBL$ approach for all scenarios. Nevertheless, we note a convergence of the index in cases where the scenarios are restricted: (i) when the extra memory is low, and (ii) when the number of SFC increases. Indeed, in this scenario where the infrastructure is highly constrained, load balancing becomes a challenging task, especially considering the stringent latency requirements for all SFC. This limits our algorithm ability to balance, thus prioritizing latency compliance. Finally, as the number of nodes rises, Jain's indices of both approaches converge. Indeed, an infrastructure, composed of significantly large node numbers and a static capacity, can still achieve more balanced deployment without specific techniques.

Chapter 5

Micro-services SFC Use Cases related to the MOSAICO Project

As part of the MOSAICO project, several network services have been proposed by project participants. The architecture chosen for these services is a micro-services architecture. Examples include the Cloud Gaming session detector, the L4S monitoring service and the L4S security detector micro-services. Based on these different services, we can setup SFCs leveraging the micro-services approach, responding to a certain need. In what follows, we propose three different uses cases linked to our projects, by presenting different SFCs. It should be noticed, however, that some of the micro-services linked to these various SFCs are implemented using P4 technology, which means that they are placed in a predefined, fixed position on certain network servers.

5.1 SFC 1: Low Latency Cloud Gaming Forwarding

SFC 1 is a virtual function service chain designed to optimize and enhance the Cloud Gaming (CG) experience. This SFC integrates two major components: the Cloud Gaming Session Detector and the L4S Micro-Services Architecture. The Cloud Gaming Session Detector aims to identify and classify real-time gaming traffic to treat it as a low-latency (LL) service. To achieve this, it is deployed at the edge of the network, an essential strategy for countering the Bufferbloat phenomenon. The methodology relies on the use of a decision tree (DT)-based classifier that determines whether a flow is associated with Cloud Gaming. This classification process relies on 12 features, calculated over a 33 ms time window, which are derived from packet sizes and intermediate arrival times (IATs). Architecturally, several micro-services are integrated: a "Probe" which listens to a specific network interface and calculates the characteristics of each pair of IP addresses, a "Load Balancer" which distributes the load among the calculation nodes using a round robin strategy, a "Classifier DT" which classifies a time window according to the twelve selected characteristics, and finally an "Aggregator" which associates to each pair of IP addresses its number of time windows labeled "CG". On the other hand, the L4S micro-services architecture aims to redefine traditional L4S architecture by adopting a micro-services-based approach, offering greater flexibility and ease of upgradability. This architecture includes a Packet Classifier that assigns packets to the LL or Best Effort (BE) queue, using machine learning methods to detect LL session behavior. It also integrates a Queuing System that manages the queuing of LL or BE packets to meet low-latency requirements. The "AQM Computations" micro-service is responsible for calculating indicators to determine whether a packet should be transmitted, marked or dropped, with the possibility of considering and replacing several AQMs as required. Finally, "Packet Decision" makes the final decision on whether to transmit, mark or discard packets. SFC 1 is a solution that combines detection and classification of game traffic with an L4S architecture based on micro-services. This combination optimizes the Cloud Gaming experience, guaranteeing low latency and efficient traffic management. The modularity offered by the micro-services approach also enables scalability and adaptability to different scenarios and network requirements.

5.2 SFC 2: Securingg Cloud Gaming Traffic

SFC 2 is a solution focused on ensuring optimal traffic management with a particular emphasis on security, especially for cloud gaming applications. It ensures that traffic not only flows smoothly and efficiently but is also protected against potential threats. At the start of this chain, the Cloud Gaming Session Detector plays an essential role in filtering incoming traffic. Based on advanced algorithms, it accurately determines whether a specific flow is associated with Cloud Gaming. This initial distinction is crucial in ensuring that relevant flows are treated with the necessary priority, while paving the way for enhanced security measures. Security is at the heart of SFC 2, and this is where the L4S Security Detector comes in. This component goes beyond simple traffic management to focus on proactive threat detection. Using advanced machine learning techniques, it scans traffic to identify abnormal or malicious behavior, ensuring that traffic is not only managed, but also secured against potential attacks. Finally, L4S Monitoring services ensure continuous traffic monitoring. By capturing detailed metrics in real time, this component offers a granular view of traffic behavior, enabling rapid detection of anomalies and providing essential information for real-time adjustments. In essence, SFC 2 combines traffic management, enhanced security and continuous monitoring to ensure that cloud gaming applications enjoy a seamless user experience while being protected against emerging threats.

5.3 SFC 3: Optimized Cloud Gaming Traffic Management

SFC 3 is designed as an integrated solution to optimize traffic management, particularly for cloud gaming applications. It is built around a synergy between traffic classification, management and monitoring to ensure a smooth, responsive user experience. At the heart of this chain is the Cloud Gaming Session Detector, which acts as a first line of the micro-service chain by identifying and classifying gaming traffic in real time. Leveraging a sophisticated decision-tree classifier, it analyzes the intrinsic characteristics of the traffic to determine its nature. This ability to distinguish gaming traffic from other types of traffic is essential to ensure that network resources are allocated appropriately. Once traffic has been classified, this function takes over traffic management using a micro-services-based L4S approach. It breaks down the L4S architecture into several independent micro-services, including the classifier, the queuing system, AQM calculations and packet decision. This modularity allows greater flexibility and adaptability to different transport protocols. Finally, real-time monitoring is provided by L4S monitoring service. This component ensures that traffic behavior is constantly monitored, providing valuable information that can be used to quickly detect anomalies and adjust traffic management accordingly. In short, SFC 3 is a holistic solution that aims to ensure that cloud gaming traffic is handled with the necessary priority and efficiency, while being constantly monitored to ensure optimum performance.

Chapter 6

Conclusion

This work has highlighted the crucial importance of optimal placement and micro-service chain management in the context of modern networks, particularly for low-latency (LL) Service Function Chains (SFCs). Thanks to a rigorous optimisation modeling and a heuristic approach, we were able to explore the associated challenges in depth and propose viable solutions. Evaluation scenarios played a key role in validating the performance of our models under a variety of conditions, reinforcing their relevance and applicability. In addition, the practical implementation of these concepts, accessible for replication, demonstrated their effectiveness and robustness. The specific use cases of the MOSAICO project illustrate the relevance and applicability of these concepts in real-life scenarios.

Bibliography

[1] Dejene BoruOljira et al. "Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture". In: *SIGCOMM Comput. Commun. Rev.* 50.2 (May 2020), pp. 37–44. ISSN: 0146-4833. DOI: 10.1145/3402413.3402419. URL: https://doi.org/10.1145/3402413.3402419.

[2] Karamjeet Kaur, Veenu Mangat, and Krishan Kumar. "A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture". In: *Computer Science Review* 38 (2020). ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100298.

[3] Jie Sun et al. "A survey on the placement of virtual network functions". In: *Journal of Network and Computer Applications* 202 (2022), p. 103361. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103361.

[4] B. Addis, G. Carello, and M. Gao. "On a virtual network functions placement and routing problem: Some properties and a comparison of two formulations". In: *Networks* 75.2 (Nov. 2019), pp. 158–182.

[5] A. Gupta et al. "On service-chaining strategies using Virtual Network Functions in operator networks". In: *Computer Networks* 133 (2018), pp. 1–16.

[6] R. Riggio et al. "Scheduling Wireless Virtual Networks Functions". In: *TNSM* 13.2 (2016), pp. 240–252.

[7] R. Cohen et al. "Near optimal placement of virtual network functions". In: *INFOCOM*. 2015, pp. 1346–1354.

[8] W. Rankothge et al. "Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms". In: *TNSM* 14.2 (2017), pp. 343–356.

[9] M. Casado et al. "Virtualizing the Network Forwarding Plane". In: *Proceedings of PRESTO'10*. ACM, 2010.

[10] M. C. Luizelli et al. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *IFIP/IEEE IM*. 2015, pp. 98–106.

[11] M. C. Luizelli et al. "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining". In: *Computer Communications* 102.C (2017), pp. 67–77.

[12] H. Moens and F. D. Turck. "VNF-P: A model for efficient placement of virtualized network functions". In: *IFIP/IEEE CNSM*. 2014, pp. 418–423.

[13] Leila Askari et al. "Virtual-network-function placement for dynamic service chaining in metro-area networks". In: May 2018, pp. 136–141. DOI: 10.23919/ONDM.2018.8396120.

[14] Hédi Nabli. "An overview on the simplex algorithm". In: *Applied Mathematics and Computation* 210 (2009), pp. 479–489. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2009.01.013.

[15] G.D. Forney. "The viterbi algorithm". In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278. DOI: 10.1109/PROC.1973.9030.

[16] Anish Hirwe and Kotaro Kataoka. "LightChain: A lightweight optimisation of VNF placement for service chaining in NFV". In: *IEEE NetSoft Conference and Workshops (NetSoft)*. 2016, pp. 33–37. DOI: 10.1109/NETSOFT.2016.7502438.

[17] Akshay Gadre, Anix Anbiah, and Krishna Sivalingam. "Centralized approaches for virtual network function placement in SDN-enabled networks". In: *EURASIP Journal on Wireless Communications and Networking* 2018 (Aug. 2018). DOI: 10.1186/s13638-018-1216-0.

[18] S Meena Kumari and N Geethanjali. "A survey on shortest path routing algorithms for public transport travel". In: *Global Journal of Computer Science and Technology* 9 (2010), pp. 73–76.

[19] David Eppstein. "Finding the k shortest paths". In: *SIAM Journal on computing* 28.2 (1998), pp. 652–673.

[20] Yang Zhang et al. "ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining". In: *SOSR*. ACM, 2017, pp. 143–149.

[21] Linux Sofware Foundation. *Data Plane Development Kit*. URL: https://www.dpdk.org/ (visited on 05/07/2020).

[22] C. Sun et al. "NFP: Enabling Network Function Parallelism in NFV". In: *SIGCOMM*. ACM, Aug. 2017, pp. 43–56.

[23] S. Xie, J. Ma, and J. Zhao. "FlexChain: Bridging Parallelism and Placement for Service Function Chains". In: *TNSM* 18.1 (2021), pp. 195–208.

[24] I-Chieh Lin, Yu-Hsuan Yeh, and Kate Ching-Ju Lin. "Toward Optimal Partial Parallelization for Service Function Chaining". In: *IEEE/ACM Transactions on Networking* 29.5 (2021), pp. 2033–2044. DOI: 10.1109/TNET.2021.3075709.

[25] S. R. Chowdhury et al. "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges". In: *Network* 33.3 (2019), pp. 168–176.

[26] G. Liu et al. "Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions". In: *SIGCOMM*. 2018, pp. 504–517.

[27] A. Bremler-Barr, Y. Harchol, and D. Hay. "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions". In: *SIGCOMM*. ACM, 2016.

[28] Dharmendra Shadija, Mo Rezai, and Richard Hill. "Microservices: Granularity vs. Performance". In: Dec. 2017, pp. 215–220. DOI: 10.1145/3147234.3148093.

[29] Maziar Nekovee et al. "Towards AI-enabled Microservice Architecture for Network Function Virtualization". In: *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*. 2020, pp. 1–8. DOI: 10.1109/ComNet47917.2020.9306098.

[30] H. Hassan, M. Jammal, and A. Shami. "Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms". In: *Network* 33.2 (2019), pp. 202–210.

[31] A. Sheoran et al. "Contain-ed: An NFV Micro-Service System for Containing e2e Latency". In: vol. 47. 5. ACM, Aug. 2017, pp. 12–17.

[32] Z. Meng et al. "MicroNF: An Efficient Framework for Enabling Modularized Service Chains in NFV". In: *JSAC* 37.8 (2019), pp. 1851–1865.

[33] S. Chowdhury et al. "A Disaggregated Packet Processing Architecture for Network Function Virtualization". In: *JSAC* 38.6 (2020).

[34] A. Mouaci et al. "Virtual Network Functions Placement and Routing Problem: Path formulation". In: *IFIP Networking*. 2020, pp. 55–63.

[35] Bernardetta Addis, Meihui Gao, and Giuliana Carello. "On the complexity of a Virtual Network Function Placement and Routing problem". In: *Electronic Notes in Discrete Mathematics* 69 (2018), pp. 197–204. ISSN: 1571-0653. DOI: https://doi.org/10.1016/j.endm.2018.07.026.

[36] Jose Ordonez-Lucena et al. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". In: *IEEE Communications Magazine* 55.5 (2017), pp. 80–87. DOI: 10.1109/MCOM.2017.1600935.

[37] Richelle Adams. "Active Queue Management: A Survey". In: *IEEE Communications Surveys Tutorials* 15.3 (2013), pp. 1425–1476. DOI: 10.1109/SURV.2012.082212.00018.

[38] Margarida Ferreira et al. "Counterfeiting Congestion Control Algorithms". In: *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. HotNets '21. Virtual Event, United Kingdom: Association for Computing Machinery, 2021, pp. 132–139. ISBN: 9781450390873. DOI: 10.1145/3484266.3487381. URL: https://doi.org/10.1145/3484266.3487381.

[39] Szilveszter Nádas et al. "A Congestion Control Independent L4S Scheduler". In: *Proceedings of the Applied Networking Research Workshop*. ANRW '20. Virtual Event, Spain: Association for Computing Machinery, 2020, pp. 45–51. ISBN: 9781450380393. DOI: 10.1145/3404868.3406669. URL: https://doi.org/10.1145/3404868.3406669.

[40] C. Bouras et al. "Enhancing the DiffServ architecture of a simulation environment". In: *Proceedings. Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*. 2002, pp. 108–115. DOI: 10.1109/DISRTA.2002.1166896.

[41] Paola Cappanera, Federica Paganelli, and Francesca Paradiso. "VNF placement for service chaining in a distributed cloud environment with multiple stakeholders". In: *Computer Communications* 133 (Oct. 2018). DOI: 10.1016/j.comcom.2018.10.008.

[42] Behrooz Farkiani et al. "Prioritized Deployment of Dynamic Service Function Chains". In: *IEEE/ACM Transactions on Networking* 29.3 (2021), pp. 979–993. DOI: 10.1109/TNET.2021.3055074.

[43] Amir Mohamad and Hossam S. Hassanein. "PSVShare: A Priority-based SFC placement with VNF Sharing". In: *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2020, pp. 25–30. DOI: 10.1109/NFV-SDN50289.2020.9289837.

[44] Hichem Magnouche, Guillaume Doyen, and Caroline Prodhon. "Leveraging Micro-Services for Ultra-Low Latency: An optimization Model for Service Function Chains Placement". In: *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 2022, pp. 198–206. DOI: 10.1109/NetSoft54395.2022.9844040.

[45] Zhi-Quan Luo and Jong-Shi Pang. "Analysis of iterative waterfilling algorithm for multiuser power control in digital subscriber lines". In: *EURASIP Journal on Advances in Signal Processing* 2006 (2006), pp. 1–10.

[46] M. Dianati, X. Shen, and S. Naik. "A new fairness index for radio resource allocation in wireless networks". In: *IEEE Wireless Communications and Networking Conference, 2005*. Vol. 2. 2005, 712–717 Vol. 2. DOI: 10.1109/WCNC.2005.1424595.

[47] Giulio Bartoli et al. "An Efficient Resource Allocation Scheme for Applications in LR-WPANs Based on a Stable Matching With Externalities Approach". In: *IEEE Transactions on Vehicular Technology* 68.6 (2019), pp. 58–59. DOI: 10.1109/TVT.2019.2909136.

[48] Faizul Bari et al. "Orchestrating Virtualized Network Functions". In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 725–739. DOI: 10.1109/TNSM.2016.2569020.

[49] Mohammad M. Tajiki et al. "Joint Energy Efficient and QoS-Aware Path Allocation and VNF Placement for Service Function Chaining". In: *IEEE Transactions on Network and Service Management* 16.1 (2019), pp. 374–388. DOI: 10.1109/TNSM.2018.2873225.