Leveraging Micro-Services for Ultra-Low Latency: An optimization Model for Service Function Chains Placement

Hichem Magnouche, Guillaume Doyen, Caroline Prodhon

Leveraging Micro-Services for Ultra-Low Latency: An Optimization Model for Service Function Chains Placement

Hichem Magnouche*, Guillaume Doyen†, Caroline Prodhon*
*LIST3N, University of Technology of Troyes, Troyes, France, {first.last}@utt.fr
†OCIF - IRISA (UMR CNRS 6074), IMT Atlantique, Rennes, France, guillaume.doyen@imt-atlantique.fr

***Abstract*—The evolution of the Internet tends toward ever requiring lower latency services. Cloud robotics or drone piloting are service use-cases in which the latency of traffic cannot exceed a few milliseconds. Reducing the latency can be achieved through several means, and micro-services deployed over virtual infrastructures appears as a promising way by enabling service chain reductions, micro-function mutualization and parallelism. However, the placement and routing of such components appears as an harder task to achieve as compared to monolithic approaches of the state of the art. Consequently, we propose in this paper a comprehensive optimization model in charge of placing micro-services in a virtualized network infrastructure, under ultra-low latency constraints while preserving resource consumption. By challenging our model with several realistic scenarios in terms of topology and service function chains (SFC), we demonstrate to what extent it improves the overall performance of SFC by especially minimizing the gap between the expected latency and the actual one, as compared to several competitors, thus making it a well-fitted approach for ultra-low latency services.**

***Index Terms*—micro-services, Placement and Routing, Optimization, Orchestration.**

I. Introduction

The Internet is ever evolving according to novel usage and needs expressed from end-users and the evolution of the technologies that it leverages. Reducing traffic latency is a core concern for today's cloud gaming and interactive video systems, and especially for emerging services such as cloud robotics, metavers, augmented and virtual reality and haptic internet. For these future services, being able to carry their traffic with no more than a few millisecond is a prerequisite to enable them bringing the expected quality of service to the end-users and consequently allow their adoption.

Reducing the Internet latency is not an issue targeting a single component or service in the end-to-end delivery of traffic, but rather, it can be achieved by considering all the points that can be inflected to globally improve the traffic forwarding delay. Some relevant work has been proposed during the last few years on novel congestion control algorithms (e.g. TCP-Prague) or active queue management strategies (e.g. L4S) to improve the sole packet forwarding performance in routing elements. Besides, network traffic does not only cross routers but also a set of Network Functions (NF) such as firewalls or NAT, all bringing a latency penalty, and thus being subject to improvement.

The Network Function Virtualization (NFV) is now acknowledged as the standard means to deploy such NF. Although its standard usage consists to deploy monolithic NF, it has been recently identified that the split of such functions into micro-services could be highly beneficial to reduce the usage of resources while improving the overall performance of the service delivery. Indeed, micro-services enable the reduction of Service Function Chains (SFC) by removing redundant micro-functions (e.g. packet parsing), mutualizing micro-functions belonging to different NF (e.g. packet classifiers) or parallelizing the processing on independent packet fields. However, in the context where the number of micro-services is roughly one order of magnitude larger than that of network functions, and service chains must be restructured to actually leverage the power of micro-services, the question of their efficient orchestration appears as an open issue. Indeed, standard orchestration algorithms, in charge of placement and routing of monolithic NF, are no longer applicable given these particular characteristics (i.e. parallelization, mutualization and scale) exposed by microservices since they consider SFC service chains as linear and therefore do not handle forks and merges.

To that aim, we propose in this paper an optimization model which allows a routing and placement orchestration function to operate on SFC that will eventually be deployed as micro-services while preserving ultra-low latency constraints. Our model intrinsically integrates the capability to both mutualize and parallelize micro-services. Furthermore, it is comprehensive, meaning that it encompasses the set of constraints that must be considered to efficiently achieve a satisfying deployment of service function chains and especially the resource it consumes. In order to validate the relevance of our proposal, we implemented it into the CPLEX solver and we performed an exhaustive performance evaluation in which we considered a realistic telco topology, some set of SFC extracted from realistic use-cases and we demonstrate to what extent it outperforms current approches from the state of the art leveraging monolithic NF but also micro-services leveraging the sole mutualization feature.

The paper is organized as follows. Section II presents a literature review of the recent approaches for service function orchestration. Section III presents our model and its math-

ematical formulation. In Section IV, tests on the model are developed and analyzed. Finally, Section V concludes the paper.

II. Related Work

A large literature covers the question of placement and chaining of Virtual Network Functions (VNF) in NFV as a means to ensure a prescribed performance level while ensuring a satisfying usage of resources. Besides, several improvements on VNF have been recently proposed to further enhance the overall latency induced by service chains. These are especially VNF parallelization and micro-service architectures. As a set of fields closely related to our work, we subsequently review them and state to what extent our approach fills the gap to provide efficient orchestration means for the placement and chaining of restructured SFC composed of micro-services fulfilling ultra-low latency constraints.

A. VNF Placement and Routing

The VNF Placement and Routing (VNF-PR) problem has attracted a lot of attention for a decade [1]. It exhibits 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 the majority 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 [2], [3], while some others develop a heuristic algorithm [4], [5] or a meta-heuristic [6]. In [2], Addis *et al.* consider the placement and routing problem of VNF with only one type of VNF for all services thus making it restricted. In Casado *et al.* [7], the authors consider the problem in a similar way with a single type of VNF but they use a heuristic algorithm to solve it and consequently reduce the computation time.

In [8], Luizelli *et al.* deal with the VNF-PR issue by using an ILP composed of 9 constraints reflecting the real situation of a network. In this work, close from ours regarding the operational features it considers while being restricted to monolithic VNF and not micro-services, 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 especially break down the problem into sub-problems and try to optimize them. To that aim, they develop a Variable Neighborhood Search algorithm that tries to find a near-optimal solution for each sub-problem. Similarly, in [9], the authors keep the same model but they use a meta-heuristic-based algorithm and they demonstrate to what extent it is more efficient. In [10], the authors minimize the number of activated nodes instead of VNF instances which shows the relevance of the cost metric choice in the formulation of the VNF-PR issue.

B. VNF Parallelization

In order to optimise the latency of SFC, some contributions propose to parallelize the execution of VNF. [11] 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 services function 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 [12]. Subsequently, among the most relevant proposals, [13] 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 [11], [13] also proposes a parallelization table allowing to state which VNF can be parallelized. Finally, [14] proposes a parallelization approach that addresses the weaknesses of [11] and [13] by parallelizing VNF placed on the same node without any condition on the whole SFC, thus preventing the loss in copy/merge latency on different nodes. Besides, it also allows better agility because the parallelism is decided after deployment and does not require the whole SFC to be on the same node.

C. Micro-services: Architectures and Orchestration

For a couple of years, the benefits of micro-services have been demonstrated [15] 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 [16] and Openbox [17], all bringing specific performance enhancements which mainly leverage lightweight virtualization with containers and zero-copy of packets with DPDK [12].

Concerning the placement and chaining of micro-services, in [18], 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. However, no resolution algorithm is developed to demonstrate the capability of the approach to be considered in real situations. In [19], 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

that exhibit a heavy communication and should therefore be deployed in a close way. [20] 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 upon reception in order to re-use micro-services already deployed in the infrastructure and also to re-factor micro-services when possible; (2) a micro-services placement which tries to consolidate them on the same node (i.e. with the largest number of micro-services related to the same VNF); 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 of grouping two identical micro-services belonging to the same SFC in a single one. It makes 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 [20], [21] as part of the functional decomposition of VNFs into microservices. 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 [20] who focuses on this particular enhancement. Besides, [21] 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 [1], [2], [7]–[10], VNF parallelism [11], [13], [14] or micro-services decomposition and mutualisation [15], [18], [21]. As highlighted in [20], we also state that the micro-services approach used without mutualization, exhibits a negative impact on the global latency of SFC. In a similar way, the parallelism, used in a non-optimal way, can further increase the latency instead of reducing it [14] and to the best of our knowledge, it has only been considered for monolithic VNF. Consequently, we propose in this paper a comprehensive optimization model specific for a robust and flexible micro-services orchestration leveraging both mutualization and optimized parallelism.

III. A Micro-service Orchestration Model

In this section, we present the micro-services placement and chaining orchestration problem 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 internal and external parallelization of micro-services according to the infrastructure setup (i.e. number of nodes and links as well as their available resources).

A. 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.

The optimization objective chosen in our work is the minimization of the latency delay on service requests according to their latency specification. Indeed, we consider that:

- The available nodes have no usage cost but the available resources expressed in terms of CPU and memory are limited;
- The available links have no usage cost but the flow rate on each one is limited;
- The memory required for the deployment of micro-services is similar for all micro-services;
- Sharing the usage of micro-services between different SFC is not allowed.

B. 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 Mixed Integer Linear Programming (MILP) translating the problem of placement and chaining of micro-services, which is solved by taking into consideration the parallelism parameters among other constraints.

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 [14], [20]. To understand the pre-processing, let us take the example depicted in Figure 1, where a request q is made up of an original SCF containing 5 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 4 micro-services as pictured in the second part of Figure 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.

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 II-B, 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.

This 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 2 shows 3 different possible deployments for request q, as introduced in Figure 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. 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 time, a single group (g_1) is assigned to each of nodes B, C and D. In this case also, a fork cost is accounted because the latter is external. Beyond, this SFC example

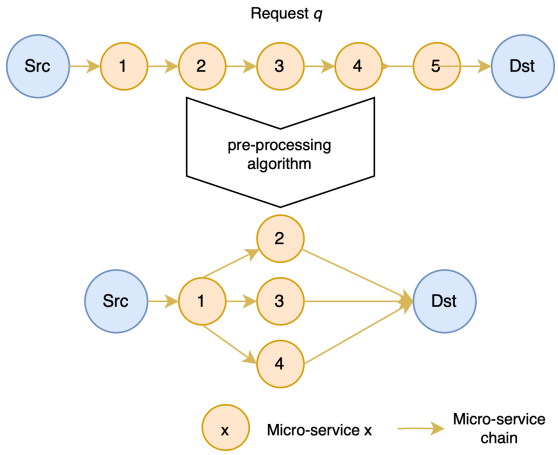

Fig. 1: Pre-processing example

and underlying infrastructure may lead to other possibilities, actually depending on the memory capacities of each node.

C. Mathematical Formulation

Table I 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 longest path, managing the notion of group as introduced in section III-B2. 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 gap latency for each request $q \in Q$.

This leads to the following model, consisting in minimizing an objective function relative to the lag behind expected

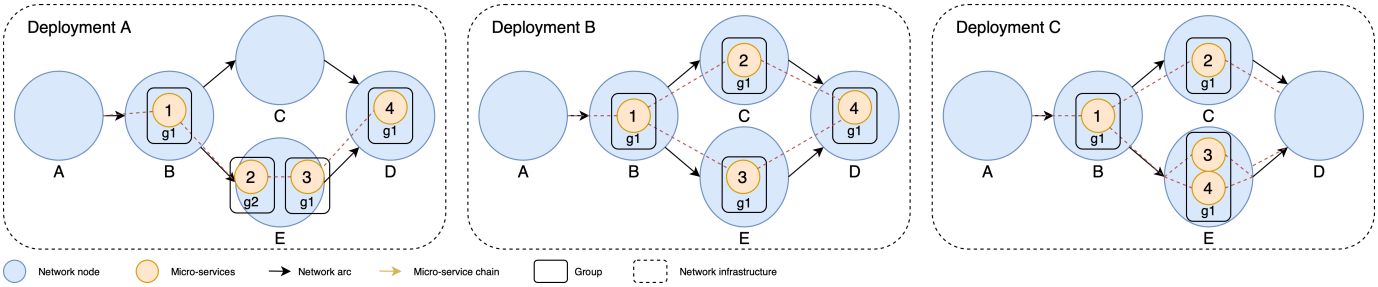

Fig. 2: A deployment example

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 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 I: Notation table

latency and subject to the two dozen of constraints that we describe subsequently.

The objective function (1) minimizes the latency delay, r_q of request q. The second term is relative to additional variables used in the constraints.

$$\min \sum_{q \in Q} r_q + \sum_{n \in N} \sum_{q \in Q} \sum_{f \in F_q} \sum_{h \in F_q} y_{nfq} + bf_{nqh} + p_{nfq} \quad (1)$$

Constraints (2) and (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 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.

$$\sum_{e \in N} a_{enq} \geq a_{nsq} \qquad \forall q \in Q, n, s \in N \neq S_q, D_q \quad (2)$$

$$\sum_{s \in N} a_{nsq} \geq a_{enq} \qquad \forall q \in Q, n, e \in N \neq S_q, D_q \quad (3)$$

In order to avoid cycles in the chain and to force it to be elementary, constraints (4) ensure that each node has an order in the chain and that this order is respected.

$$o_{n_1 q} \geq o_{n_2 q} + a_{n_2 n_1 q} - m(1 - a_{n_2 n_1 q}) \qquad \forall n_1, n_2 \in N \quad (4)$$

The set of constraints (5) to (10) allow to set variables y_{nfq} to 1 in case function f is placed on node n or before for request q.

$$y_{nfq} \geq x_{nfq} \qquad \forall n \in N, f \in F, q \in Q \quad (5)$$

$$a_{n_1 n_2 q} - 1 + x_{n_1 f q} - x_{n_2 f q} \leq y_{n_2 f q} \qquad \forall n_1, n_2 \in N, \forall q \in Q, \forall f \in F_q \quad (6)$$

$$y_{n_1 f q} - 1 + a_{n_1 n_2 q} \leq y_{n_2 f q} \qquad \forall n_1, n_2 \in N, \forall q \in Q, \forall f \in F_q \quad (7)$$

$$\sum_{n_1 \in N} y_{n_1 f q} + a_{n_1 n q} \leq p_{nfq} \qquad \forall n \in N, \forall q \in Q, \forall f \in F_q \quad (8)$$

$$x_{nfq} \leq p_{nfq} \qquad \forall n \in N, \forall q \in Q, \forall f \in F_q \quad (9)$$

$$y_{nfq} \leq p_{nfq} \qquad \forall n \in N, \forall q \in Q, \forall f \in F_q \quad (10)$$

More specifically, constraints (5) allow to set variables y_{nfq} to 1 if x_{nfq} equal to 1. Constraints (6) aim at setting variables $y_{n_2 f q}$ to 1 if the relative function f is placed on node n_1, preceding node n_2 and concerning request q. Constraints (7) allow to set variables $y_{n_2 f q}$ to 1 if $y_{n_1 f q}$ equal to 1 and if n_1 is

preceding node n_2 concerning request q. The set of constraints (8), (9) and (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 (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.

$$y_{nfq} \geq y_{D_q fq} \qquad \forall q \in Q, \forall f \in F_q, \forall n \in N \tag{11}$$

Constraints (12) to (14) 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.

$$\gamma_{nqgf_1} + \gamma_{nqgf_2} \leq T_{f_1 f_2} + 1 \quad \forall n \in N, \forall q \in Q, \forall g, f_1, f_2 \in F_q \tag{12}$$

$$\gamma_{nqgf_1} + \gamma_{nqgf_2} \leq P_{f_1 f_2 q} + 1 \quad \forall n \in N, \forall q \in Q, \forall g, f_1, f_2 \in F_q \tag{13}$$

$$\sigma_{nqgh} \geq \lambda_f(\gamma_{nqgf} + l_{nn_2qh} - 1) \quad \forall q \in Q, f \in F_q, \forall n, n_2 \in N, \forall h, g \in F_q \tag{14}$$

Constraints (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.

Constraints (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.

$$\sum_{n_1 \in N} \sum_{n_2 \in N} l_{n_1 n_2 qh} * \Delta_{n_1 n_2} + \sum_{n \in N} \sum_{g \in Fq} \sigma_{nqgh} + \sum_{n \in N} bf_n * CB - \Lambda_q \leq r_q \quad \forall q \in Q, \forall h \in F_q \tag{15}$$

Constraints (16) and (17) respectively ensure that the CPU resources and the available flow rate on each link are respected for each node.

$$\sum_{q \in Q} \sum_{f \in F_q} x_{nfq} * Rm_f \leq R_n \qquad \forall n \in N \tag{16}$$

$$\sum_{q \in Q} \sum_{f \in F_q} a_{n_1 n_2 q} * Db_q \leq Da_{n_1 n_2} \qquad \forall n_1, n_2 \in N \tag{17}$$

Finally, the model includes the following types of variables.

$$x_{nfq} \in \{0,1\} \qquad \forall n \in N, f \in F, q \in Q \tag{18}$$

$$y_{nfq} \in \{0,1\} \qquad \forall n \in N, f \in F, q \in Q \tag{19}$$

$$a_{n_1 n_2 q} \in \{0,1\} \qquad \forall n_1, n_2 \in N, q \in Q \tag{20}$$

$$l_{n_1 n_2 qh} \in \{0,1\} \qquad \forall n_1, n_2 \in N, q \in Q, h \in F_q \tag{21}$$

$$\gamma_{nqfg} \in \{0,1\} \qquad \forall n \in N, q \in Q, f \in F, g \in F_q \tag{22}$$

$$bf_{nqh} \in \{0,1\} \qquad \forall n \in N, q \in Q, h \in F_q \tag{23}$$

$$p_{nfq} \in \{0,1\} \qquad \forall n \in N, q \in Q, f \in F \tag{24}$$

$$r_q \geq 0 \qquad \forall q \in Q \tag{25}$$

$$\sigma_{nqgh} \geq 0 \qquad n \in N, q \in Q, g \in F_q, h \in F_q \tag{26}$$

$$o_{nq} \geq 0 \qquad n \in N, q \in Q \tag{27}$$

Besides, the model also includes some additional constraints, not detailed here due to space constraints. Briefly, they 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-services related to its request; (i) each deployed micro-services 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 [2], [22]. 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 mutualisation option, it reduces to the classical VNF placement and chaining problem that is already NP-Complete [23].

IV. Evaluation

A. 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 reproductibility. The correctness checking has consisted in verifying the following bias: (1) absence of circuits, (2) deployment of all micro-services related to each request, (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 a 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.

B. Evaluation Scenarios

Our evaluation scenarios aims at (1) understanding the performance of our model in realistic situations and (2) comparing it with current competitors. We have especially considered those standing for acknowledged approaches in the literature, namely: (a) Monolithic VNF placement (*Mono*); (b) Micro-services placement with neither mutualization nor parallelization enhancements (*Micro*); (c) Micro-services placement with mutualization enhancement (*MicroM*), and finally (d) Micro-services placement with both mutualization and parallelization enhancements (*MicroMP*); the latter standing for our model.

All the parameters we considered in our evaluation are summarized in Table II and motivated subsequently. The implemented topology, extracted from the SNDlib[1] library, is

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

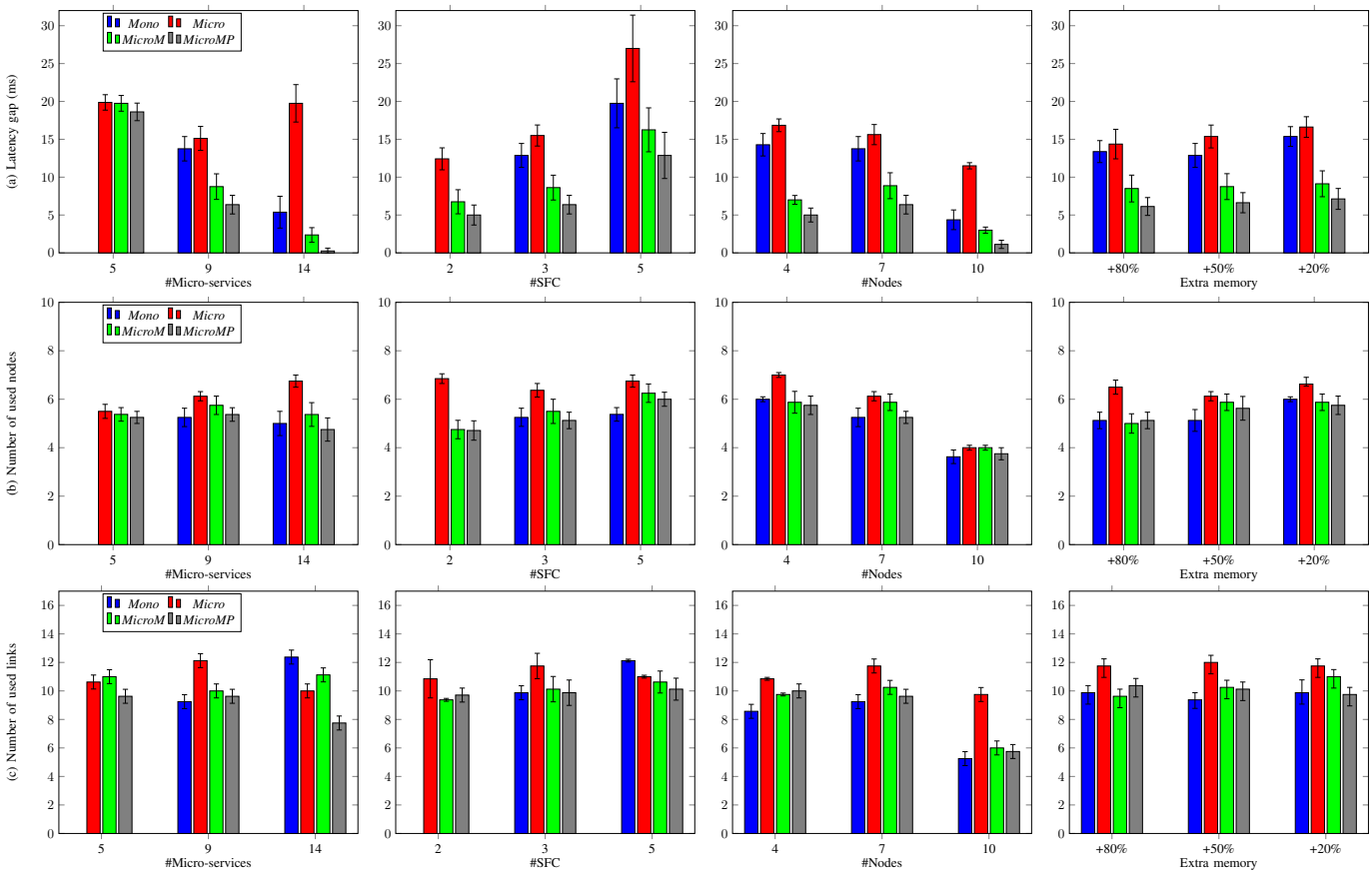

Fig. 3: Histogram of the different successful measurements featuring the performance of our model (*MicroMP*) against three competitors (*Mono*, *Micro* and *MicroM*). (a) Gap of latency between the SFC requirement and that computed by the placement algorithm; (b) Number of nodes required to place the 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.

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
VNF Proc. latency	4ms
#SFC	2-5
SFC length	5-14 micro-services
#Nodes	4-10
Node capacity	3-10 instances of micro-services

TABLE II: Evaluation parameters

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 [9], [22] which are composed of NF (Firewall, NAT, Traffic monitor and IPS) whose decomposition into microservices is acknowledged. As the core benefits of micro-services, we have considered the mutualization and parallelization tables illustrated in Table III, as defined in [20] and [11], 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 covers the scales provided in Table II for the number of nodes, SFC numbers, SFC lengths and nodes capacities. Overall, the results exposed subsequently required

	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 III: Parallelism (left) and mutualization (right) tables as defined in [20] and [11]

more than 30 hours of computation to encompass all cases. Finally, the prescribed latency for SFC ranges from 5 to 10ms, thus standing for those of realistic ultra-low latency use-cases.

C. Results Analysis

The different metrics we measure in our performance evaluation campaign aims 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; (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; and (3) the number of links activated for the deployment of all SFC, as depicted in third row of Figure 3.

1) Latency Benefit: The first line of Figure 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 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 4 micro-services on average [20]. 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 (e.g. #Micro-services = 5, #SFC = 2), the model cannot find a solution for *Mono*, despite a sufficient amount of infrastructure resources for their deployment. Indeed, the breakdown into microservices allows for better agility as compared to *Mono* because the components are lighter and memory space management is therefore easier to achieve.

D. Analysis of Computation Features

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

Table IV summarises the average computation features of CPLEX for the different instances. Each instance represents a configuration setup with the values presented in Table II, 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 the *Mono* approach, the optimal solutions were found much earlier (1,5 seconds on average) contrary to the *Micro* approach 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 the *Mono* approach is zero while that of *Micro* is around 45%. This indicates that the solutions obtained for the *Micro* approach can be

improved if we allowed more computation time. Nevertheless, with such a limited computation time, the *Micro* approach is penalized. By contrast, regarding the *MicroM* and *MicroMP* approaches, the computation time is below the limit for more than 55% of the cases. In comparison to the *Mono* approach, 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 still better solutions for the *MicroM* and *MicroMP* approach; (2) the switch to *MicroM* and *MicroMP* requires more computing time due to the complexity of the approach, but less than the *Micro* approach since there are fewer micro-services to deploy, thus demonstrating the reasonable cost of these approaches.

V. Conclusion and Future Work

The development of ultra-low latency applications has required the development of different models optimizing the placement and chaining of VNF, but also the transition to the micro-services approach in order to further reduce latency. However, considering the micro-services approach without optimization controversially augments the latency and the mutualization of redundant micro-services and their parallelization appears as the expected enhancement to reveal the actual performance gain of micro-services. In this paper, we have proposed a micro-services orchestration approach composed of a pre-processing algorithm and a comprehensive mathematical model allowing the placement and chaining of micro-services taking into account their mutualization and parallelism and having as a main objective the reduction of the SFC latency to reach strong prerequisites. With extensive computation of the model, we have demonstrated that it exhibits the best latency performance as compared to monolithic approaches and also various micro-services alternatives. However, the resolution of the model with an exact method is very expensive in computation time which leads us to foresee the realization, in our future work, of an approximate resolution method based on a heuristic algorithm. Beyond, we plan to develop a dynamic version which adapts the placement and chaining according to the evolution of service demands and the infrastructure resources.

Acknowledgment

This work is partially funded by the French ANR MOSAICO project, No ANR-19-CE25-0012.

References

[1] K. Kaur, V. Mangat, and K. Kumar, "A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture," *Computer Science Review*, vol. 38, 2020.

[2] B. Addis, G. Carello, and M. Gao, "On a virtual network functions placement and routing problem: Some properties and a comparison of two formulations," *Networks*, vol. 75, no. 2, pp. 158–182, 11 2019.

[3] A. Gupta, M. Farhan Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service-chaining strategies using virtual network functions in operator networks," *Computer Networks*, vol. 133, pp. 1–16, 2018.

[4] R. Riggio, A. Bradai, D. Harutyunyan, T. Rasheed, and T. Ahmed, "Scheduling wireless virtual networks functions," *TNSM*, vol. 13, no. 2, pp. 240–252, 2016.

[5] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *INFOCOM*, 2015, pp. 1346–1354.

[6] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *TNSM*, vol. 14, no. 2, pp. 343–356, 2017.

[7] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of PRESTO'10*. ACM, 2010.

[8] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE IM*, 2015, pp. 98–106.

[9] M. C. Luizelli, W. L. Da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, vol. 102, no. C, pp. 67–77, 2017.

[10] 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.

[11] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *SOSR*. ACM, 2017, pp. 143–149.

[12] L. S. Foundation. Data plane development kit. [Online]. Available: https://www.dpdk.org/

[13] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *SIGCOMM*. ACM, 08 2017, pp. 43–56.

[14] S. Xie, J. Ma, and J. Zhao, "Flexchain: Bridging parallelism and placement for service function chains," *TNSM*, vol. 18, no. 1, pp. 195–208, 2021.

[15] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-architecting nfv ecosystem with microservices: State of the art and research challenges," *Network*, vol. 33, no. 3, pp. 168–176, 2019.

[16] G. Liu, Y. Ren, M. Yurchenko, K. K. Ramakrishnan, and T. Wood, "Microboxes: High performance nfv with customizable, asynchronous tcp stacks and dynamic subscriptions," in *SIGCOMM*, 2018, p. 504–517.

[17] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *SIGCOMM*. ACM, 2016.

[18] H. Hassan, M. Jammal, and A. Shami, "Exploring microservices as the architecture of choice for network function virtualization platforms," *Network*, vol. 33, no. 2, pp. 202–210, 2019.

[19] A. Sheoran, P. Sharma, S. Fahmy, and V. Saxena, "Contained: An nfv micro-service system for containing e2e latency," vol. 47, no. 5. ACM, 08 2017, pp. 12–17.

[20] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "Microntf: An efficient framework for enabling modularized service chains in nfv," *JSAC*, vol. 37, no. 8, pp. 1851–1865, 2019.

[21] S. Chowdhury, A. Rahman, H. Bian, T. Bai, and R. Boutaba, "A disaggregated packet processing architecture for network function virtualization," *JSAC*, vol. 38, no. 6, 2020.

[22] A. Mouaci, E. Gourdin, I. LjubiĆ, and N. Perrot, "Virtual network functions placement and routing problem: Path formulation," in *IFIP Networking*, 2020, pp. 55–63.

[23] B. Addis, M. Gao, and G. Carello, "On the complexity of a virtual network function placement and routing problem," *Electronic Notes in Discrete Mathematics*, vol. 69, pp. 197–204, 2018.