VNF Placement and Dynamic NUMA node Selection through Core Consolidation at the Edge and Cloud

Taha Ben Salah, Marios Avgeris, Aris Leivadeas (*Senior Member, IEEE*), Ioannis Lambadaris

***Abstract*—The recent networking trends driven primarily by the different virtualization technologies, such as Network Function Virtualization (NFV) and Service Function Chaining (SFC) pave the way for next-generation network services. In the 5G and beyond era, such services usually have strict delay requirements and the wider adoption of the distribution of their computational needs across the Edge-to-Cloud continuum is certainly a step in the right direction. However, the majority of the optimization solutions for placing the virtualized services so far focus on server selection, leaving other areas such as the impact of Non-Uniform Memory Access (NUMA) and CPU core selection underexplored. In this work, we herein formulate the problem of placing services as SFCs on an Edge/Cloud infrastructure, as a Mixed Integer Programming (MIP) problem. Then, we propose a heuristic algorithm called "Dynamic numa node Selection through Cores consolidation – DySCo" to solve it, which optimizes the placement in terms of server, NUMA and core selection. To the best of our knowledge, this is the first attempt to optimize network service placement in an Edge-Cloud interplay. Extensive simulation evaluation shows that DySCo is able to perform close to optimal while finding a solution in a real time fashion. Compared to a mix of baselines and modified solutions from the literature to treat this new problem, DySCo reduces on average the deployment cost by 17.53% and the delay by 28.88% for a given SFC.**

***Index Terms*—Network Function Virtualization, Service Function Chaining, Edge Computing, Cloud Computing, Resource Allocation, Non-Uniform Memory Access.**

I. Introduction

The new and upcoming 5G and Internet of Things (IoT) networks make the requirements in terms of Quality of Service (QoS) offered to users increasingly strict. High-speed services requiring data transfer rates up to 20 Gigabit per second and ultra-reliable services with low latency in the order of 1 millisecond, are becoming the norm [1]. To achieve such stringent QoS requirements while reducing the related costs, various technologies have been developed, which are mainly emphasizing on service virtualization. The pinnacle of this effort, Network Function Virtualization (NFV) allows for decoupling the network functions (e.g., Network Address Translation - NAT, Firewall) from dedicated hardware and having it as a software instance, called Virtualized Network Function (VNF), running on a conventional server on the Cloud [2]. Service Providers (SPs) design their own VNFs and by chaining them in a specific order they create Service Function Chains (SFCs) in order to provide their users with the requested network services. Leveraging the flexibility of virtualization, SFCs can be deployed in the proximity of end users, at the network Edge (Edge Computing), shortening propagation times and overall latency, thus resulting in improved QoS.

Although virtualization allows for distributing the SFC placement across the Edge-to-Cloud continuum and for reducing the deployment costs compared to a rigid middlebox deployment, reaping the benefits of this procedure is not straightforward; optimizing the placement of the VNF components among the available servers plays an important role in improving QoS [3]. For example, deploying an SFC exclusively at the Edge might overload the infrastructure, causing a performance drop and additional computational latency. On the other hand, solely utilizing a Cloud infrastructure results in greater propagation delay. Hence, the need to balance the load between the remote Cloud and the Edge is evident [4].

Apart from the location of the server, another important parameter to optimize during the VNF placement is the communication between the CPU processor and the memory for task processing. When the processors of a server access the memory simultaneously, the congestion on the CPU-memory bus adds latency, hence deteriorating the individual processing times. Due to the limitation of such uniform memory access, a Non-Uniform Memory Access (NUMA) architecture has emerged [5]. This technique divides both the memory (RAM) and the processor (CPU cores) into sub-blocks, then assigns each of these sub-blocks of CPU cores and RAM together forming what is referred to as a NUMA node. These different nodes are then interconnected enabling any CPU core to communicate with any memory block in the system. An example of a NUMA server with two nodes is illustrated on Fig. 1. This coupling allows for faster memory access when the CPU core is directly assigned to it, or a slower access time if the CPU communicates with the memory of another NUMA node [6]. Recent studies [7] have verified the impact of carefully selecting NUMA nodes during task placement in reducing delays and increasing system throughput. Additionally, selecting CPU cores during the VNF placement also has an often overlooked impact on the processing delay [8] [9].

As we see, optimizing the SFC placement to achieve high QoS can be a complex procedure. With this in mind, we propose a solution that reduces the end-to-end delay of SFCs deployed on an Edge-to-Cloud continuum while reducing the deployment costs. To the best of our knowledge, this is the first

T. B. Salah, M. Avgeris, and A. Leivadeas are with the Dept. of Software and Information Technology Engineering, École de Technologie Supérieure, University of Quebec, Montreal, Canada. I. Lambadaris is with the Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada. E-mails: taha.ben-salah.1@ens.etsmtl.ca, marios.avgeris@etsmtl.ca, aris.leivadeas@etsmtl.ca, ioannis@sce.carleton.ca

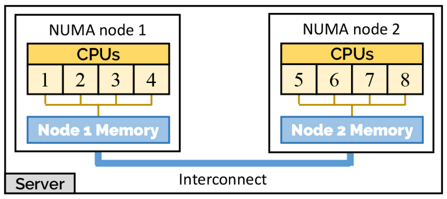


Fig. 1: A server with 2 NUMA nodes and 8 CPU cores.

work on SFC deployment, hybridly or exclusively at the Edge and/or Cloud, that not only optimizes server selection during VNF allocation, but also goes a step further into a fine-grained resource provisioning to the level of CPU cores within the selected NUMA node. This enables us to allocate VNFs not only to cores on the same server but, whenever possible, on the same NUMA node and on adjacent CPU cores sharing the same memory. Our approach aims to explore every possibility to minimize end-to-end delay, enhance resource utilization and throughput maximization, recognizing that micro-optimizing can contribute to overall performance improvements. The contribution of our work is threefold:

- We formulate the problem of placing the VNFs that comprise an SFC on an Edge-Cloud infrastructure, as a Mixed Integer Programming (MIP) problem. The novelty in this formulation is that a decision is made down to the CPU core level of the servers, while satisfying the location constraints (Edge/Cloud) of each VNF, as well as ensuring that no server is overloaded.
- We propose a heuristic algorithm named "Dynamic numa node Selection through Cores consolidation – DySCo" to solve this new version of the VNF placement problem. Our solution performs fine-grained resource allocation down to the CPU core level in order to reduce memory access times, that could lead to lower processing delay for each VNF. At the same time, we minimize the communication delays between the different VNFs composing the SFC, by reducing the number of hops between the selected servers. This also allows us to optimize the data flow on the utilized links, creating a double benefit: i) it decreases the consumed bandwidth which leads to an increase on the system throughput and ii) it favors the consolidation of the VMs that are used to implement the VNFs, leading to a reduction of the deployment costs.
- We evaluate the efficiency of the proposed algorithm through extensive simulation; DySCo is shown to perform close to optimal exhaustive solution of the MIP for small-scale scenarios, while operating in real time. At the same time, DysCo outperforms in terms of deployment cost and delay other approaches that do not consider the total set of constraints and objectives in such a new context.

The remainder of this paper is organized as follows: Section II highlights the related work, while the system model is detailed in Section III. The MIP formulation and the design of our proposed heuristic solution is introduced in Section IV. The obtained results of the performance evaluation are presented in Section V. Finally, in Section VI we present our conclusions and propose some future work directions.

II. Related Work

A. *VNF/SFC Placement at the Edge and Cloud*

Many works in the pertinent literature have dealt with the VNF and/or SFC placement problems by considering various optimization goals. In [10], the authors propose an approach to find deployment schemes for microservices at the Edge, that optimize the cost while meeting the demands for application average response time. Here, a microservice-based application is modeled as Service Chain while an *M/M/c* queue model is adopted to describe the operation of the microservices. The emerged NP-Complete deployment problem is first relaxed to a continuous one and then a heuristic approach is utilized to get integer solutions. On the other hand, Forti et al. [11] use a logic programming approach to solve the problem of placing VNF chains onto Edge-Cloud infrastructures. Their prototype, EdgeUsher, heuristically solves a probabilistic declarative description of the VNF chain placement problem to ensure high QoS guarantees, security, and service reliability. The authors in [12] attempt to simultaneously solve the VNF placement and the resource allocation problems at the Mobile Edge Computing (MEC) layer. To this end, they propose a genetic-based heuristic solution that minimizes the placement cost, as well as the computational resources and link utilization costs.

Another solution that leverages the power of the genetic algorithms is presented in [13]; there, the authors propose a framework that makes use of a delay and location aware genetic algorithm-based approach, in order to perform optimized sequential SFC placement and to dynamically allocate the required resources. Using a similar model, Liu et al. [14] propose two methods based on Lagrange relaxation and shortest paths in edge weighted graphs to jointly tackle the problems of VNF placement and SFC routing under multiple resource and QoS constraints. Nguyen et al. [15] also aim to minimize the resource usage cost by efficiently placing and chaining the VNFs of Cloud-based IoT SFCs that span across multiple Edge-Cloud infrastructures. To do so, two algorithms are proposed, a customized Markov approximation and a node ranking-based heuristic, to produce a near optimal solution.

A slightly different optimization objective is defined in [16], where the authors aim at minimizing the energy consumption in the infrastructure together with the resource utilization during SFC deployment; the proposed algorithm deals dynamically with the incoming SFCs and manages to decrease the number of required open servers, while at the same time reduce their idle energy consumption. Another work on energy consumption minimization during SFC embedding is presented in [17], where the authors propose a heuristic resource and energy-aware SFC strategy in an Edge–Cloud environment. Their algorithm comprises of three main procedures: the Edge/Cloud offloading decision, the VNF mapping and the virtual link mapping. Similarly, Xu et al. [18] devise a heuristic approach, based on integer linear programming (ILP) to linear programming relaxation, for the VNF placement problem, while Yue et al. [19] come up with a heuristic that utilizes shareable VNF instances to find a solution for their ILP problem. A Tabu Search meta-heuristic is proposed in [20] to solve the mixed integer programming problem of minimizing a multi-variate objective function. The target here is to optimize the end-to-end communication delay while keeping the cost minimum during the placement and deployment of service chained VNFs in an Edge-Cloud infrastructure.

Although the problem of VNF/SFC placement seems fairly well-studied in the literature with various optimization goals, none from the above works pay attention to the in-depth CPU allocation and its impact on the overall efficiency of the placement solution. However, studies performed on the characterization of individual VNF performance, showed that the NUMA architecture can cause performance degradation, when a poor CPU allocation decision is taken [21]. This has prompted us to investigate further the impact of NUMA on the system performance in general and on the efficiency of the SFC placement in particular.

B. Impact of NUMA allocation

In a typical NUMA system, a given CPU accesses the local RAM faster than a remote one on another NUMA node, hence the *non-uniform access* attribute. This is mainly due to the signal path length from a CPU core to a given memory block and it plays a significant role in the system performance; a large path leads to higher RAM access time and the more it is shared between different CPUs, the more prone it is to throughput and delay bottlenecks. This leads to noticeable impacts on the performance of large-scale systems [22].

Kim et al. [23] measured the average data access latency of VMs in a four-node NUMA machine and investigated the induced delay. Their findings show that latency tends to increase with the use of a memory block that is not directly attached to the core, with the access of the memory of a remote NUMA node and the use of the interconnection paths between NUMA nodes. Following that lead, authors in [24] investigated the impact of thread mapping on the performance and energy consumption for parallel applications on modern NUMA systems. Particularly, their efforts focused on optimizing the memory access in order to take advantage of the local access, while at the same time not overload it and avoid performance degradation due to local memory congestion. Results showed that a proper mapping can reduce the execution time by up to 23.8%. A reduction of 14.6% in energy consumption is also achieved.

Additionally, in [6], the authors investigated the interaction between NUMA and remote direct memory access through the network interface card, the performance of which is affected by the NUMA locality of the targeted node and its workload. The results showed that overloading the target node can lead to bandwidth degradation of up to 20% for simple atomic memory read operations, and up to 50% when the entire memory is accessed, meaning that twice the speedup can be achieved by directing operations to unburdened NUMA nodes.

These studies highlight the significant impact of the NUMA node selection within a server and how it can speed up the memory access time if chosen properly. However, the evaluation was limited to cases of data flow processing of some very specific applications. The general case of more complex functions like VNFs was not investigated, nor the case of chaining several functions like SFCs. In this direction, a recent study [7] has proposed a solution of VM consolidation that focuses on reducing the number of activated servers and the number of VM migrations to achieve that in the context of NUMA system. Their approach, using a Modified Grey Wolf Algorithm, optimally selects NUMA nodes within servers with the objective to reduce energy consumption by reducing the number of activated servers and VM migrations. However, no interdependency among the VMs is assumed (like in an SFC-enabled environment). This motivates our exploration to understand the impact of CPU core allocation on reducing processing delays during SFC placement.

C. Impact of CPU core allocation

In the VNF/SFC placement context, Papathanail et al. in [8] highlight the -often overlooked- performance importance of fine-grained CPU selection as complementary to server selection. To this end, different scenarios of SFC spanning across multi-core systems are investigated, with diverse resource profiles, and the following points are made: i) for higher data rate, allocating cores from the same NUMA node leads to higher throughput, ii) allocating heterogeneous workload (e.g., CPU- vs. memory-intensive tasks) to the same core leads to higher throughput than allocating a homogeneous one and iii) more than two VNFs sharing the same core leads to throughput degradation. Similarly, authors in [9] pinpoint the importance of fine-tuned placement to the level of exact CPU core allocations. They propose an algorithm that can predict performance degradation and optimize core allocation, achieving an overall higher throughput for incoming SFCs, and compared to random core allocation, an improvement in system throughput by 39.2%.

Still, for both these works the optimization criteria is system throughput and end-to-end delay minimization is not taken into account. Additionally, these particular works mainly focus only on an intra-server scenario, without considering multiple SFCs to be placed in large Edge-Cloud infrastructures, and without considering any location constraints and hardware heterogeneity that may arise in such Edge/Cloud interplay. Table I presents a comprehensive comparison between the literature and the proposed solution.

III. System Model

A. Server architecture

Memory access can be a bottleneck for the processor performance. That is why, in modern computer architectures, a hierarchical cache system is implemented, which bridges the gap between CPU and the constant lookups in the remote RAM. According to [25], the most popular hierarchy system that has been adopted by modern microprocessor designs is the three-level cache system; this dictates the use of smaller and faster level one (L1) and two (L2) caches which are kept private and in closer proximity to CPU cores, and a larger level three (L3) cache which is shared between all cores. L2 cache, however, can be also shared between two consecutive

TABLE I: Comprehensive comparison between the literature and the proposed solution.

Article	Optimization Objective					Provisioning		Use of NUMA
	Cost	Resource utilization	Delay	Bandwidth	Energy	Implicit	Explicit	
Leivadeas et *al.* [20]	✓	✓	✓			✓		
Nguyen et *al.* [15]	✓	✓				✓		
Xu et *al.* [18]	✓					✓		
Deng et *al.* [10]	✓					✓		
Kiran et *al.* [12]	✓	✓				✓		
Sun et *al.* [16]		✓			✓	✓		
Thanh et *al.* [17]		✓			✓	✓		
Yue et *al.* [19]		✓				✓		
Magoula et *al.* [13]			✓			✓		
Liu et *al.* [14]	✓					✓		
Forti et *al.* [11]			✓	✓		✓		
Diamanti et *al.* [4]		✓		✓	✓		✓	
Kim & Park [23]			✓				✓	Node
Hu et *al.* [7]					✓		✓	Node
Papathanail et *al.* [8]				✓			✓	CPU core
Yu et *al.* [9]				✓			✓	CPU core
DySCo	✓	✓	✓	✓			✓	CPU core

CPU cores, as shown in Fig. 2 for an example of 8 CPU cores spread in two NUMA nodes.

Kim in [23] detailed how such a hierarchy works in a NUMA server: when a CPU core requests to access data stored in the memory, first the L1 cache is checked and then L2. If the data is not there, then the shared L3 cache and the RAM of the same NUMA node are checked and subsequently those of the remote NUMA nodes. To model this operation in a three-level cache server, we introduce a penalty concept that represents the cost of fetching data from the memory for a CPU core. Based on the works of [23] and [25], we define the following cases of penalty values, also illustrated in Fig. 3:

- 1st case: The requested data are present on private cache L1 or L2; this introduces no penalty, given the direct and fast access to the data. Thus, a penalty value equal to zero is assigned to the communication between two adjacent CPU cores.
- 2nd case: The requested data are present on the L3 cache or the RAM of the local NUMA node; this adds a data access penalty $p > 0$. This penalty is inflicted due to contention on the shared cache and memory controller and is assigned to the communication between two non-adjacent CPU cores belonging to the same NUMA node.
- 3rd case: The requested data are present in a remote node; this adds a penalty $Q > p$, for crossing the interconnection between NUMA nodes and is assigned to the communication between two CPU cores belonging to different NUMA nodes.

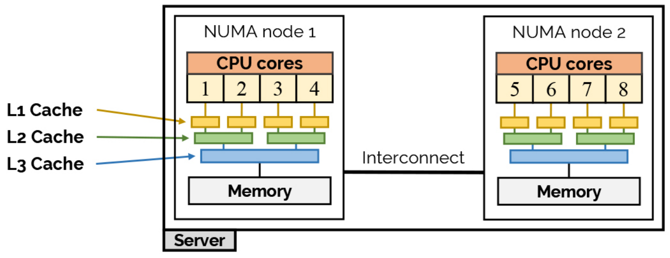


Fig. 2: Cache sharing for 8 cores spread in 2 NUMA nodes.

B. Infrastructure

We model the Edge-Cloud infrastructure as an undirected graph $G = (N, V)$, where N represents the set of servers and routers (set of nodes) and V the set of vertices/links that interconnect them. In more detail, the nodes are categorized into two types: i) the servers $S \subset N$ that host the VNFs and ii) the routers $F \subset N$ that forward the traffic. It stands that $S \cup F = N$. We also identify two server subsets, depending on their location, i.e., the Edge $S_E \subseteq S$ and the Cloud $S_C \subseteq S$, with $S_E \cup S_C = S$. Each server $s \in S$ is attributed with a number of available NUMA nodes B_s and a vector of available resources R_s, in our case consisting of CPU cores H_s and RAM Memory M_s, $R_s = (H_s, M_s)$. Furthermore, each link $(u, v) \in V$ is characterized by a bandwidth capacity $\rho(u, v)$ and a propagation delay $d_{pr}(u, v)$. To accommodate the penalty design introduced in the previous subsection, we define a penalty matrix P_s where $P_s(j, j')$ represents the penalty for the communication between cores j and $j' \in 1, ..., H_s$, $j \neq j'$, of server $s \in S$. If we consider the server in Fig. 2 as an example, its corresponding penalty matrix would be of size 8×8 as illustrated in Table II.

An SFC consisting of K interconnected (linearly or bi-furcated) VNFs, is represented as an ordered set $A = (a_1, a_2, ..., a_K)$, where $a_k, k = 1, ..., K$, denotes the k-th VNF in the chain. Each VNF is attributed with a vector of demands D_k, which consists of the requested CPU cores h_k and RAM m_k. Also, each VNF has a location constraint L_k indicating an Edge-only, Cloud-only, or location-indifferent allocation. Finally, each SFC A has a bandwidth requirement, ϕ_A. Table

TABLE II: P_s for 8-core server with 2 NUMA nodes.

CPU core	1	2	3	4	5	6	7	8
1	0	0	p	p	Q	Q	Q	Q
2	0	0	p	p	Q	Q	Q	Q
3	p	p	0	0	Q	Q	Q	Q
4	p	p	0	0	Q	Q	Q	Q
5	Q	Q	Q	Q	0	0	p	p
6	Q	Q	Q	Q	0	0	p	p
7	Q	Q	Q	Q	p	p	0	0
8	Q	Q	Q	Q	p	p	0	0

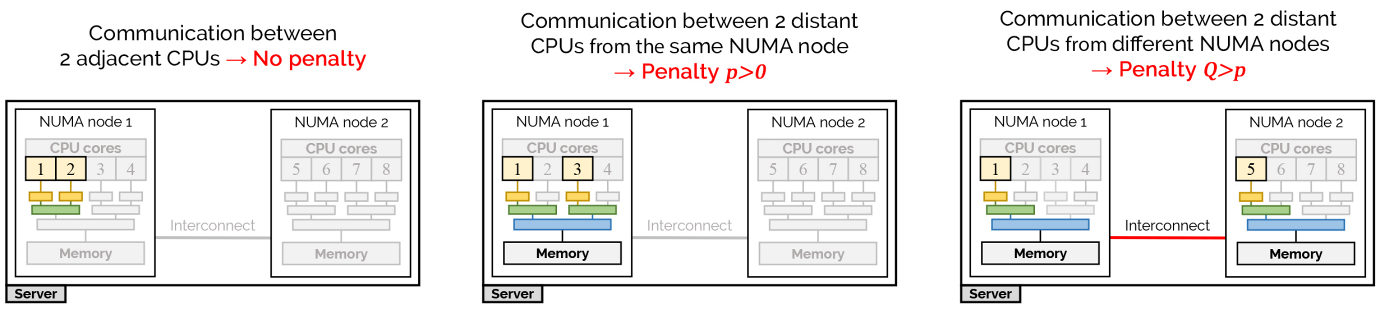

Communication between 2 adjacent CPUs → No penalty
Communication between 2 distant CPUs from the same NUMA node → Penalty p>0
Communication between 2 distant CPUs from different NUMA nodes → Penalty Q>p
NUMA node 1
NUMA node 2
CPU cores
Interconnect
Memory
Server

Fig. 3: Penalty for communication between CPU cores.

TABLE III: Summary of the Key Notation.

Symbol	Interpretation
$G = (N, V)$	Infrastructure Graph (nodes, links)
S, F	Set of servers and routers
S_E, S_C	Set of Edge and Cloud servers
s_n^E, s_n^C	The n-th server of S_E or S_C
$b \in B_s$	NUMA nodes of server $s \in S$
R_s	Available resources $[cores, GB]$ of server $s \in S$
H_s	No. of available physical cores in server $s \in S$
H_b	No. of available physical cores in NUMA node $b \in B_s$
(u, v)	A physical link $\in V$
$\rho(u, v)$	Available bandwidth (Mb/s) of link $(u, v) \in V$
$P_s(j, j')$	Penalty for communication between physical cores j, j'
$a_k \in A$	The k^{th} VNF of SFC A
A_E, A_C	Set of VNFs with Edge and Cloud location constraints
D_k	Requested resources $[cores, GB]$ of VNF $a_k \in A$
h_k	No. of requested CPU cores of VNF $a_k \in A$
L_k	Location requirement of VNF $a_k \in A$
ϕ_A	Bandwidth (Mb/s) requirement of SFC A
θ_b	Priority factor of NUMA node $b \in B_s$
Θ_s	Priority factor of server $s \in S$

III summarizes the key notation used in our modeling.

IV. Algorithm Design

A. Mixed Integer Programming Formulation

In order to solve optimally the VNF Placement problem, while taking into consideration both the NUMA nodes and the individual cores, we need to address the following two design challenges: i) restrict the allocation of the virtual cores of a specific VNF in the physical CPU cores of the same server and ii) ensure that this server is the one where said VNF has been placed in. Thus, we introduce the following variables:

- $x_{s,j}^{a_k,i}$: binary variable equal to 1, when the i^{th} core of a VNF $a_k \in A$ is allocated to the j^{th} physical core on server $s \in S$.
- $y_s^{a_k}$: binary variable equal to 1 when the VNF $a_k \in A$ is allocated on server $s \in S$.
- $z_{uv}^{a_k a_{k'}}$: integer variable equal to the traffic (bps) that flows between two interconnected VNFs $a_k, a_{k'} \in A$, if that virtual link is routed over the physical link $(u, v) \in V$.

With our placement solution we aim firstly, to optimally leverage the NUMA's hierarchical structure, that as shown in Section II can considerably reduce the processing delay overheads, and secondly reduce the total bandwidth consumed in order to reduce the deployment cost [2]. Thus, we formulate the problem of SFC allocation in an Edge-Cloud infrastructure with the following objective function:

$$\min_{x,z} \sum_{\substack{a_k \in A,\\ h_k \geq 2}} \sum_{i=2}^{h_k} \sum_{i'=1}^{i-1} \sum_{s \in S} \sum_{j=2}^{H_s} \sum_{j'=1}^{j-1} \left\lfloor \frac{x_{s,j}^{a_k,i} + x_{s,j'}^{a_k,i'}}{2} \right\rfloor \cdot P_s(j, j') + \sum_{a_k \in A} \sum_{a_{k'} \in A} \sum_{(u,v) \in V} z_{uv}^{a_k a_{k'}}. \quad (1)$$

Here, the first term represents the penalty of how far the allocated CPU cores to a VNF are, as explained in Table II. In this first term, we only consider requests with more than two CPU cores that will create a communication between their respective memory blocks. Note that the floor operator in $\lfloor \frac{x_{s,j}^{a_k,i} + x_{s,j'}^{a_k,i'}}{2} \rfloor$ gives a value equal to 1 only when $x_{s,j}^{a_k,i} = x_{s,j'}^{a_k,i'} = 1$, otherwise it gives 0. In other words, the penalty $P_s(j, j')$ is counted in the sum only when server cores j and j' are both occupied respectively by the virtual cores i and i' of the VNF a_k, where $i, i' \leq h_k$. By minimizing the penalties, the allocation of adjacent cores is rewarded, which leads to better NUMA node allocation and consequently will lead to lower processing delays and CPU to memory communication overheads. The second term tries to minimize the consumed bandwidth in the physical infrastructure, i.e., the number of hops in a path that interconnects two adjacent VNFs, favoring this way the VM consolidation. This ultimately also minimizes resource utilization and reduces deployment costs. In this system, we also identify the following constraints:

$$\sum_{i=1}^{h_k} \sum_{j=1}^{H_s} x_{s,j}^{a_k,i} = h_k \cdot y_s^{a_k}, \forall a_k \in A, \forall s \in S, \quad (2)$$

$$\sum_{a_k \in A} \sum_{i=1}^{h_k} x_{s,j}^{a_k,i} \leq 1, \forall s \in S, \forall j \in \{1, ..., H_s\}, \quad (3)$$

$$\sum_{j=1}^{H_s} x_{s,j}^{a_k,i} \leq 1, \forall a_k \in A, \forall s \in S, \forall i \in \{1, ..., h_k\}, \quad (4)$$

$$\sum_{s \in S} y_s^{a_k} = 1, \forall a_k \in A, \quad (5)$$

$$\sum_{f \in F} y_f^{a_k} = 0, \forall a_k \in A, \quad (6)$$

$$\sum_{a_k \in A} y_s^{a_k} = 0, \forall s \in S, s \notin L_k. \quad (7)$$

Constraint (2) makes sure that the number of requested cores is equal to the number of utilized cores in the server. In other words, that no requested core has been left out during the allocation. Constraint (3) ensures that each physical core in a server will host at most a single virtual core from a VNF. On the other hand, constraint (4) guarantees that each VNF core is allocated to at most one server core. At the same time, constraint (5) ensures that each VNF will be allocated in only one server and constraint (6) that no VNF will be allocated in a router. Constraint (7) satisfies the location requirements.

Additionally, we need to ensure that the servers' resources (CPU and memory) will not be oversubscribed. Thus, we introduce the following constraint:

$$\sum_{a_k \in A} D_k \cdot y_s^{a_k} \leq R_s, \forall s \in S. \tag{8}$$

Finally, regarding the interconnection of the VNFs in an SFC, we have the following constraints:

$$\sum_{a_k \in A} \sum_{\substack{a_{k'} \in A, \\ k' > k}} (z_{uv}^{a_k a_{k'}} + z_{vu}^{a_k a_{k'}}) \leq \rho(u,v), \forall (u,v) \in V, \tag{9}$$

$$\sum_{v \in N} (z_{uv}^{a_k a_{k'}} - z_{vu}^{a_k a_{k'}}) = \phi_A \cdot (y_u^{a_k} - y_u^{a_{k'}}), \qquad \forall a_k, a_{k'} \in A, \forall u \in N. \tag{10}$$

Constraint (9) ensures that capacities of the links are respected, whereas constraint (10) represents the flow conservation constraint between the source and the sinks of the virtual links.

Reformulating the optimization problem: To eliminate the non-linear floor operator in Eq. (1), we introduce the binary variable $c_{s,j,j'}^{a_k,i,i'}$, which is subject to the following constraints:

$$0 \leq c_{s,j,j'}^{a_k,i,i'} \leq 1, \tag{11}$$

$$c_{s,j,j'}^{a_k,i,i'} \leq x_{s,j}^{a_k,i}, \tag{12}$$

$$c_{s,j,j'}^{a_k,i,i'} \leq x_{s,j'}^{a_k,i'}, \tag{13}$$

$$x_{s,j}^{a_k,i} + x_{s,j'}^{a_k,i'} - 1 \leq c_{s,j,j'}^{a_k,i,i'}. \tag{14}$$

Effectively, $c_{s,j,j'}^{a_k,i,i'}$ is equal to $\lfloor \frac{x_{s,j}^{a_k,i} + x_{s,j'}^{a_k,i'}}{2} \rfloor$ and thus, $c_{s,j,j'}^{a_k,i,i'} = 1$ when $x_{s,j}^{a_k,i} = x_{s,j'}^{a_k,i'} = 1$. Therefore $c_{s,j,j'}^{a_k,i,i'}$ represents the simultaneous double occupation of CPU cores j and j' of server s by the virtual cores i and i' of VNF a_k. This allows us to reformulate the objective function (1) as follows:

$$\min_{c,z} \sum_{\substack{a_k \in A, \\ h_k \geq 2}} \sum_{i=2}^{h_k} \sum_{i'=1}^{i-1} \sum_{s \in S} \sum_{j=2}^{H_s} \sum_{j'=1}^{j-1} c_{s,j,j'}^{a_k,i,i'} \cdot P_s(j,j') + \sum_{a_k \in A} \sum_{a_{k'} \in A} \sum_{(u,v) \in V} z_{uv}^{a_k a_{k'}}. \tag{15}$$

Subsequently, the optimization problem at hand is transformed to the following Mixed Integer Linear Program (MILP):

$$\begin{cases} min & (15), \\ s.t. & \text{constraints (2) to (14).} \end{cases}$$

B. Heuristic

In general, the VNF/SFC placement problem is known in the literature to be an NP-hard problem [10] [12] [15]. This means that the execution time of a MIP solver increases exponentially with the size of the infrastructure, hence the importance of heuristic algorithms which can provide near-optimal solutions in much faster time, without being impacted by the infrastructure size, is evident. Thus, we present the *"Dynamic numa node Selection through Cores consolidation - DySCo"* algorithm, our SFC placement heuristic solution. DySCo allocates the incoming SFCs sequentially, focusing on individual service, real time optimization, enabling in this way quick adaptation to dynamic network conditions and flexibility in cases where SFCs are requested and released frequently. Its operation is broken down to three steps: 1) Server classification, 2) VNF classification and 3) SFC deployment.

1) Server Classification: We introduce the term *block* to denote a pair of consecutive available CPU cores, in a server $s \in S$, that share the same L2 cache, and *anti-block* to denote a pair of consecutive CPUs, that share the same L2 cache, with exactly one available core. We define then a priority factor θ_b for each NUMA node of that server (or NUMA "socket", assuming that the number of NUMA nodes is equal to the number of sockets) $b \in B_s$, calculated as the sum of the number of available CPU cores and the number of blocks of CPU in that NUMA node:

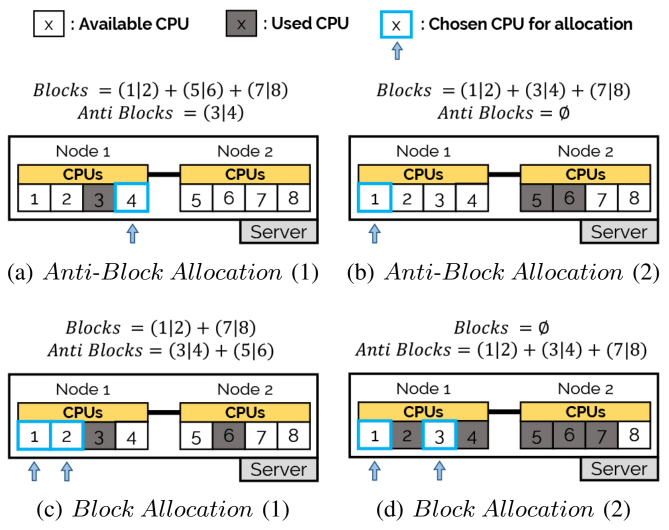
$$\theta_b = availableCores(b) + blocks(b). \tag{16}$$

Additionally, we introduce the priority factor for a server Θ_s, $s \in S$, which is given by the sum of the priority factors of its NUMA nodes:

$$\Theta_s = \sum_{b \in B_s} \theta_b. \tag{17}$$

During this step, the sets containing the Edge and the Cloud servers, S_E and S_C respectively, are sorted in a descending order according to their servers' Θ_s, which essentially prioritizes one server over another when both have the same number of available CPUs, but one has more blocks than the other. Having more blocks is equivalent to having more consecutive available cores and, therefore, favoring communication with lower penalty and thus processing delays.

2) VNF Classification: In this step, we classify each VNF of an incoming SFC request, $a_k \in A$, based on their location constraints, L_k; VNFs with Cloud or Edge requirements are placed accordingly, while VNFs with no location constraints are placed at the Edge, only if they are directly connected to VNFs with Edge requirements, otherwise, they are placed at the Cloud. By doing so, we minimize the number of hops between the different VNFs of an SFC spanning across the Edge and Cloud. This reduces the delay and bandwidth consumption, leading eventually to a reduction in the cost of deployment, as we also reduce the number of links utilized and the use of the Edge infrastructure in general. This can be beneficial, since resources at the Edge may be more scarce and more highly requested leading also to higher operational expenses [26]. This step outputs two sets containing the VNFs that are to be placed at the Edge or the Cloud, $A_E \subseteq A$ and


x : Available CPU
x : Used CPU
x : Chosen CPU for allocation
Blocks = (1|2) + (5|6) + (7|8)
Anti Blocks = (3|4)
Blocks = (1|2) + (3|4) + (7|8)
Anti Blocks = ∅
Blocks = (1|2) + (7|8)
Anti Blocks = (3|4) + (5|6)
Blocks = ∅
Anti Blocks = (1|2) + (3|4) + (7|8)
Node 1
Node 2
CPUs
1 2 3 4
5 6 7 8
Server

(a) *Anti-Block Allocation* (1) (b) *Anti-Block Allocation* (2)

(c) *Block Allocation* (1) (d) *Block Allocation* (2)

Fig. 4: *Block Allocation* and *Anti-Block Allocation* Scenarios.

$A_C \subseteq A$ respectively, where $A_E \cup A_C = A$. Finally, we sort the lists in a descending order, based on the number of requested CPU cores of each VNF, h_k.

3) SFC Deployment: Before we dive into solving the SFC placement problem, we demonstrate the basic case of a single VNF deployment, $a_k \in A$. To accommodate this procedure, we introduce two elementary functions, namely $Block\ Allocation(i, i')$ and $Anti\text{-}Block\ Allocation(i)$. The former allocates two VNF cores, i and $i' \in 1, ..., h_k$, to the first available block in the NUMA node; if there is no block available, then it allocates them to the first available two cores. On the other hand, the latter allocates a single VNF core i to the first available anti-block, or to the first available core if no anti-block is available. Fig. 4 illustrates an example of how these functions allocate VNF cores on different server configurations. We then combine these functions to a single algorithm called $CoreAllocation(h_k, b)$ (Algorithm 1). This procedure is responsible for allocating the requested h_k cores of VNF a_k to NUMA node b. In a nutshell, Algorithm 1 tries to iteratively allocate pairs of VNF cores in adjacent blocks in a server, as much as possible, in order to reduce the induced penalty. Any remaining single core is allocated in an anti-block (if available), favoring in this way CPU consolidation instead of “breaking” an empty block that can be used for future allocations. Hence, the $CoreAllocation$ algorithm can be applied and generalized for any number of requested CPU cores from a VNF, either even or odd.

Having established the placement procedure for a single VNF, we move forward to explaining the entire SFC A placement solution. For this procedure, DySCo operates in two sequential steps: i) placing VNFs to the Edge and then ii) placing VNFs to the Cloud.

i) Edge Allocation: Let s_1^E be the first server on the S_E set. It represents the Edge server with the highest priority factor, i.e., the best candidate for the VNF placement. Depending on its number of available CPU cores $H_{s_1^E}$, DySCo can either fit all the VNFs of the A_E in it (*Single Server Allocation - SSA*), or more servers are needed (*Multi Server Allocation - MSA*).

Algorithm 1 $CoreAllocation$

```
Input: $h_k, b$
 1: $i \leftarrow 1$
 2: while $i \leq h_k$ do
 3:     if ($i < h_k$) then
 4:         $i' \leftarrow i + 1$
 5:         $Block\ Allocation(i, i')$
 6:         $i \leftarrow i + 2$
 7:     else
 8:         $Anti\text{-}Block\ Allocation(i)$
 9:         $i \leftarrow i + 1$
10:     end if
11: end while
```

Algorithm 2 SSA

```
Input: $A_E$, $s_1^E$
 1: for all ($a_k \in A_E$) do
 2:     get ($b_m, H_{b_m}$) and ($b_M, H_{b_M}$) from $s_1^E$
 3:     if ($h_k > H_{b_M}$) then        /* Multi-NUMA node */
 4:         while ($h_k > H_{b_m}$) do
 5:             get ($b_M, H_{b_M}$)
 6:             $CoreAllocation(H_{b_M}, b_M)$
 7:             $h_k \leftarrow h_k - H_{b_M}$
 8:         end while
 9:         if ($h_k \neq 0$) then $CoreAllocation(h_k, b_m)$
10:         end if
11:     end if
12:     if ($h_k \leq H_{b_m}$) then        /* Single-NUMA node */
13:         if ($h_k == 2$ AND $H_{b_m} == 2$) then
14:             if ($b_m$ has block) then
15:                 $CoreAllocation(h_k, b_m)$
16:             else $CoreAllocation(h_k, b_M)$
17:             end if
18:         else $CoreAllocation(h_k, b_m)$
19:         end if
20:     end if
21:     if ($H_{b_m} \leq h_k \leq H_{b_M}$) then $CoreAllocation(h_k, b_M)$
22:     end if
23: end for
```

In the case when SSA is feasible (Algorithm 2), we first sort the NUMA nodes $b \in B_{s_1^E}$ by their priority factor θ_b. Then, we select two specific NUMA nodes as follows: b_m, which is the NUMA node with the lowest θ_b and b_M, which is the NUMA node with the highest θ_b respectively. Additionally, we denote by H_{b_m} and H_{b_M} the number of their available cores. Depending on the number of requested CPU cores h_k of VNF $a_k \in A_E$ and the availability on b_m and b_M, DySCo allocates it into s_1^E. The case in line 13 of the SSA pseudo-code, helps in further optimizing the resource allocation by finding a trade-off between allocating at an already-utilized NUMA node for resource consolidation and between allocating at a non-utilized NUMA node to reduce penalties. An example is illustrated in Fig. 5. For a partially utilized NUMA node, the SSA algorithm will prefer to use it if there is a block of CPU cores that share

TABLE IV: Configurations of the different Infrastructure, Servers and VNFs/SFCs.

Type	Infrastructure size			Server's characteristics			
	Small	Medium	Large	RAM	Storage	NUMA nodes	CPUs per node
Cloud	3	6	9	256 GB	2 TB	2	12 or 24
Edge	9	18	27	64 GB		1 or 2	8

VNF's characteristics		SFC's characteristics	
Requested CPU cores	RAM	Bandwidth (Mb/s)	Number of VNFs
2, 3 or 4	4 or 8 GB	10, 20, 50 60, 70 or 80	2 - 5

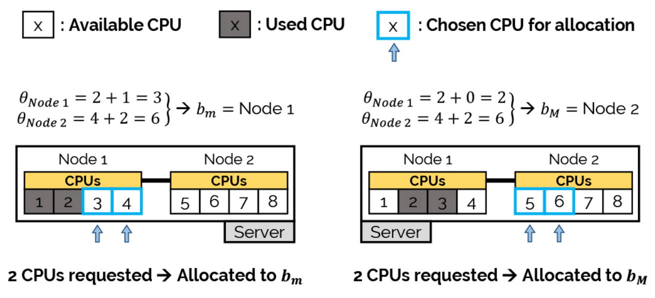

x : Available CPU
x : Used CPU
x : Chosen CPU for allocation
$\theta_{Node\,1} = 2+1 = 3$
$\theta_{Node\,2} = 4+2 = 6$
$\rightarrow b_m$ = Node 1
$\theta_{Node\,1} = 2+0 = 2$
$\theta_{Node\,2} = 4+2 = 6$
$\rightarrow b_M$ = Node 2
Node 1
Node 2
CPUs
1 2 3 4
5 6 7 8
Server
2 CPUs requested → Allocated to b_m
2 CPUs requested → Allocated to b_M

Fig. 5: Enhancing consolidation during the SSA algorithm.

Algorithm 3 MSA

```
Input: $A_E$, $S_E$
1: calculate $(s_2^E, ..., s_{n+1}^E)$
2: for all $(a_k \in A_E)$ do
3:     get $(s_m, H_{s_m})$ and $(s_M, H_{s_M})$ from $(s_2^E, ..., s_{n+1}^E)$
4:     if $(h_k \leq H_{s_m})$ then $SSA(a_k, s_m)$
5:     end if
6:     if $(H_{s_m} < h_k \leq H_{s_M})$ then $SSA(a_k, s_M)$
7:     end if
8: end for
```

the same L2 cache. This will enhance the consolidation and reduce the penalties. Yet, if the two cores requested belong to two anti-blocks, then the L3 cache will be used, which will incur more penalty. Thus, in the latter case it will be more cost-efficient to use the second NUMA node, which is idle but has more consecutive blocks. After placing a_k, the variables b_m, H_{b_m}, b_M and H_{b_M} are recalculated before proceeding to a_{k+1}. This adds a dynamic aspect to the NUMA node selection and optimization at each iteration.

In the case of MSA (Algorithm 3), since multiple servers are needed to host the VNFs of A_E, we sort the Edge servers in a descending order based on their distance in number of hops from s_1^E. Then, we select the closest n servers with enough capacity to accommodate the VNFs of A_E, creating the set $(s_2^E, ..., s_{n+1}^E)$. We denote as s_m and s_M, the servers of this set with the lowest and the highest priority factors Θ_s respectively and as H_{s_m}, H_{s_M} the number of their available cores. Similarly to the SSA procedure, each VNF $a_k \in A_E$ is placed sequentially to a server, by comparing h_k, H_{s_m}, and H_{s_M}. The actual placement is performed on the selected server using Algorithm 2. After placing a_k, the variables s_m, H_{s_m}, s_M and H_{s_M} are recalculated before proceeding to a_{k+1}. This procedure is iterated until all the VNFs of A_E are placed.

ii) Cloud Allocation: The procedure of placing VNFs at the Cloud is the same as that at the Edge; we simply replace A_E with A_C and S_E with S_C and follow the same steps.

The operation flow of DySCo is given in Algorithm 4, while Fig. 6 illustrates the relationships and control mechanisms among the building algorithmic components. In the latter, the ability to execute parts of the mechanism in parallel is evident.

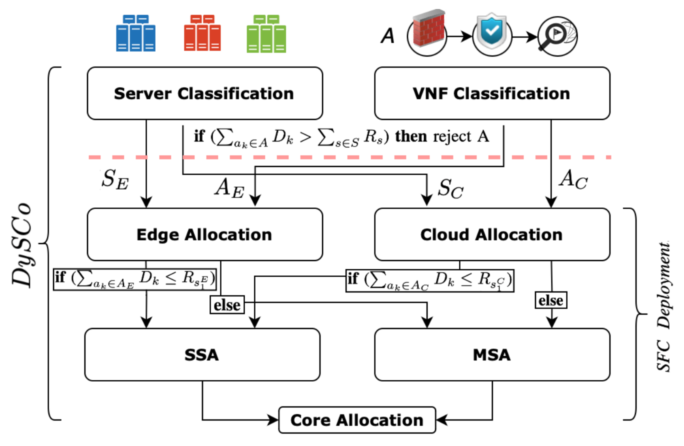

A
Server Classification
VNF Classification
if $(\sum_{a_k \in A} D_k > \sum_{s \in S} R_s)$ then reject A
S_E
A_E
S_C
A_C
Edge Allocation
Cloud Allocation
if $(\sum_{a_k \in A_E} D_k \leq R_{s_1^E})$
else
if $(\sum_{a_k \in A_C} D_k \leq R_{s_1^C})$
else
SSA
MSA
Core Allocation
DySCo
SFC Deployment

Fig. 6: DySCo Information and Control Flow.

Algorithm 4 $DySCo$

```
Input: $G$, $SFCrequests$
1: for all $A \in SFCrequests$ do
2:     $(S_E, S_C) \leftarrow$ 1) Server Classification
3:     $(A_E, A_C) \leftarrow$ 2) VNF Classification
4:     if $(\sum_{a_k \in A} D_k > \sum_{s \in S} R_s)$ then reject A
5:     else        /* 3) SFC Deployment */
6:         /* i) Edge Allocation */
7:         if $(A_E \neq \emptyset)$ then
8:             if $(\sum_{a_k \in A_E} D_k \leq R_{s_1^E})$ then $SSA(A_E, s_1^E)$
9:             else $MSA(A_E, S_E)$
10:            end if
11:        end if
12:        /* ii) Cloud Allocation */
13:        if $(A_C \neq \emptyset)$ then
14:            if $(\sum_{a_k \in A_C} D_k \leq R_{s_1^C})$ then $SSA(A_C, s_1^C)$
15:            else $MSA(A_C, S_C)$
16:            end if
17:        end if
18:    end if
19: end for
```

V. Performance Evaluation

A. Experimental set up and simulation scenarios

In this section, we assess the performance of DySCo. For the evaluation we conducted simulations on three different sizes of infrastructure: "Small" (3 Cloud Nodes & 9 Edge Nodes), "Medium" (6 Cloud Nodes & 18 Edge Nodes), "Large" (9

Cloud Nodes & 27 Edge Nodes). The specifications of the different servers for each infrastructure are summarized on the first part of Table IV.

To better stress the system under realistic real world scenarios, we performed two families of experiments: the first aims at studying the dynamics of resource allocation with SFCs arriving and departing the system according to the clients' demands (*Dynamic Setting*). The requests' arrival rate follows a Poisson distribution with a rate of $4SFCs/hour$, and their departure is set by an exponential distribution with an average of $24h$. In the second family of experiments, the incoming SFCs remain allocated in the infrastructure throughout the entire execution of the algorithm (*Static Setting*). This allowed us to evaluate DySCo performance in a more static environment that offers fewer and fewer available resources to each new SFC request. Throughout the experimentation, the SFC requests arrive sequentially and are treated one by one. The specifications of the SFCs and their VNFs is provided in the second part of Table IV. It should be noted, that these specifications have been based on real data presented in [27].

B. Comparison algorithms

To evaluate the performance of DySCo, we compare it to the optimal solution of the MIP, as well as with two other SFC placement algorithms: the *First Fit (1stFit)* and the *Hybrid Heuristic Grey Wolf Algorithm (HHGWA)* [7]. 1stFit is a baseline algorithm that greedily chooses the first available server, and then picks the first available CPU core of the first available NUMA node within the server to host the VNFs' cores. On the other hand, the HHGWA is a VM consolidation solution proposed by Hu et al. [7] to optimize the NUMA node selection for each VM allocation, in order to reduce the total number of utilized servers. This algorithm is selected as it follows a novel approach for optimizing NUMA nodes selection within each candidate server prior to hosting a VM. Although HHGWA does not consider SFC-like interdependencies among VMs, to our knowledge it is the closest work in the literature that optimizes placement down to NUMA node selection in a large-scale infrastructure and not in an intra-server setting.

In more detail, HHGWA is based on the Grey Wolf Algorithm [28] to select the best candidate servers to host the VMs. First, it sorts the VMs by their *average request ratio*, i.e., the ratio between the average amount of requested resources of each type and the available capacity of this type on the server. The VM with the highest average request ratio is prioritized during the placement. Similarly, the selection of NUMA nodes within the candidate servers is made by sorting them using another ratio, that of *remaining resources utilization*, in order to balance the load between the different NUMA nodes within a given server. Then, the HHGWA proceeds to match and place the sorted VMs with the sorted NUMA nodes, and any VM that failed to be allocated at this stage will be added to a queue for a subsequent reallocation step. To make the comparison between HHGWA and DySCo fairer, the following modifications were made:

- We introduced location constraints to the VM placement in HHGWA, as the authors in [7] did not consider an Edge-Cloud infrastructure for their solution.
- Following the allocation of the VNFs of a SFC, a shortest path algorithm is used to interconnect them, since the notion of SFC was not considered in the initial version of the algorithm.

Table V summarizes the different algorithms used for the comparison. We should also note here that all the algorithms were implemented in a Java-based simulator [2], while the CPLEX library was used for the MIP solution. Every experiment/measurement was repeated 10 times, changing a seed value that leads to slightly different infrastructure and VNF characteristics, following always the range of values provided in Table IV.

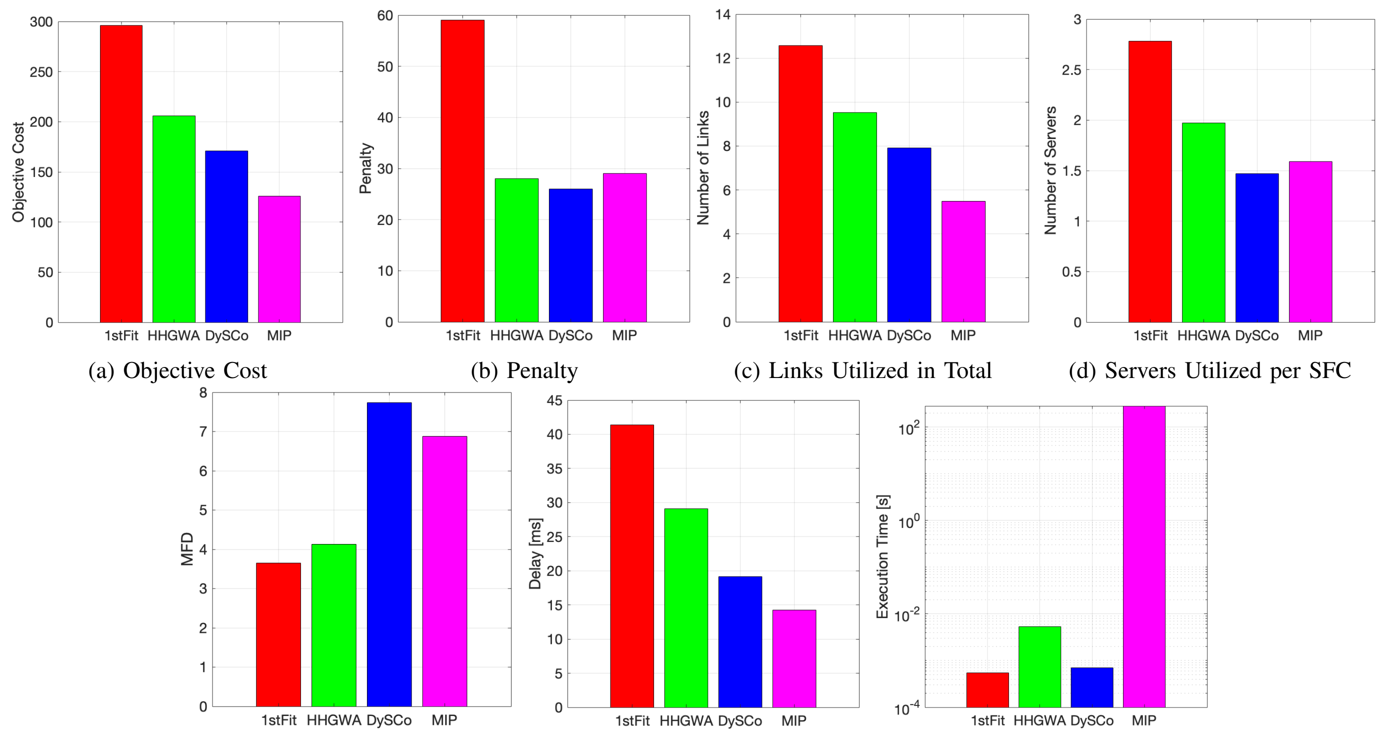
C. Results

1) Dynamic Scenario: In this family of experiments, we used the small infrastructure and 10 SFCs, while the results are illustrated in Fig. 7. As explained above, we run ten different tests for each algorithm and at each iteration, we only change the SFC and the server characteristics. After each test, we extract the average value of each metric under evaluation, for all the SFCs. Then we average all results for the ten different tests and plot the final values. The reason we first performed such a small-scale evaluation, is because we wanted to have a direct comparison with MIP, which was proved to be intractable for larger scale scenarios.

Fig. 7a showcases the objective cost for each algorithm, which consists of the sum of the penalties between the differ-

TABLE V: Summary of the compared algorithms.

Algorithm steps		**1stFit**	**HHGWA**		**DySCo**	**MIP**
1	**Sorting servers**	-	-		by Priority Factor	Find optimal placement that solves the problem
	Classifying VNFs	VNFs with Edge/Cloud constraints → put them at Edge/Cloud accordingly VNFs with no constraints → look for harmony between a_{k-1}, a_k and a_{k+1}				
	Sorting VNFs	by original SFC order	by Average request Ratio		by requested CPUs	
2	**Checking availability**	If the infrastructure has enough capacity → Proceed to SFC allocation Else → Reject entire SFC				
3	**Choosing the server**	the 1^{st} available randomly	by Grey Wolf Algorithm	Recovery after failure	by Priority Factor	
	Choosing NUMA node	the 1^{st} available by server's order	by remaining resources utilization		by Priority Factor	
	Choosing CPU cores	the 1^{st} available by server's order	the 1^{st} available by server's order		Mechanism of CPU cores consolidation	



(a) Objective Cost (b) Penalty (c) Links Utilized in Total (d) Servers Utilized per SFC

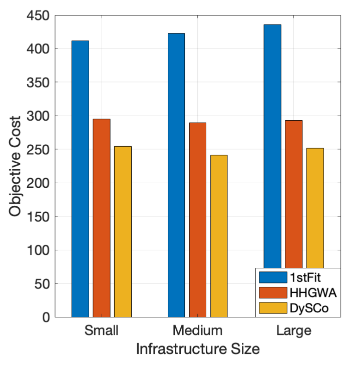
(e) Maximum Free Distance (f) Propagation Delay per SFC (g) Execution Time per placement

Fig. 7: Dynamic Scenario Evaluation: Small Infrastructure, 10 SFCs.

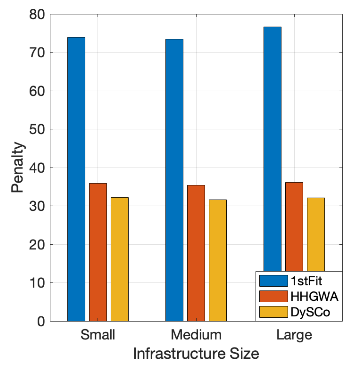
ent utilized cores, plus the consumed bandwidth on the links, as defined in Eq. (15). The total penalties are also highlighted in Fig. 7b. As we see, DySCo performs the closest to the optimal solution (MIP) in terms of objective cost, followed by HHGWA and then 1stFit. When it comes to the penalty in particular, DySCo outperforms all the compared algorithms, even MIP. This is the result of the core consolidation mechanism used by DySCo, which favors allocation by blocks, while avoiding anti-blocks as much as possible, to reduce the CPU to memory communication overheads. The MIP placement yields a 13.6% higher penalty than DySCo, since it uses a multi-objective function that not only optimizes the penalty, but also the consumed bandwidth. In more detail, the MIP potentially allocates SFCs in less attractive CPU core positions compared to DySCo, if this allocation scheme provides a lower overall objective cost. Similarly, since HHGWA has a goal to optimize the NUMA node selection it also achieves a lower penalty than MIP. However, the absence of a fine-grained CPU block definition yields a 9.13% higher penalty for HHGWA compared to DySCo. Finally, 1stFit scores the highest penalty and objective cost because of the absence of any optimization during NUMA node and CPU core selection.

Fig. 7c presents the total number of links (for all the SFCs) and Fig. 7d the number of servers utilized per SFC for the placement by each algorithm (results are averaged over 10 simulation runs). The MIP uses a slightly higher number of servers to deploy an SFC (1.59) compared to DySCo that requires the least (1.47). However, it seems that MIP performs better in routing optimization, as it uses less links than DySCo overall. This comes as a reasonable outcome, since MIP jointly solves the VNFs placement and their interconnections, by selectively choosing complete SFC embeddings over the infrastructure, rather than first allocating VNFs and then finding their proper interconnections. DySCo mainly emphasizes on the CPU consolidation that reduces the overall penalty and the number of used servers. However, indirectly, by selecting neighboring servers during the MSA case, it tries to also reduce the number of links, attenuating thus the gap with the optimal solution. In contrast, HHGWA again falls short compared to DySCo, as although it optimizes server and NUMA node selection, it does not leverage that well the possible creation of anti-blocks for future SFCs as DySCo does. This reflects in DySCo utilizing 16.84% less links and 25.38% less servers than HHGWA respectively. Finally, the lack of any CPU allocation optimization during the placement results in the 1stFit algorithm performing the worst.

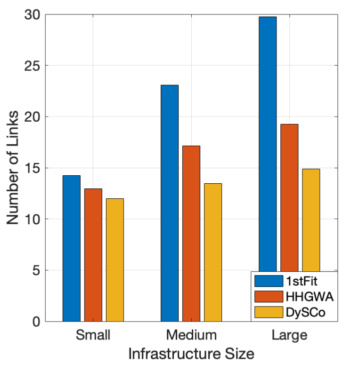
To investigate further the performance of the CPU cores consolidation mechanism used by DySCo, we devise a metric called *Maximum Free Distance (MFD)*. This metric represents the maximum number of consecutive available CPU cores in a NUMA node (the higher the better). Higher distance ensures higher availability to accommodate future VNFs, which promotes server consolidation, and reduces the cost of deployment. Fig. 7e shows the average MFD per server for the compared algorithms. Once again DySCo outperforms the others by scoring the highest MFD (7.74), followed by MIP (6.88). HHGWA comes third, while 1stFit once more give the worst results in terms of MFD. Moving on to the evaluation of the average propagation delay achieved per SFC



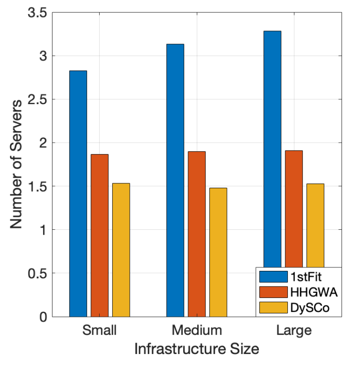
(a) Objective Cost



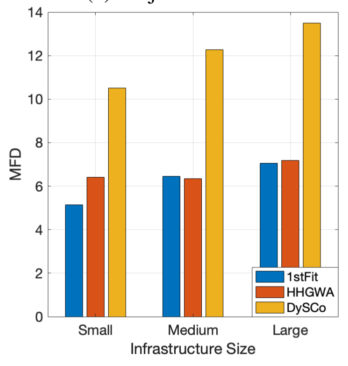
(b) Penalty



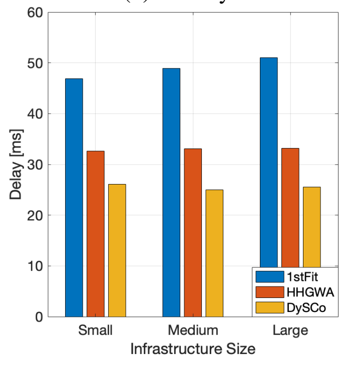
(c) Links Utilized in Total



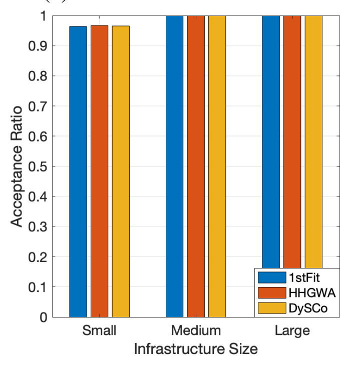
(d) Servers Utilized per SFC



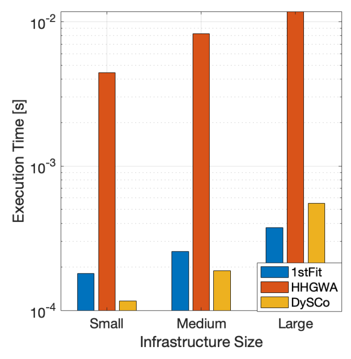
(e) Maximum Free Distance



(f) Propagation Delay per SFC



(g) Acceptance Ratio



(h) Execution Time per placement

Fig. 8: Dynamic Scenario Evaluation: Small-Medium-Large Infrastructure, $15 - 105$ SFC Requests (Scalability).

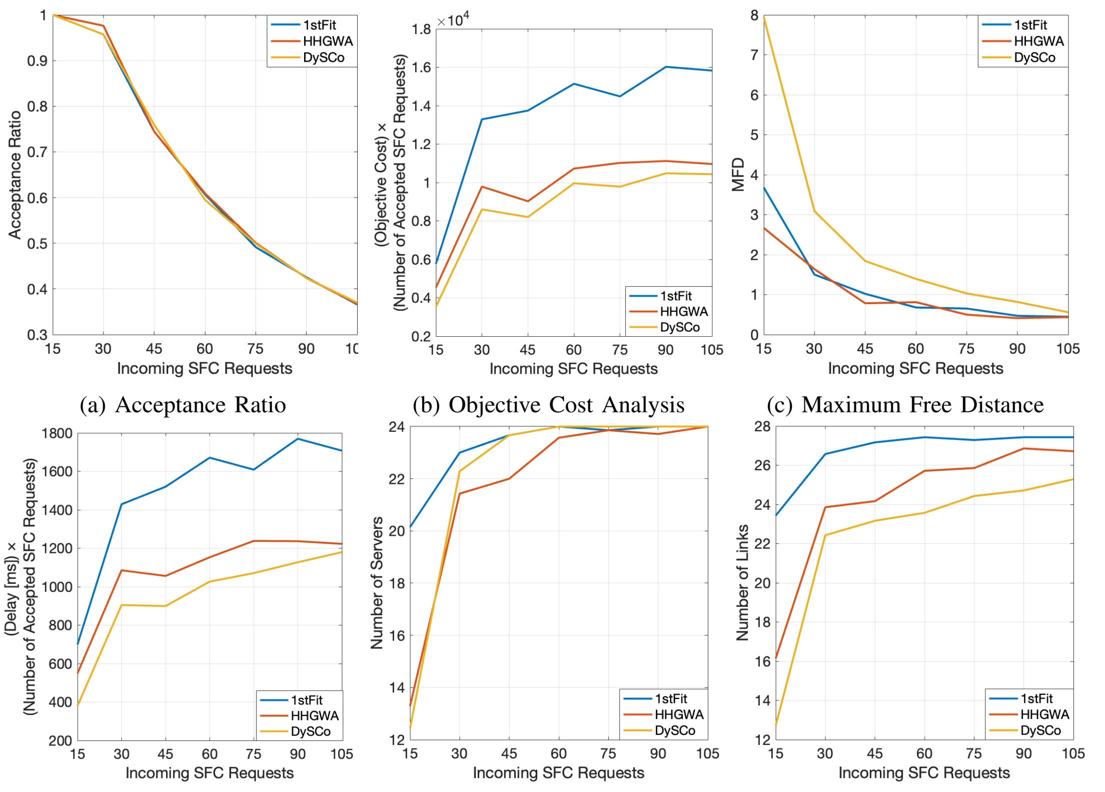
(Fig. 7f), the MIP comes out on top. As already noted, MIP optimizes the routing between the different VNFs of a given SFC thus uses the least number of links with the shortest distance, which results in the lowest delay. DySCo adds a couple of ms compared to the optimal, followed by HHGWA (twice than the optimal) and 1stFit (three times more than the optimal). Specifically compared to HHGWA, DySCo achieves a 34.2% lower propagation delay.

Finally, Fig. 7g illustrates the average execution time per SFC placement, in a logarithmic scale. As expected, the MIP, having the highest computational complexity, needs $283.62s$ on average to place a single SFC. On the other hand, the greedy 1stFit is the fastest one taking only $0.54ms$. DySCo goes hand in hand with the greedy solution ($0.7ms$), while HHGWA takes around $5.29ms$, which is still fast compared to the MIP, but 7.5 times slower compared to DySCo, due to the increased complexity of the Grey Wolf algorithm.

Since we have established the positioning of DySCo and the benchmarking algorithms with respect to MIP, we then investigate the scalability of DySCo under various network loads and infrastructure sizes. To do so, we generate a workload between 15 to 105 SFCs, with a step of 15. We execute seven iterations for each algorithm and each workload setting, only changing the characteristics of the servers and SFCs per iteration, randomly, from a predefined set as in Table IV. We then average the obtained metrics. The same set of experiments is repeated for the three infrastructure sizes and the results are plotted in Fig. 8. We note here that the MIP is excluded from this set of experiments due to its high computational complexity which leads to intractable execution times and memory requirements for large infrastructures and high number of SFCs.

Fig. 8a illustrates the objective costs and Fig. 8b the penalties as a function of the size of the infrastructure, for the compared algorithms. Keeping up with the previous set of experiments, 1stFit performs the worst due to the absence of any resource allocation during the placement and its performance deteriorates as the infrastructure size increases. Oppositely, HHGWA shows an almost stable performance regardless of the size. The explanation for that is that optimizing servers and NUMA nodes guarantees a certain performance stability. Naturally, DySCo showcases a stable performance throughout the different sizes as well, and at the same time yields the lowest penalty and objective cost. On average, DySCo achieves a 17.53% lower objective cost compared to HHGWA and 70.01% lower than that of 1stFit. This highlights the capability of the proposed core consolidation mechanism to ensure the best stable performance regardless of the network size.

In terms of average total utilized links, as seen in Fig. 8c, 1stFit shows the highest increase when executed in a large infrastructure compared to a small one, doubling the utilized links from 14.24 to 29.72. On the other hand, HHGWA shows a link increase close to 40%, while DySCo results in the lowest one, 24.11%, which is also 39.72% lower than HHGWA. A similar behaviour is demonstrated in the average number of utilized servers per SFC (Fig. 8d). Again, 1stFit performs poorly, with an increase rate of 16.26% between the small and the large infrastructure. HHGWA and DySCo show an almost stable performance regardless of the infrastructure size, with DySCo using on average 21.87% less servers than HHGWA.



(a) Acceptance Ratio (b) Objective Cost Analysis (c) Maximum Free Distance

(d) Delay Analysis (e) Servers Utilized per SFC (f) Links Utilized in Total

Fig. 9: Static Scenario Evaluation: Medium Infrastructure (Fixed), $15-105$ SFC Requests.

Fig. 8e showcases the MFD metric for each algorithm, for the different infrastructure sizes. DySCo achieves double the MFD attained by HHGWA and 1stFit; also, its MFD increases with the infrastructure size and this shows that DySCo favors the spread of the SFCs over the available servers providing more available resources for the future placements. This is a result of DySCo's dynamic server selection, during which it sorts them by their priority factor after each iteration, thus boosting the load balancing in the infrastructure. In Fig. 8f the average delay per SFC is illustrated. DySCo is again the best algorithm of the three, scoring the lowest delay, followed by HHGWA for which the average delay is 28.88% higher. Additionally, both algorithms show almost stable behaviour between the different infrastructure sizes, which highlights the importance of resource consolidation in guaranteeing stable performance. On the other hand, 1stFit fails to keep a consistent delay per SFC as the infrastructure size increases.

Moving on, we investigate the capability of each algorithm in successfully placing SFCs ("acceptance ratio"). As seen in Fig. 8g, all algorithms achieve to accommodate every incoming request for SFC placement in the medium and large infrastructures. The results are more interesting in the case of the small infrastructure, where the network resources are not enough to accommodate all the requests. There, HHGWA achieves the best ratio thanks to its server consolidation technique, however the differences among the algorithms are negligible. As shown in Fig. 8h, the slightly better acceptance ratio for HHGWA comes with a price in its execution time per SFC placement, which also increases with the infrastructure size. This happens due to the increased complexity of the Grey Wolf Algorithm. On the other side DySCo achieves significantly faster execution times, similar to those of the greedy 1stFit, with $0.28ms$ and $0.26ms$ on average respectively, a reduction of around 96% compared to HHGWA.

2) Static Scenario: For the second family of experiments, we fix the infrastructure size to medium, and we assume that SFCs arrive and remain in the infrastructure until the end of each experiment. The MIP is again excluded for reasons already stated. The results are plotted in Fig. 9 in an incremental step of SFCs. Fig. 9a displays the SFC acceptance ratio for each algorithm. As seen here, not all the incoming SFC requests can be accommodated when they are more than 30 (infrastructure saturation point) and rejections begin to occur. The compared algorithms show similar performance, with DySCo being slightly better than the other two. In order to visualize the variation of the objective cost as a function of the number of SFCs, we plot the curve of the objective function multiplied by the number of accepted SFCs in Fig. 9b. We see that the objective cost increases with the number of allocated SFCs; this happens due to the increase in the number of links and servers utilized in order to accommodate a growing number of SFCs. DySCo manages to score the lowest cost for all different tested number of SFCs, followed by HHGWA and 1stFit, which once more performs the worst.

Fig. 9c illustrates the MFD, which decreases exponentially for all the compared algorithms as the number of incoming SFC requests increases, an expected behaviour as no SFC departures take place. Like in the previous family of experiments, DySCo guarantees the best average MFD out of the three, balancing the load more efficiently among the infrastructure's servers. A short delay analysis follows in Fig. 9d; here, we display the cumulative delays of all SFCs multiplied by the number of accepted SFCs, which allows for a crisper illustration of the delay behavior for the three algorithms. Once again, DySCo scores the lowest delays, followed by HHGWA, with an increase of 15.18%, and then 1stFit.

In the final part of the evaluation, we compare the total number of servers utilized to place the SFCs and the total number of links to interconnect them. In Fig. 9e we notice between 30 (saturation threshold) and 60 incoming SFC requests, the HHGWA uses the least number of servers (7.57% fewer than DySCo and 1stFit). This is due to the main objective of HHGWA which seeks to consolidate as many servers as possible. However, the proposed core consolidation mechanism ensures that DySCo performs on average very close to HHGWA. Additionally, Fig. 9f shows that DySCo utilizes the least number of links in total, compared to the other algorithms; on average 8.12% less compared to HHGWA and 19.46% less compared to 1stFit. This happens because the server consolidation mechanism used by HHGWA, although it manages to keep the total number of servers low, it causes an increase in the number of utilized links, since it does not pay attention in a multi-server allocation scenario which two or more servers to interconnect. In contrast, DySCo utilizes slightly more servers, but fewer links to interconnect them, since neighboring servers will be usually selected.

D. Discussion

To summarize, after thoroughly evaluating the proposed mechanism under different experiment families, we conclude that DySCo dominates when compared to baselines from the literature. Apart from performing close to the optimal solution (MIP) in terms of objective cost achieved, the proposed mechanism additionally manages to minimize the introduced penalties and subsequently the deployment cost. This is a reflection of two factors; first, the emphasis on CPU-core consolidation, which significantly reduces the overall number of utilized servers. Second, DySCo's strategic approach to server selection, which prioritizes neighboring servers, that mitigates the number of links used. Notably, the improvements extend beyond optimizing the resource utilization; the proposed mechanism demonstrates close-to-optimal performance in minimizing end-to-end delay for individual SFCs as well. When it comes to its scalability, DySCo performs robustly across various infrastructure sizes and workloads, achieving the lowest and most stable utilization of links and servers, while keeping the delay again minimum. As a byproduct, this fine-grained server and core selection promotes load balancing and ensures enough resources for future VNF placements, while the system throughput is improved as well. Impressively, DySCo achieves the above enhancements in a fraction of the execution time reported by the competition.

VI. Conclusion

In this work, we propose DySCo, a novel VNF/SFC allocation algorithm that optimizes not only the server selection, but also the NUMA node selection and CPU core allocation in it, in an Edge-Cloud setting. Our goal was mainly to prioritize the use of L1 and L2 cache memories instead of the L3 and remote RAM, which can add significant performance bottlenecks. At the same time and as a secondary goal we tried to reduce the end-to-end delay and the total bandwidth consumption of the SFC and subsequently the deployment cost. To evaluate the performance of the proposed solution, we compared it to the optimal MIP case, as well as two other algorithms: a greedy approach, called 1stFit and HHGWA, a solution from the literature that also optimizes both the server and the NUMA node selection. The results show that, due to its CPU core consolidation mechanism, DySCo guarantees performance stability even at large-scale infrastructures, while performing close to the optimal and in real time.

As part of our future work, first we are planning to investigate the modeling of the NUMA nodes as an extension of the network graph, which could potentially allow for adopting generic VNF placement techniques to enhance DySCo's performance. Then, we intent to assess the performance of DySCo in a real-world environment, to better illustrate the direct benefits of reducing the delay overhead and throughput bottlenecks caused by a poor NUMA node and CPU core allocation. Finally, aspects of resource utilisation prediction through appropriate Machine Learning techniques will be studied and incorporated to the proposed approach to further improve its performance in the long term.

Acknowledgment

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Grant no. RGPIN-2019-05250.

References

[1] N. Alliance, "5G white paper," *Next generation mobile networks, white paper*, vol. 1, 2015.

[2] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Optimal virtualized network function allocation for an SDN enabled cloud," *Computer Standards & Interfaces*, vol. 54, pp. 266–278, 2017.

[3] A. Laghrissi and T. Taleb, "A Survey on the Placement of Virtual Resources and Virtual Network Functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2019.

[4] M. Diamanti, P. Charatsaris, E. E. Tsiropoulou, and S. Papavassiliou, "Incentive mechanism and resource allocation for edge-fog networks driven by multi-dimensional contract and game theories," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 435–452, 2022.

[5] M. Popov, A. Jimborean, and D. Black-Schaffer, "Efficient Thread/Page/Parallelism Autotuning for NUMA Systems," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 342–353. [Online]. Available: https://doi.org/10.1145/3330345.3330376

[6] J. Nelson and R. Palmieri, "Performance Evaluation of the Impact of NUMA on One-sided RDMA Interactions," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020, pp. 288–298.

[7] K. Hu, W. Lin, T. Huang, K. Li, and L. Ma, "Virtual Machine Consolidation for NUMA Systems: A Hybrid Heuristic Grey Wolf Approach," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020, pp. 569–576.

[8] G. Papathanail, A. Pentelas, and P. Papadimitriou, "Towards Fine-grained Resource Allocation in NFV Infrastructures," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[9] H. Yu, Z. Zheng, J. Shen, C. Miao, C. Sun, H. Hu, J. Bi, J. Wu, and J. Wang, "Octans: Optimal placement of service function chains in many-core systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2202–2215, 2021.

[10] S. Deng, Z. Xiang, J. Taheri, M. A. Khoshkholghi, J. Yin, A. Y. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1907–1923, 2020.

[11] S. Forti, F. Paganelli, and A. Brogi, "Probabilistic QoS-aware placement of VNF chains at the edge," *Theory and Practice of Logic Programming*, vol. 22, no. 1, pp. 1–36, 2022.

[12] N. Kiran, X. Liu, S. Wang, and C. Yin, "VNF placement and resource allocation in SDN/NFV-enabled MEC networks," in *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2020, pp. 1–6.

[13] L. Magoula, S. Barmpounakis, I. Stavrakakis, and N. Alonistioti, "A genetic algorithm approach for service function chain placement in 5G and beyond, virtualized edge networks," *Computer Networks*, vol. 195, p. 108157, 2021.

[14] L. Liu, S. Guo, G. Liu, and Y. Yang, "Joint dynamical VNF placement and SFC routing in NFV-enabled SDNs," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4263–4276, 2021.

[15] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Placement and chaining for run-time IoT service deployment in edge-cloud," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 459–472, 2019.

[16] G. Sun, R. Zhou, J. Sun, H. Yu, and A. V. Vasilakos, "Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6116–6131, 2020.

[17] N. H. Thanh, N. T. Kien, N. Van Hoa, T. T. Huong, F. Wamser, and T. Hossfeld, "Energy-Aware Service Function Chain Embedding in Edge–Cloud Environments for IoT Applications," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 465–13 486, 2021.

[18] Z. Xu, W. Gong, Q. Xia, W. Liang, O. F. Rana, and G. Wu, "NFV-enabled IoT service provisioning in mobile edge clouds," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1892–1906, 2020.

[19] Y. Yue, B. Cheng, X. Liu, M. Wang, B. Li, and J. Chen, "Resource optimization and delay guarantee virtual network function placement for mapping SFC requests in cloud networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1508–1523, 2021.

[20] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "VNF placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[21] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*, 2018, p. 285–292.

[22] C. Lameter, "NUMA (Non-Uniform Memory Access): An Overview: NUMA Becomes More Common Because Memory Controllers Get Close to Execution Units on Microprocessors." *Queue*, vol. 11, no. 7, p. 40–51, jul 2013. [Online]. Available: https://doi.org/10.1145/2508834.2513149

[23] C. Kim and K. H. Park, "Credit-based runtime placement of virtual machines on a single NUMA system for QoS of data access performance," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1633–1646, 2014.

[24] M. Agung, M. A. Amrizal, R. Egawa, and H. Takizawa, "The Impacts of Locality and Memory Congestion-aware Thread Mapping on Energy Consumption of Modern NUMA Systems," in *2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2019, pp. 1–3.

[25] A. V. Nori, J. Gaur, S. Rai, S. Subramoney, and H. Wang, "Criticality aware tiered cache hierarchy: a fundamental relook at multi-level cache hierarchies," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 96–109.

[26] R. Olaniyan, O. Fadahunsi, M. Maheswaran, and M. F. Zhani, "Opportunistic Edge Computing: Concepts, opportunities and research challenges," *Future Generation Computer Systems*, vol. 89, pp. 633–645, 2018.

[27] Matthias Falkner. (2019) Virtualizing Enterprise Network Functions. Cisco Systems. [Online]. Available: https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2019/pdf/BRKCRS-3447.pdf

[28] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

Taha Ben Salah is currently a Master of Science student in the Department of Information Technology Engineering at "École de Technologie Supérieure (ETS)" in Montreal, Canada. He received his bachelor's degree in IT and telecommunication engineering from "École Supérieure des Télécommunications de Tunis (SUP'COM)", Tunis, Tunisia in September 2018. His main research interests include Cloud and Edge Computing, optimizing data processing at the edge network, network function virtualization (NFV), and Non-Uniform Memory Access (NUMA) architecture.

Marios Avgeris received the Diploma degree in Electrical Engineering in 2016 and the Ph.D. degree in Electrical Engineering from the National Technical University of Athens in 2021. He has been a Postdoctoral Fellow with the Department of Systems and Computer Engineering, Carleton University and the Software and Information Technology Engineering department at the École de Technologie Supérieure (ÉTS) since 2021. His research interests include edge and cloud computing, computational offloading, NFV placement and network optimization and management.

Aris Leivadeas (S'12-M'15-SM'21) is currently an Associate Professor with the Department of Software and Information Technology Engineering at the École de technologie Supérieure (ETS), Montreal, Canada. From 2015 to 2018 he was a postdoctoral fellow in the Department of Systems and Computer Engineering, at Carleton University, Ottawa Canada. In parallel, Aris worked as an intern at Ericsson and collaborated with Cisco in Ottawa, Canada. He received his diploma in Electrical and Computer Engineering from the University of Patras in 2008, the M.Sc. degree in Engineering from King's College London in 2009, and the Ph.D degree in Electrical and Computer Engineering from the National Technical University of Athens in 2015. His research interests include Network Function Virtualization, Cloud and Edge Computing, IoT, and network automation and management. He received the best paper award in ACM ICPE'18 and '23 and IEEE iThings'21 and the best presentation award in IEEE HPSR'20.

Ioannis Lambadaris received his diploma in Electrical Engineering from the Polytechnic School of the Aristotle University of Thessaloniki in 1984, the MSc degree in Engineering from Brown University in 1985, and the Ph.D degree in Electrical Engineering from the University of Maryland in 1991. After finishing his graduate education, he was a Research Associate at Concordia University, Montreal, QC, Canada, from 1991 to 1992. Since September 1992, he has been with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada, where he is a professor in the same department. His research interests include performance analysis of computer communication networks, resource allocation, and optimization algorithms. While at Carleton University, he received the Premiers Research Excellence Award, and the Carleton University Research Excellence Award (2000–2001), for his research achievements in the area of modeling and performance analysis of computer networks.