Received ; | Revised ; | Accepted

DOI: xxx/xxxx

SPECIAL ISSUE ARTICLE

Composing Network Service Chains at the Edge: A Resilient and Adaptive Software-Defined Approach

Pradeeban Kathiravelu[1,2] | Peter Van Roy[2] | Luís Veiga[1]

[1]INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
[2]Université catholique de Louvain, Louvain-la-Neuve, Belgium

Correspondence
Pradeeban Kathiravelu. Email: pradeeban.kathiravelu@tecnico.ulisboa.pt

Present Address
INESC-ID Lisboa, Rua Alves Redol 9, Lisboa 1000-029, Portugal

Abstract

The scale of data and the demand for high data rates in applications are rapidly increasing over the years. By moving the computing resources closer to the users, edge computing minimizes the latency caused by otherwise pulling the data between the user and a centralized platform. Thus, data centers and computing nodes at the edge continue to overtake or replace traditional centralized cloud platforms in various 5G applications and use cases.

Network Functions Virtualization (NFV) virtualizes dedicated hardware network middleboxes into Virtual Network Functions (VNFs) and deploys them on top of servers and data centers. Due to the inherent bandwidth cost associated with the distance between the network services and the end user, more and more third-party VNF providers choose to deploy their services at the edge, rather than in a centralized cloud.

A typical Network Service Chaining (NSC) consumes various network services, where one's output becomes the input of another. As network services are often latency-sensitive, NSC needs to be carried out maintaining proximity to the user, as well as among the services composing the NSC. In this paper, we present the design of a resilient and adaptive framework, following an approach inspired by Software-Defined Networking (SDN), for the remote users to construct Network Service Chains (NSCs) in the Mobile and Edge Computing (MEC) environments. Our evaluations highlight the efficiency and optimality of the framework in finding the best-fit among several third-party edge VNFs in deploying the user NSCs, abiding by various user policies.

1 | INTRODUCTION

Latency-sensitive Internet services such as high-frequency trading[1], online gaming[2], and remote surgery[3] are reaching geographically diverse locations, far from the tier-1 cities that typically host the cloud data centers. These services operate with real-time constraints on latency and a need for stable bandwidth. Therefore, the proximity of these service deployments to the end-user plays a significant, often critical role in the users' Quality of Experience (QoE). Cloud platforms are opening up more regions, availability zones, and Points of Presence (PoP)[4], to cope up with latency-sensitive and data-intensive web applications. Thus, distributed cloud architectures[5] are getting more mainstream, as a compromise between the centralized cloud and

independent on-premise deployments, where the computation and data are distributed or replicated across multiple geographical regions near to the users.

Edge Data Centers

Network services perform better when they are deployed and served from edge data centers, compared to the traditional cloud platforms, due to the increasing demand for low latency[6]. This preference for a locality-aware execution leads the VNFs to be hosted at the edge to serve the large and geographically-distributed user base. Therefore, the infrastructure and the service executions keep moving closer to the user[7]. Recently, many edge providers (such as 365 Data Centers[8], vXchnge[9], and EdgeConneX[10]) are opening up their data centers in several tier-2 cities far from the typical cloud regions, with a long-term vision of covering a vast area of the globe, starting from the USA and the EU. Edge data center providers such as vXchnge promise an uptime higher than 99.99999%, thus placing themselves a better candidate for the Internet of Things (IoT) applications[11], rather than a centralized cloud[12].

On-premise deployment of VNFs offers bandwidth-efficient execution for interactive I/O bound applications. Nevertheless, configuring all the VNFs in-house is not feasible for everyone due to the deployment and maintenance costs as well as the resource limitations. For example, thin IoT clients do not have resources to host their VNFs in them due to resource scarcity. Edge data centers offer the benefits of proximity typical for the on-premise data center deployments, while still providing the separation of infrastructure from the applications of the user as in a cloud platform[13]. Therefore, several network services are currently moved to the edge of the networks. Realizing this market potential, several of the edge providers[8,9,10] offer numerous network services (such as captcha verification[14] and firewalls[15]) in addition to the infrastructure.

Network Softwarization

SDN and NFV enable network softwarization, by separating the control of the data plane devices into a logically unified network controller while consolidating the network functions as software middleboxes. SDN facilitates a global awareness and controllability of the network data plane devices. NFV virtualizes various network services and deploys them in servers instead of having them as individual proprietary hardware middleboxes. Network softwarization minimizes the capital and operational expenditures (CAPEX and OPEX), as the software middleboxes are cheaper to acquire and more flexible compared to hardware middleboxes, and a softwarized network is more cost-efficient to manage. By managing the network via a software controller, network softwarization enables data center networks to scale out towards the edge[16]. Thus, network softwarization continues to revolutionize edge computing, with its support for a dynamic formation, configuring, and programming networks through software constructs[17].

Challenges in Composing NSCs at the Edge

Finding the best-fit VNFs that offer a high QoE while abiding by the user policies remains a significant challenge in the multi-user edge environments. Mobile environments face further hurdles due to their resource-constrained nature and the requirement for live migration of data and execution. By increasing the number of deployment and execution alternatives, the rise of the distributed and decentralized edge nodes has further increased the complexity of finding an optimal resource placement at the edge. Current edge providers, despite their increasing number, give limited control to the users on seamlessly choosing the VNF offering based on their policies.

NSC (also known as SFC, Service Function Chaining) is a chain of network operations or middlebox actions created in a programmable and configurable manner[18]. NSC is considered a core concept in 5G deployment, enabled by SDN[19] and NFV[20]. While Application Programming Interfaces (APIs) of the VNFs need to be compatible and interoperable with each other[21], the entire service workflow should be carried out in a network-aware manner for a high-performance and QoE-compliant NSC. The VNFs should be placed strategically to minimize communication overheads and number of hops between the services during the execution of an NSC workflow. Each VNF in an NSC has its specific requirements, constraints, and objectives. Due to the various number of network services each NSC is composed of, challenges inherent to the VNF placement multiply in NSC scenarios. Ideally, a user should be able to construct her NSCs, leveraging VNFs from various providers at the edge. However, seamless migrations and interoperability among the VNF providers are significantly lacking. Therefore, the users are typically locked to a single provider for the execution of an entire NSC.

Various attempts have been made to mitigate the current challenges faced by the VNF execution in MEC and 5G environments. The research proposes software-defined approaches for efficient operation of the 5G environments, for network services such as selective forwarding attack mitigation[22]. D-NFV Alliance[23] purposes VNFs for MEC and 5G. Nevertheless, the current researches limit their focus to i) deploying VNFs in the MEC and 5G environments, ii) composing NSC in the data center and

cloud environments, and iii) optimal VNF placement by a service provider for NSC. Efficiently constructing NSCs by a user in a MEC environment requires chaining services spanning multiple data centers and nodes of various scale, often offered by different providers (i.e., different vendors) from numerous network domains. Therefore, it brings in more challenges, concerning secured data access[24] and optimal VNF selection considering the locality and user policies for minimal latency and high QoE. The volume and variety of services at the edge, as well as numerous policies defined by the edge VNF users for their NSCs, make VNF placement and resource allocation of NSCs in the MEC environments a multidimensional optimization problem that has not been studied well yet.

Motivation

Given the above premises, we aim at addressing the following research questions in this paper:

(RQ_1) Can we simplify and generalize the VNF allocation for seamless execution of the user NSCs at the edge, in the presence of several third-party edge VNF providers and numerous users?

(RQ_2) Can we bring the control of the NSC back to the user from the service providers, despite using third-party edge VNFs?

(RQ_3) Can we optimally provision the user NSCs across multi-vendor edge VNFs, adhering to the user-defined policies?

(RQ_4) Can we scale SDN to compose NSCs at the edge in a resilient and agile manner?

Contributions

The goal of this paper is to answer the identified research questions, focusing on 5G and MEC environments. The main contributions of this paper are:

1. A scalable and optimal framework for the NSC placements at the edge, consisting of an SDN architecture extended with Message-Oriented Middleware (MOM)[25] for global awareness of the VNF nodes. (RQ_3 and RQ_4)

2. Generalizing the challenge of resource allocation, service discovery, and workflow migration on the edge platforms for an adaptive execution of NSC as Mixed Integer Linear Programming (MILP) problems. (RQ_1)

3. A graph-based approach to solving the MILP models from the perspective of a remote user of third-party edge VNF providers. (RQ_1 and RQ_3)

4. Adaptive algorithms, executing independently and decentralized from the devices of the VNF users, to optimally provision the user NSCs across the edge VNFs for a locality-aware and policy-aware execution. (RQ_2 and RQ_3)

We implemented *Évora* (**E**dge **V**NF **O**rchestration with **R**esilience and **A**gility), to construct and deploy NSCs at the edge. In this paper, we elaborate in detail, how *Évora* tackles the current shortcomings in composing NSCs at the edge by proposing a software-defined approach. We deployed and performed an experimental evaluation of *Évora* for NSC execution at the edge. In particular: (i) we evaluated the correctness and optimality of the *Évora* in its NSC allocations adhering to the user-defined policies, and (ii) we assessed the complexity and scalability of the *Évora* graph-based approach in solving the optimal VNF allocation, posed as the MILP problems. The results obtained indicate that *Évora* ensures that the user-defined policies are met in allocating the user NSCs across the edge VNFs.

Évora considers various system constraints such as bandwidth efficiency and economic aspects, as well as user-defined policies such as constraints on required throughput and uptime. *Évora* exploits its extended SDN architecture in a wide-area network to discover the VNFs. It is aware of the entire global network through its MOM messages. However, it chooses the edge VNFs locally for the execution of the NSC requests of the users, by wholly and exclusively searching the nodes identified via the MOM messages. Thus, by leveraging SDN and MOM as well as the edge environments, *Évora* follows a "think globally, act locally" approach. *Évora* ensures optimality in the NSC placement at the edge from the user perspective, while not sacrificing the performance of the user NSC and the VNF providers. *Évora* brings the control of the NSC back to the user, from the cloud and service providers, as it used to be in the days before the invent of multi-tenant cloud platforms. Thus, *Évora* transforms the user into the central entity of the NSC ecosystem, from being a passive participant. Furthermore, it supports the seamless inclusion of more network domains by adding their respective controller to the *Évora* ecosystem by subscribing through the MOM messages. Thus, it efficiently scales out for the MEC and 5G environments with several service nodes and users. Exploiting its MOM architecture, *Évora* also lets the edge VNFs to notify the VNF users on new resource availability (such as the availability of new

VNFs) as well as new resource constraints (such as congestion in an edge node, a VNF failure, or a failure of an edge node). Thus, it further supports dynamically migrating a user NSC to better VNFs and deploying stages of the NSC across several VNFs to offer minimal end-to-end latency.

Paper Organization

In the following sections, we first discuss our motivation for bringing NSC to the edge in Section 2. Section 3 elaborates the *Évora* graph-based approach and the MILP models to build NSCs at the edge. Section 4 presents the *Évora* algorithms in constructing the NSCs adaptively and resiliently. Section 5 describes the implementation of the *Évora* framework. We evaluate the *Évora* approach in Section 6 and discuss state of the art and related work in Section 7. Finally, we conclude the paper with the summary of the research and future work in Section 8.

2 | FROM NFV TO NSC: SERVICE EXECUTION AT THE EDGE

SDN and NFV paradigms combined with the advancements in the edge computing are paving the way forward for low-latency 5G networks[26]. In this section, we will elaborate the motivating scenario for the NSCs in the 5G environments, leveraging SDN and network services at the edge.

VNFs at the Edge

We need to minimize the number of hops and geographical distance among the service deployments as well as to the user for acceptable and improved performance of an NSC[27]. Figure 1 elaborates a sample scenario with various users (A, B, C, D, E, and F) and many edge nodes (n_1, n_2, n_3, n_4, n_5, and n_6) with multiple VNF offerings (s_1, s_2, s_3, s_4, and s_5). If the user D needs to consume the services s_1, s_2, s_3, and s_4 independently in parallel, based on the proximity, we can observe, in a small-scale scenario, that edge nodes n_3 and n_4 offer the best option. Such an observation, however, ignores the other parameters such as the Quality of Service (QoS) and Service Level Objectives (SLOs) for the sake of simplicity.

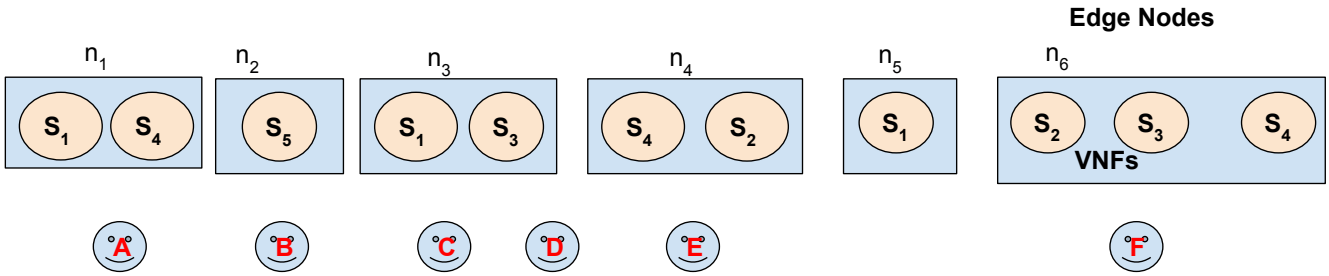


FIGURE 1 Service Deployments at the Edge

NSCs at the Edge

Now, we consider the scenario of an NSC: $s_1 \circ s_2 \circ s_3 \circ s_4$. Here the output of s_1 is forwarded as the input to s_2, and s_2's output to s_3, and s_3's output to s_4, and finally s_4's execution outcome is returned to the user. Figure 2 elaborates two potential execution alternatives. Given the dependencies between the services (since the execution of s_{n+1} depends on the execution and the outcome of s_n in this example) in an NSC, the proximity between the services becomes relevant too. Since the node n_6 hosts s_2, s_3, and s_4, the communication delay between the service deployments is minimal. Thus, in this scenario, execution in n_5 and n_6 can be a better candidate than the nodes n_3 and n_4, as n_3 and n_4 have more inter-node communications.

NSCs at the MEC and 5G Environments

Due to the dynamic nature of the MEC environments and the 5G applications, fast relocation and migration of services are essential for the QoS guarantees[28]. Consider a mobile user who is accessing an interactive online service (such as Voice over IP) through a protected environment. She connects with the nearest edge services that offer auxiliary services: firewall, parental control, spam protection, intrusion detection systems (IDS), Distributed Denial of Service (DDoS) mitigation[29], network address translation (NAT), WAN optimizer, and load balancer. The Internet traffic that reaches her goes through some or all of these services before finally reaching her, based on her policies on each web application or web address. We depict this scenario as

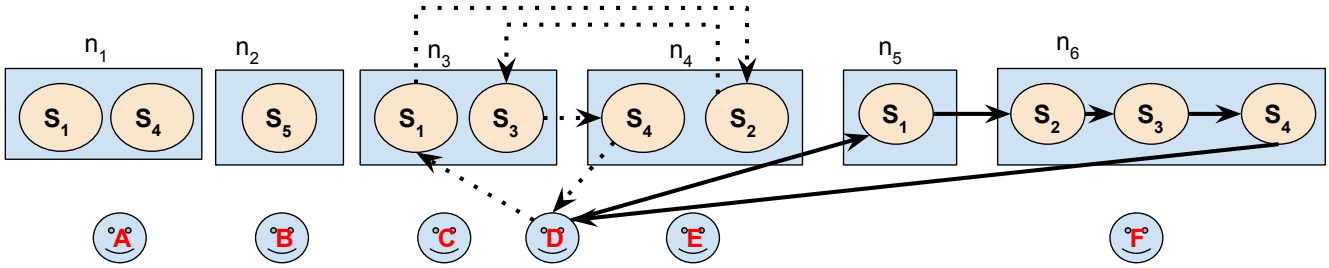


FIGURE 2 NSC at the Edge

an NSC of $s_5 \circ s_4 \circ s_3$ in Figure 3 . For the user A, VNFs hosted in n_2, n_1, and n_3 appear to be the closest options. However, the user can be moving in a vehicle, thus changing the location, proximity to the edge nodes, and mobile stations[30]. Therefore, the edge nodes that serve her would change over time based on her proximity. If the user moves from the initial position A_0 and reaches the new position A_1 in the middle of the NSC workflow (after the execution of s_5 in n_2), it becomes apparent that $n_2 \circ n_6$ becomes the new flow of data. Thus, the possibility to perform a live migration of the workload (for example, a data stream) to another deployment or node is crucial for the QoE. The user mobility needs securely sharing the execution state between the adjacent VNFs.

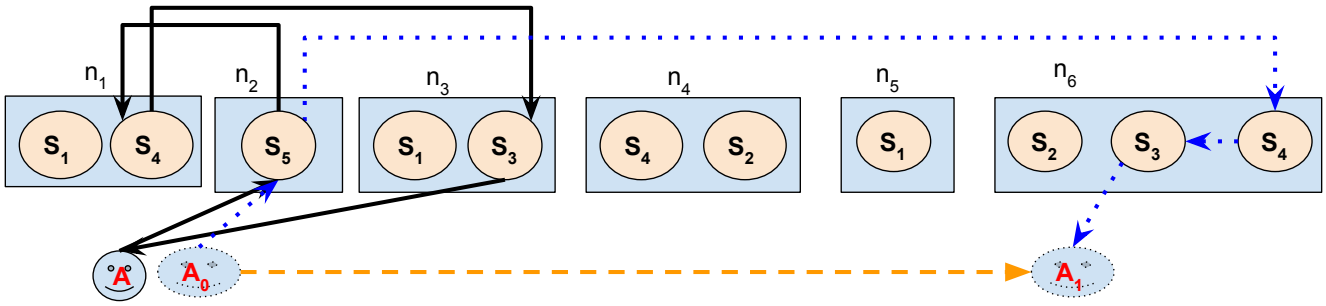


FIGURE 3 Mobile User of NSC at the Edge

For the sake of simplicity, we weighed only the geographical distance in this section through the simple illustrations. Nevertheless, the execution indeed needs to consider various other parameters and constraints, including the cost, bandwidth or data rate of each option, and the offered QoS.

Illustrative Example of a Parallel NSC Execution

Furthermore, an NSC should be able to branch out traffic to different VNFs depending on the user policies. For example, consider the scenario of parental-control for the Internet browsing. The Internet traffic may go through a firewall and virus scanner in a regular mode but go through the firewall and parental control before reaching the virus scanner before arriving at the children's browser sessions. Subsequently, an NSC can be represented by multiple forms of directed graphs as shown by Figure 4 . The flow is divided and sent in two different paths: one going through s_1 and s_2 and the other just s_1, before both merging to s_3 as in the above parental control scenario. The flow is then directed to s_4, where based on the policies, a certain matching subset of data reaches the user directly while the remaining goes through s_5. This NSC can be defined as $(s_1 \circ s_2 + s_1) \circ s_3 \circ (s_4 \circ s_5 + s_4)$, or it can be reduced to $s_1 \circ (s_2 + 1) \circ s_3 \circ s_4 \circ (s_5 + 1)$. The alternative representations increase the potential execution alternatives while also supporting parallel execution with redundancy in the paths.

An edge node can be anything from a small computing node such as a Raspberry Pi to a fully fledged data center at the edge. Executing NSCs at the edge is thus an NP-hard problem consisting of VNF placement, choosing the optimal service deployments, and workflow migration[31]. We envision a complete and independent end-to-end NSC orchestration from the user devices, together with logically centralized coordination of the edge environment. We propose to extend SDN for wireless and wide area networks to solve this resource allocation problem at the edge efficiently, thus letting the user dynamically define and compose NSCs in a decentralized manner.

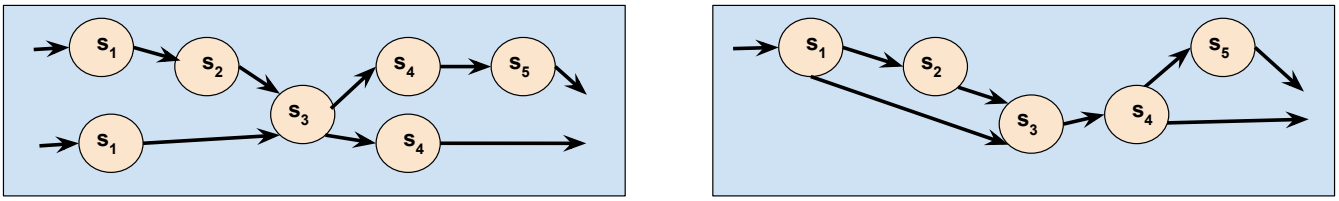


FIGURE 4 Parallel Execution Alternatives of an NSC

TABLE 1 Notation of the *Évora* Representation

G	The acyclic multigraph of edge nodes
H	The hypergraph of edge VNFs
$\mathcal{V}$	The set of compute nodes hosting the VNFs
$\mathcal{L}$	The set of links that connect the compute nodes
$\mathcal{N}$	The set of compute nodes considered by an *Évora* deployment
S	The set of VNFs considered by an *Évora* deployment
S_i	The set of VNFs in a node n_i
S_j	The set of VNFs composing an NSC ψ_j
s_*	A VNF instance; $s_* \in S$
X	The set of VNFs, annotated by the node deployment
I	The set of intra-node links between the VNFs of each node
E	The set of links that connect the edge VNFs
R	The set of resources (such as memory, CPU, and bandwidth) provided by the edge VNF provider

3 | *Évora*: EDGE VNF ORCHESTRATION WITH RESILIENCE AND AGILITY

Traditionally, NSCs have had few services serially chained. However, in a MEC and 5G environments, the need for service migrations and mobility has given rise to network services as VNFs and microservices. This rising number of VNF offerings has increased the scale and complexity in composing the NSCs[32]. In this section, we will propose a graph representation to the edge VNFs and formulate MILP models to optimally provision the user NSCs across the edge VNFs. Section 4 elaborates the *Évora* algorithms to solve the MILP problems.

3.1 | NSC at the Edge: Graph Representation

Évora models the MEC environment consisting of various interconnected computing nodes as an undirected acyclic multigraph $G = (\mathcal{V}, \mathcal{L})$[1]. $\mathcal{V}$ represents the set of nodes hosting the VNFs, and $\mathcal{L}$ represents the links that connect them. $\forall$i,j,k $\in \mathbb{Z}^+$, each node $n_i \in \mathcal{N} \subset \mathcal{V}$ consists of a set of services $S_i \subseteq S$ and $s_k \in S$. A user u_j can define her NSC ψ_j as a composite function or a service composition. These NSCs are composed of a set of services S_j; $S_j \subseteq S$. Thus, $\forall s_* \in S_j$; $\psi_j = s_a \circ s_b \circ \ldots \circ s_x$. Complete details on the notations are listed in Table 1 .

Since each node offers one or more VNFs, *Évora* denotes the service deployments in a hypergraph $H = (X, E)$, as an overlay on the node graph $G = (\mathcal{V}, \mathcal{L})$. *Évora* minimizes the NSC resource allocation problem into how to place ψ_j in the hypergraph H optimally. Each vertex of H can be either i) a single vertex (interchangeable with a vertex of G) or ii) a complete graph of multiple service deployments within a vertex of G. The former refers to a node that offers only a single network service, while the latter refers to one that offers multiple network services. $E = \mathcal{L} \cup I$, where I represents the set of intra-node links between the services in each node. X represents the services annotated with their node deployments; $\forall\, x_i \in X$, $x_i = n_j.s_k$, where s_k is any

[1]To avoid overloading the words 'node' and 'edge', we refer to the graph properties with 'vertex' and 'link'. In our terminology, 'edge' is strictly restricted to the edge environment, and 'node' is strictly restricted to the compute nodes at the edge.

of the services deployed in the node $n_j \in \mathcal{V}$. Since intra-node service composition is more bandwidth-efficient than inter-node executions, the edges of the hypergraph are differentiated in weight accordingly to represent the minimal latency between the E hosted in a given $\mathcal{V}$.

The multigraph G consists of control and data flows between the edge nodes. In this multigraph, some links represent the control flows among the edge nodes as well as with the mobile/edge user who holds an orchestrator that manages her NSC execution in the graph. Figure 5 illustrates the edge environment with the user. The user connects to the nearest nodes, through either a dedicated connection or the public Internet. The orchestrator instance in the user device detects and identifies the nodes required for the execution of each service in the NSC that the user has composed. The intra-node links of the hypergraph of Figure 5 , $I = \{n_1.\{s_1, s_4\}\}, n_3.\{s_1, s_3\}\}, n_4.\{s_2, s_4\}, n_6.\{s_2, s_3, s_4\}, n_7.\{s_1, s_3\}, n_{12}.\{s_3, s_4\}, n_{16}.\{s_3, s_4\}\}$. We interpret the edge environment of VNF deployments as a hypergraph with the service installations inside a node as vertices, and nodes themselves as supernodes, where a single hyperedge connects all the services inside a data center.

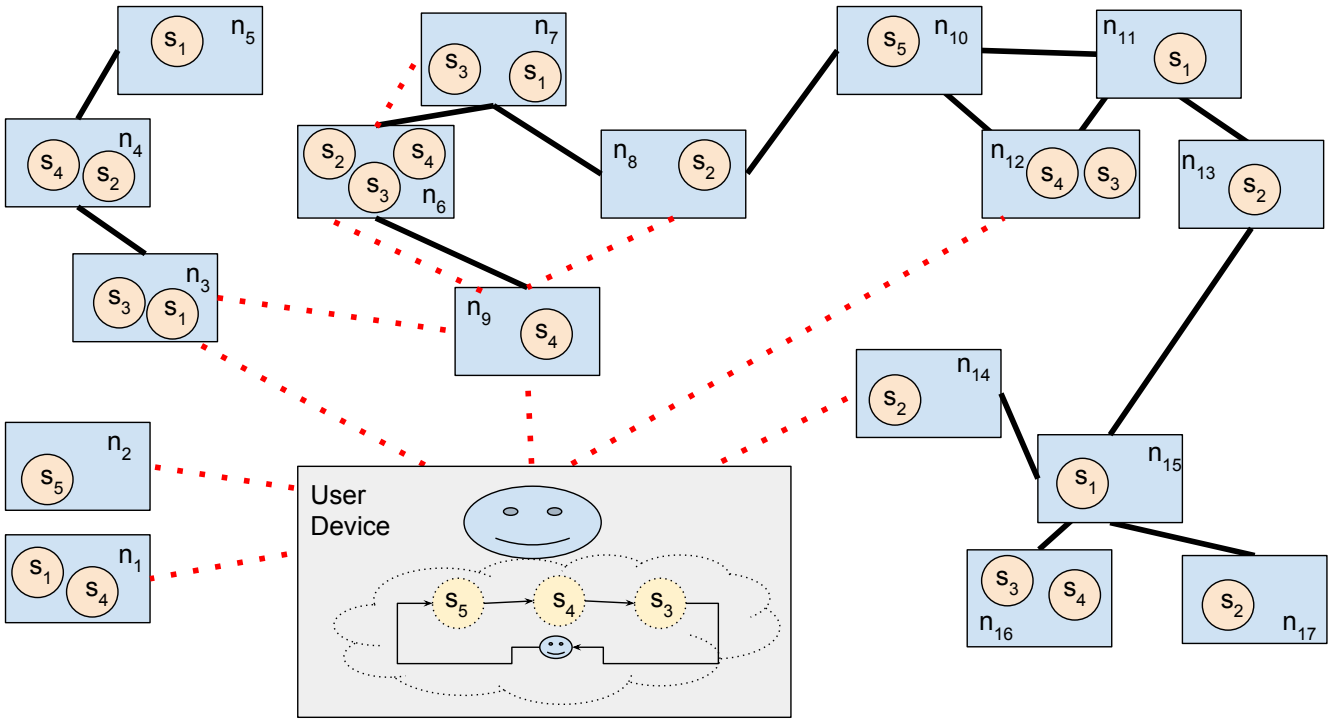


FIGURE 5 A User Defined NSC among the Edge Nodes

SDN protocols such as OpenFlow[33] have been extended for wide-area networks to create Software-Defined Wide Area Networks (SD-WAN)[34]. Edge data centers often have (either logically or physically) separate links for control flows instead of sharing the same links with the data flow, as data flows are heavy while control flows are light-weight and more critical. In Figure 5 , the dotted lines represent such separate links of control flows, including control flows of OpenFlow and SD-WAN controllers to manage the data plane. *Évora* considers an undirected acyclic graph with only the data links that are responsible for the VNF traffic in its service allocation, eliminating these additional control links from the model. The user defines her NSC and seeks the best-fit for the execution based on the policies of her NSC. In Figure 5 , the NSC is defined as $s_5 \circ s_4 \circ s_3$. To minimize latency, simply based on the distance between the edge nodes, the first and last VNFs need to be proximate to the user, whereas the services in the NSC themselves need to be close to each other. In the ideal case, all the services will be close to each other and the user. The proximity aims to minimize the latency caused by too many hops and the geographical distance between the user and services.

3.2 | NSC at the Edge: MILP Models

Edge VNF User Perspective

An NSC execution should satisfy the user policies by choosing the best VNF offerings among the available alternatives. The user policies may consist of several parameters, such as throughput, monthly cost, and latency. For example, the user needs to maximize the throughput (T) of her NSCs while minimizing the total monthly cost (C) and latency (L). To support user policies in a scalable manner, the MILP models are defined from the user perspective. The models are solved by algorithms executed from each VNF user device, for the user's own NSCs.

Equation 1 defines the latency L_s as the contribution to the overall latency of the NSC from a service s composing the NSC. τ_s represents latency of the service s. ϱ_s represents the latency between the ingress port and connecting link from the previous service, or the time taken for a unit data to arrive at the current service s from s_{-1} ($s_{-1}, s \in \psi$). Thus, $\tau_s + \varrho_s$ represents the time interval between the departure of a unit data from the VNF s_{-1} and the time VNF s outputs the data through its egress port. t_0^s and t_f^s represent the start and end time of the execution of any VNF s composing the NSC ψ.

$$L_s = \tau_s + \varrho_s = t_f^s - t_f^{s_{-1}} \tag{1}$$

Since the VNF provider can charge a differentiated unit price for the VNF offerings based on the current demand and resource availability, we define the cost incurred by a VNF offering by a time integral of the service unit cost over the VNF execution period. c_s denotes the unit cost to acquire a VNF s. Equation 2 describes the execution cost of s as part of a user NSC.

$$C_s = \int_{t_0^s}^{t_f^s} c_s dt \tag{2}$$

The edge VNF user should minimize her overall operational cost. Therefore, she must aim at reducing the total cost of all the services in the NSC, rather than performing a local minimization for each of her VNFs. As a global minimization problem, we find and store the effective amount composed of C_s, L_s, and T_s of each VNF of NSC as C, L, and T for each NSC, as depicted by Equation 3. Here, the latency and cost of the NSC are a simple sum of the individual values of them for each of the VNF in the NSC. The throughput of the NSC is limited to the value of the VNF that has the least of the throughput.

$$C = \sum_{s \in \psi} C_s; L = \sum_{s \in \psi} L_s; T = \{s \in \psi | min(T_s)\} \tag{3}$$

In practice, generally a VNF provider does not change the unit cost of a service within a brief time. Therefore, for a reasonably short NSC execution time, we can assume the unit cost of a service to be constant throughout. Thus, we can simplify C_s as $(t_f^s - t_0^s) \times c_s$. Maximum total volume of data passed through a VNF can be expanded as $duration \times throughput$, $(t_f^s - t_0^s) \times T_s$. Finally, we model the edge NSC allocation as a MILP problem as defined by Equation 4.

$$\alpha, \beta, \gamma \in \mathbb{N}, \textbf{minimize}(\alpha C + \beta L + \gamma T^{-1}) \tag{4}$$

T^{-1}, the inverse of throughput, denotes how much time it takes to process a unit data (measured in Mb) end-to-end through the NSC. For example, a service we hosted on Amazon EC2 offered 50 Mbps or 0.02 s/Mb. α, β, and γ are integers defined by the user as the policy for the NSC in the form of "$C\ \alpha; L\ \beta; T\ \gamma$". The policy gives weight to the parameters, relative to each other as in a ratio. It can also include further constraints and SLOs such as "$T \geq 50$" to indicate the user requirement to have a minimum end-to-end throughput of 50 Mbps for the NSC. The penalty value $F = \alpha C + \beta L + \gamma T^{-1}$ can be generalized to include more configuration options as in Equation 5. The minimization problem can also be extended to include additional properties (such as uptime, QoS guarantees, and carbon efficiency), policies, and constraints.

$$\alpha, \beta, \gamma \in \mathbb{N}, a, b \in \mathbb{R}_{>0}, \gamma \times c \in \mathbb{R}_{<0}, \textbf{minimize}(\alpha C^a + \beta L^b + \gamma T^c) \tag{5}$$

Edge VNF Provider Perspective

The VNF provider needs to be economically sustainable. Therefore, she needs to maximize the number of VNF deployments η_s for each VNF $s \in S_i$, to increase the income, while still not exceeding the total available resources $r \in R$, at any given time t. Equation 6 expresses these constraints while maximizing the overall platform profit over time. $\upsilon_{s,r}$ refers to the consumption of resource r by a single VNF deployment of s. Δc_s refers to the profit of each VNF, as a difference between the income c_s and

expense estimated for the particular VNF deployment ι_s. We limit our focus to resource optimization at the edge node as a single unit, though each node may consist of several servers with individual constraints in their resources.

$$\eta_s \in \mathbb{N}, \forall \epsilon \in \mathbb{R}_{>0}, \Delta c_s = c_s - \iota_s, \textbf{maximize}\left(\int_{t>\epsilon} (\eta_s \times \Delta c_s) dt \right). \\ \textbf{s.t.} \forall r \in R, r \geq \sum_{s \in S_i} \eta_s(t) \times \upsilon_{s,r}(t). \tag{6}$$

Thus, we spawn a VNF instance in the node only when it is profitable for the provider. New VNF instances are not spawned while the current profit is negative, and any existing instance may be terminated without affecting the active NSC workflows. Thus, when the VNF provider incurs an economic loss, the change in the number of any VNF instance will be a non-positive value as Equation 7 illustrates.

$$\forall s \in S_i, \Delta c_s(t) \leq 0 \Rightarrow \frac{d\eta_s}{dt} \leq 0. \tag{7}$$

Managing the VNFs inside each edge VNF provider (regardless of its nature or scale) is analogous to NSCs in data centers, a well-studied topic[31]. Therefore, though we formulate MILP models for both the user and the edge VNF providers, we focus on the edge user perspective of NSCs and solve the MILP optimization problem illustrated by the Equation 5.

4 | *Évora* ALGORITHMS

Each user device in the *Évora* ecosystem consists of an **Event Manager** that finds VNFs from the edge nodes, based on event notifications. Each user device also contains an **Orchestrator** that optimally allocates the NSCs among the VNFs identified by the Event Manager. The orchestrator executes algorithms that efficiently resolve the NP-hard problem of hypergraph matching[35] for the NSC allocation. It initializes the global variables that represent the node and service graphs.

For each user NSC, the orchestrator identifies the multiple execution paths and chooses the best fit for the current NSC based on its policy (which of the parameters to be minimized or maximized and their respective thresholds). It thus solves the MILP models efficiently for each user, for each of their NSCs along with their policies. The execution is performed only once per a user NSC defined with a given policy. The chosen execution path is stored for subsequent executions of the NSC with the same policies. Thus *Évora* reduces the overhead in finding the optimal VNFs to 0 in subsequent accesses.

4.1 | *Évora* Global Environment

Algorithm 1 defines the process of orchestrating the user environment by initializing and maintaining the environment consisting of graphs and other global variables. The global variables (defined in lines 2 - 5) are initialized by the *orchestrateEnvironment* procedure and later updated by events received as notifications from the edge nodes, based on the event subscriptions of the user.

The *orchestrateEnvironment* procedure takes a map of user-defined NSCs and their respective policies including SLOs (such as latency, throughput, and loss rate) and other objectives such as minimal cost. *initOrchestrator()* (invoked in line 8) initializes the *Évora* orchestrator that manages the VNF requests to chain them as NSC. *initEventManager()* (invoked in line 9) initializes the Event Manager, subscribes to the notifications as defined by the orchestrator, by providing the endpoint of the brokers co-located with a controller in edge nodes (line 11).

Upon initialization, the broker retrieves the details of edge nodes from the broker queue with the topic *INITIALIZE_EVENT* (line 12). *INITIALIZE_EVENT* is a system-defined global static constant, used as the topic for the data necessary to bootstrap the system in the broker. For each of the node received from the broker queue, its properties are added to the *edgeNodesMap* with the key of the node identifier (line 14). Furthermore, the *next nodes* are added to the *edgeNodesMap* with the key *{DATACENTER_ID}_+'NEXT'*, to represent the node graph (line 15), by denoting the properties of the links. The services of each node are added to the *servicesMultiMap* to represent the services hypergraph (line 16). Each service can be directly retrieved from the multimap by providing the node. The *initNscMap* is added as the keys of the nscMap global variable, as a <*nsc, policy*> tuple, with the value of each entry set to null (line 19).

The Event Manager listens to the relevant notifications published by the edge nodes. An event will be triggered for the availability of new nodes, unavailability or status update of an existing node, availability of new services in an existing node, and

Algorithm 1 Orchestrating the Environment

```
1:  global variables
2:      edgeNodesMap                                          ▷ <property of edge node, relevant value>
3:      servicesMultiMap                                      ▷ <node, deployed services>
4:      matchingSubgraphsMultiMap                             ▷ <nsc, matching paths>
5:      nscMap                                                ▷ «nsc, policy>, executionPath>
6:  end global variables
7:  procedure ORCHESTRATEENVIRONMENT(initNscMap<nsc, policy>)  ▷ The policy includes the user policies for the MILP models
8:      initOrchestrator()
9:      initEventManager()
10:     for (broker in brokers) do
11:         subscribe(broker)
12:         initNodes ← getEvent(broker, INITIALIZE_EVENT)
13:         for (node in initNodes) do
14:             edgeNodesMap.add(node.getKey(), node.getValue().properties)
15:             edgeNodesMap.add(node.getKey() + 'NEXT', node.getValue().links)   ▷ Linking the next nodes to the current node.
16:             servicesMultiMap.add(node.getKey(), node.getValue().services)
17:         end for
18:     end for
19:     nscMap ← <initNscMap, NULL>
20:     repeat
21:         if (eventReceived) then
22:             updateUserEnvironment(event)
23:         end if
24:     until (aborted)
25:     edgeNodesMap ← NULL
26:     servicesMultiMap ← NULL
27:     nscMap ← NULL
28:     matchingSubgraphsMultiMap ← NULL
29: end procedure
```

unavailability or status update of an existing service in a node. The event is triggered either by the action of the edge nodes or by the user herself such as moving her location from where she executes the NSC. When an event is received from one of the brokers in the edge nodes, the user environment and the global variables (*edgeNodesMap, servicesMultiMap, matchingSubgraphsMultiMap, and nscMap)* are appropriately updated by invoking the *updateUserEnvironment* procedure (lines 21 - 23). This update ensures that the NSC executions are dynamically updated to reflect the current status of the edge nodes. When the orchestrator is terminated, the global variables are reset to null (lines 25 - 28). As the *orchestrateEnvironment* procedure manages the user environment, the user has updated information on the available service deployments at the edge.

4.2 | NSC Execution Paths at the Edge

Algorithm 2 is an implementation of a graph matching problem, designed for adaptive NSC allocation at the edge. User-defined custom graph matching algorithms can replace this graph matching algorithm. This algorithm confirms the interconnection between the service deployments for a complete NSC execution, rather than merely confirming the existence of a VNF in a node from *servicesMultiMap*. The algorithm depicts the process of finding the potential execution paths for an NSC, which can be later used to determine the exact NSC execution flow for any given policy. It takes *nsc* defined by the user programmatically or through a configuration file, as well as a pointer to a graph traversal function **iTraverse()** as the input. Since the nearest nodes of the hypergraph represent the services in the same node, the algorithm by default chooses breadth-first traversal as the implementation of **iTraverse()**. Any user-defined traversal such as random walk can replace the traversal implementation. With the choice of the traversal algorithm, the constructed execution paths are already sorted to fit the search criteria.

partialNSCs is a list of arrays (line 2). Each array consists of multiple entries that are in the form of tuple <*service, currentNode*>. The list traces the potential NSC execution paths in the node graph, as the algorithm constructs them into partial chains

Algorithm 2 Finding NSCs at the Edge

```
1:  local variables
2:      partialNSCs                                          ▷ List{<currentNode, Service>[]}
3:  end local variables
4:  procedure INITNSC(nsc, iTraverse())
5:      partialNSCs.add(ϕ)                                   ▷ Initialize the list with an empty entry
6:      repeat
7:          currentNode ← iTraverse(edgeNodesMap)            ▷ Traverse the map, one node at a time
8:          for ((service in currentNode) and (service in nsc)) do   ▷ Services composing nsc found in the currentNode
9:              for ( (pNSC in partialNSCs) AND (service NOT in pNSC) ) do   ▷ Service found not to be in partialNSC elements
10:                 if ( ((edgeNodesMap.get(currentNode.getKey()+
                                'NEXT')) AND pNSC) <> NULL ) then
                                    ▷ A link exists between an element in pNSC and currentNode
11:                     pNSC.put(service, currentNode)
12:                 end if
13:             end for
14:             partialNSCs.add(newPartialNSC(service, currentNode))   ▷ Also add as a new pNSC
15:             for (pc in partialNSCs) do
16:                 if (pc.length = nsc.length) then         ▷ Array at full capacity. pc has been constructed as the NSC
17:                     pc.persistPenaltyValues(pc)          ▷ Add metadata to pc object
18:                     matchingSubgraphsMultiMap.add(nsc, pc)
19:                     partialNSCs.remove(pc)
20:                 end if
21:             end for
22:         end for
23:     until (allNodesTraversed)
24: end procedure
```

of the NSC. Each array in the list consists of maximum entries equal to the number of services in the NSC. These arrays grow with the number of service deployments found from the edge nodes, as a matching subset of the complete NSC. Similarly, the *partialNSCs* list itself grows as more potential execution alternatives are constructed. Thus, it provides chains of pointers to the matching node for each of the services in the NSC. The multiple arrays in the *partialNSCs* list indicate the possible multiple execution paths, stored against their service.

The algorithm initializes *partialNSCs* list with one entry, the default object ϕ (line 5). Then it repeatedly traverses the graph till it completes visiting each of the nodes. **iTraverse()** traverses the *edgeNodesMap*, one node at a time, returning the current node at each move (line 7). Then the algorithm finds whether the current node has one or more services that are part of the NSC (line 8), as it can readily retrieve the services available in each node from the *servicesMultiMap*.

For each array representing the constructed partial NSC *pNSC* in the *partialNSCs* list, the algorithm checks whether the service is not already present (line 9). Then it confirms that there are overlaps between a partial NSC *pNSC* and the 'NEXT' pointers of the current node stored in the *edgeNodesMap* to denote the links (line 10). An overlap indicates the existence of one or more links, showing the possibility to add the current node into the NSC. Thus, the algorithm puts the <*service, currentNode*> tuple into the *pNSC* array to add the next entry in the chain (line 11). For each node with a service matching one from NSC, a new object is created with the tuple <*service, currentNode*> as the first VNF, and added to the *partialNSCs* list (line 14). This approach gives space to construct new potential NSCs with the currently traversed node as the first entry. Equation 8 depicts the construction of *partialNSCs* list in the form of NSCs. The subsets of the longest partial NSC are also stored in the list, to identify and store all the alternative execution paths effectively.

$$\forall s_y \in S_j, y \in \mathbb{N}, \forall n_z \in V, z \in \mathbb{N}, \{l, m, o, \ldots, t, p, q, r, \ldots, k\} \subset \mathbb{N}, \textbf{When } serviceIDs = \{p, q, r, \ldots, k\}, \tag{8}$$

$$partialNSCs = \{\ [\phi], [< s_p, n_l >], [< s_p, n_l >, < s_q, n_m >], [< s_p, n_l >, < s_q, n_m >, < s_r, n_o >],$$
$$\ldots, [< s_p, n_l >, < s_q, n_m >, < s_r, n_o >, \ldots, < s_k, n_t >]\ \}$$

The algorithm further checks the *partialNSCs* list during each traversal for complete NSCs (line 15). Since each element in the *partialNSCs* list is an array towards building the NSC when any of the arrays consists of services equal to the number of services

in the NSC, the algorithm considers the array to be complete. Thus, it speeds up the performance of finding the complete NSCs among the entries by just comparing the array lengths with the number of services composing NSC, instead of comparing the services themselves (line 16).

Évora computes the penalty values consisting of parameters such as T^{-1} and C for each chosen NSC, by retrieving the unit values (such as c_s and τ_s) from the edge VNF provider APIs for each of the VNFs. Since its goal is to satisfy the optimization of the MILP models, it stores the computed penalty values consisting of parameters as metadata for the NSC, using *persistPenaltyValues()* (line 17). Storing of penalty values as metadata against each NSC increases the performance of finding the best-fit NSCs for different user-defined penalty ratios (such as α, β, and γ) at each execution.

Finally, *pc* is added to the *matchingSubgraphsMultiMap* against the key, *nsc* (line 18). Thus, multiple matches of execution paths for each *nsc* are identified and stored, which can later be traversed to find the best fit for any user-defined policy for a given NSC. The caching of multiple options ensures quick reaction to the dynamic changes of the edge nodes, such as node downtime, service interruption, resource over-utilization in any given service deployment, and network congestion. *pc* is then removed from the *partialNSCs* list since the NSC is now complete and therefore no further service needs to be added to it (line 19). The algorithm thus finds all the potential execution paths until it finishes entirely traversing the nodes detected through the MOM messages (line 23). Consequently, the *initNSC* procedure traverses the node graph, to find all the complete matching NSC execution paths and stores them accordingly in the service graphs. By a complete search with all the nodes in the graph, and then choosing the NSC with the minimal penalty value, *Évora* ensures that the optimal NSC is chosen.

4.3 | Resilient and Adaptive Execution of the NSCs

Algorithm 3 presents the NSC execution procedure where the orchestrator starts execution of the user-defined NSC. As the core of *Évora* approach, this algorithm takes an NSC defined by the user (*nsc*), the relevant policies that need to be satisfied and optimized through the MILP models (*policy*), and a pointer to the **iTraverse()** function interface as the input parameters.

Algorithm 3 Executing an NSC at the Edge

```
1:  procedure EXECUTENSC(nsc, policy, iTraverse())
2:      nscGraph ← nscMap.get(<nsc, policy>)
3:      if (nscGraph = NULL) then                          ▷ One-time initialization per <nsc, policy>
4:          if (matchingSubgraphsMultiMap.get(nsc) = NULL) then
5:              initNSC(nsc, iTraverse())
6:          end if
7:          nscGraph ← matchingSubgraphsMultiMap.getBest(nsc, policy)   ▷ Resolve the MILP models based on the policy.
8:          nscMap.put(<nsc, policy>, nscGraph)            ▷ Replace the NULL value set in Algorithm 1 for the entry
9:      end if
10:     nextNodes ← nscGraph.getFirst()                    ▷ For parallel execution with multiple first VNFs, as elaborated in Figure 4
11:     for (node in nextNodes) do
12:         send(node, nscGraph)
13:     end for
14:     sendEvent(broker, <nsc, policy>)
15: end procedure
```

First, the *executeNSC* procedure initializes *matchingSubgraphsMultiMap* and *nscGraph* for the relevant NSC execution, if they were not initialized previously. *initNSC()* procedure (elaborated in Algorithm 2) is called to initialize the *matchingSubgraphsMultiMap* (line 5). Each NSC is defined along with the policies that it needs to abide by, such as minimal cost or minimal latency. The algorithm chooses the best fit among the matching execution paths from the multimap based on the user-defined policy, to resolve the MILP models (line 7). Since the penalty values for each NSC is stored as metadata against each NSC entry in Algorithm 2, we can solve the minimization/maximization problem by a simple integer substitution (of α, β, γ, ...) in $\Theta(n)$ time. Here n stands for the number of identified NSC execution paths for a user NSC.

nscGraph defines the chosen execution path or the chain of VNFs executing the NSC workflow. *nscGraph* is stored against the <*nsc, policy*> tuple in the *nscMap* (line 8). The first nodes to initialize the NSC workflow are identified from *nscGraph* (line 10). The VNF execution is triggered at the first nodes in the execution path (lines 11 - 13). As the NSC may consecutively invoke

two or more parallel service deployments as we elaborated in Figure 4 , the execution is generalized for distributed and parallel execution of NSC. An event is sent to the respective broker using *sendEvent()* (line 14), with information on the NSC and policy.

5 | PROTOTYPE IMPLEMENTATION

Évora proposes MOM protocols such as Advanced Message Queuing Protocol (AMQP)[36], MQTT[37], Simple Text Oriented Messaging Protocol (STOMP)[38], Java Message Service (JMS)[39], and Extensible Messaging and Presence Protocol (XMPP)[40] to be used in the edge nodes to publish their service status. We developed a prototype of *Évora* using Oracle Java 1.8.0 with OpenDaylight[41] as the core SDN Controller and ActiveMQ[42] broker. User devices consist of an orchestrator that has a message broker in the queue. Thus, the user can pervasively sense and detect the VNFs at the edge. We envision this as an edge version of IoT, or an Internet of Services at the edge, more realistic and practical with the problem space confined to the edge, than the entire Internet ecosystem.

Figure 6 presents an illustrative deployment of the edge nodes and the user device. It also demonstrates how the orchestrator of the user device visualizes the hypergraph of the edge node services in a *graph of complete graphs*. The edge nodes are detected through the MOM messages, by subscribing to the relevant notifications. The user chooses to find only the relevant VNF nodes at the edge. Thus, she can effectively control the size of the graphs based on her interests on various VNFs. The size of the *Évora* graph representation in each user device can be anywhere from a few KBs to several MBs. *Évora* represents each node by a complete graph. Each link to the node that connects it to adjacent nodes or the Internet is depicted by multiple links to each of the service in the nodes. The connected graphs representing each node consists of nodes that denote the services connected by stronger links than the inter-node links of the hypergraph as illustrated.

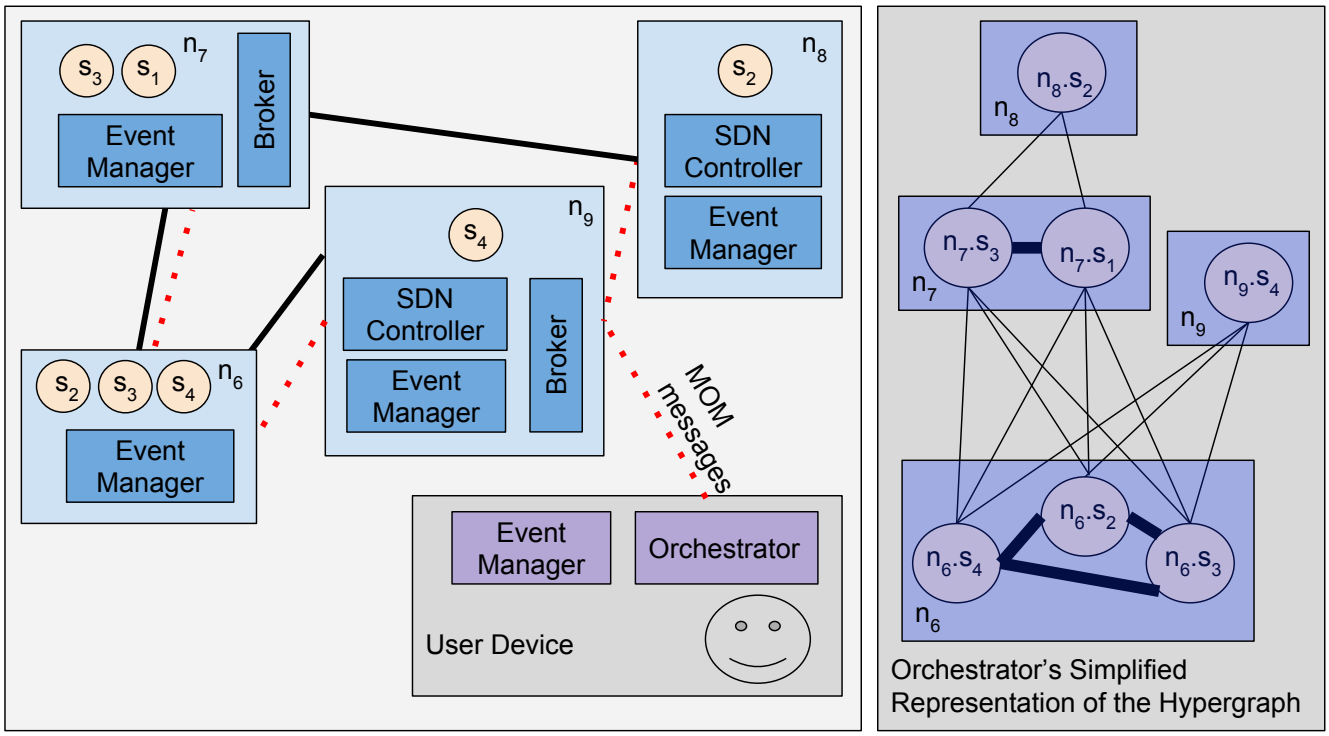


FIGURE 6 An *Évora* deployment: Edge Nodes and the User Device

Some nodes have a broker, while every edge node consists of an Event Manager. Event Manager is a thread that functions as an event publisher and event listener, to publish messages to the subscribers through the broker, and subscribe to notifications for relevant events. We developed the interaction between the Event Manager and the broker, through AMQP messages. *Évora*

SDN controller is an extended SDN controller with VNF orchestration capabilities. The controllers manage the node networks, with the ability to dynamically program the network. MOM messages do not necessarily require a static link, as they can go through over the Internet or an overlay network, and can be formed dynamically without a predefined hierarchy. The user device consists of an orchestrator where the user can define an NSC as a composition of various services. With an Event Manager, the user device also is subscribed to the broker instances to know the status of the edge VNF deployments.

There should be a secured connection between the nodes for the data transfer between the service executions in an NSC. Edge nodes of a single provider in a neighborhood are typically connected to form a private network to minimize latency, reduce expenditures and security risks caused by utilizing the public Internet for local inter-node traffic, and maximize throughput. Unlike a stand-alone VNF execution, NSC requires a workflow of multiple services. A VNF provider typically guarantees a protected connection for live migration of data between her VNF offerings. *Évora* ensures completeness of a user NSC execution through the availability of a connected graph of nodes consisting of all the services in the NSC.

In a typical edge user scenario (such as an edge data center used for network services), the user is subscribed to offerings from particular edge nodes and will have dedicated connections established between the edge nodes and the user on-premise deployments and servers to avoid latency. In a mobile environment or an execution environment composed of many light-weight edge nodes, the user needs to develop links dynamically through the Event Manager. As with the cloud platforms and web services, edge nodes should provide standardized APIs for the customers to interact with their service offerings, and support inter-node migration of data and execution. Unless a single edge VNF provider offers all the related network services, it is fair to expect that they provide relevant secured APIs for NSCs, to maintain a competitive business model. We anticipate the API-driven microservices era that we witness in the current industry will make this more widespread soon in 5G and MEC environments.

6 | EVALUATION

We assess *Évora* algorithms with microbenchmarks to evaluate its performance and ability to abide by the user policies.

6.1 | Problem Size and Scalability of *Évora*

Évora represents the edge services hypergraph as a graph of complete graphs, as elaborated in Section 3. The number of links in the node graph depends on the average degree of each edge node, in its connections with other edge nodes, and the total number of edge nodes considered in a given deployment of *Évora*. First, we evaluate the growth of the service graph representation against the increasing number of services in each node and the average number of links between the nodes.

Figure 7 illustrates the growth of the problem space of constructing NSCs at the edge. Just with 10 VNF per edge node, and nine links in the node graph, *Évora* service graph reaches 1000 links. We observe that the number of links in the service graph representation grows linearly with the number of links between the edge nodes, and exponentially with the average number of services per edge node. The growth of the alternative parts composing the NSC illustrates the scale of the resource allocation problem.

We further profiled the performance of *Évora* with microbenchmarks to evaluate its performance with scale. Table 2 lists how the orchestrator in the user device performs in various stages of its execution. The most time-consuming tasks are constructing the graphs initially and finding the potential VNFs for an NSC via graph matching for the first time. Constructing the edge and VNF graphs are performed at the initialization of the user device by the Algorithm 1. The time it takes is bound by the number of edge nodes and the number of brokers. The number of brokers can be at most as high as the number of edge nodes, in the worst case scenario for the performance of *Évora* orchestrator. In the best case, the number of brokers is much smaller in number compared to the number of edge nodes. Therefore the *Évora* orchestrator takes $\Theta(n^2)$ time to initialize the graphs. The n depends on the number of edge nodes that the user device identifies. The data is stored in indexed map structures. Therefore, the subsequent update messages received from the brokers take $\Theta(1)$ time to keep the graphs updated, regardless of the number of edge nodes and VNFs present. Therefore, with time, the overhead caused by the edge notifications in practice is a constant. In the worst case, if most of the edge nodes send update notifications at once, the update time will be $\Theta(log(n))$.

Once the graph is constructed, the orchestrator needs to find the potential VNFs for a given NSC, by executing Algorithm 2. The MILP models on maximizing or minimizing specific parameters are not considered in the initial graph matching to find and store all the matching pairs of VNFs. Hence, this step is performed regardless of the user policies. This graph matching takes

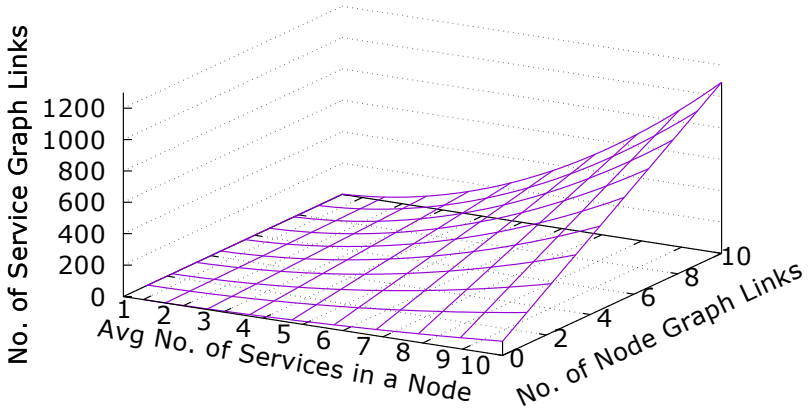


FIGURE 7 Representation of the service graph from the node graph

TABLE 2 Performance and Scalability of *Évora* Orchestrator Algorithms

Alg.	Description	Frequency	$\Theta(f(n))$
1	Construct the graphs	Once per user device	$\Theta(n^2)$
1	Updates to the graphs	Upon an update message from a broker	$\Theta(1)$
2	Finding potential VNFs for an NSC via graph matching	Once per user NSC definition	$\Theta(n^2)$
3	Choosing the best-fit VNFs by solving the MILP models	Once per <nsc, policy> pair	$\Theta(n)$
3	Executing an NSC abiding by a user policy	Once for each NSC execution	$\Theta(1)$

quadratic time, which scales depending on the number of VNFs in the graph. In practice, this is often executed just once, along with the initialization of the graphs. The updates are handled within a logarithmic time scale, similar to the graph initialization.

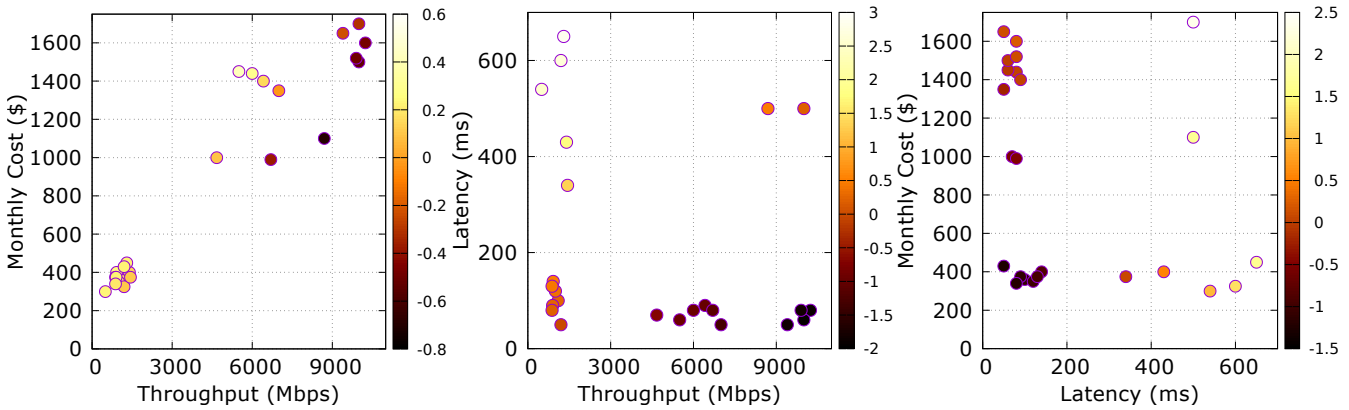
Finally, solving the MILP models for each $< nsc, policy >$ pair takes in all the cases $\Theta(n)$ time. Once the VNFs are chosen for a user NSC abiding by the policies, the subsequent overheads to the NSC execution from *Évora* orchestration are negligible constant time. Thus, by using its efficient map data structures to store the graphs, *Évora* minimizes the overheads on more frequent executions, while making the MILP optimization time linear. Furthermore, by storing the previously computed results such as all the matching VNFs, *Évora* handles the updates more efficiently, from constant to logarithmic times in the average and worst case scenarios. While the initialization process itself takes a few seconds (depending on the computational power and the number of edge devices) in the presence of thousands of edge nodes, this is one-time initialization process. Through its scalable architecture empowered with MOM, *Évora* allocates the user NSCs across the edge VNFs, despite the presence of 1000 edge nodes, along with a million service graph links. We measured and estimated the time overhead from *Évora* for the first execution of an NSC to be less than 0.3%, taking a few seconds when the NSC executions (such as browsing the Internet through a protected NSC with firewalls, parental control, and load balancers) are in the scales of minutes to hours. Furthermore, except for the first flow of a user NSC, all the subsequent flows do not incur an overhead from the *Évora* algorithms.

6.2 | Efficient VNF Allocation at the Edge

We evaluated the accuracy of *Évora* (to assess its effectiveness in the sense of how accurately its decisions reflect the user's intent) in VNF allocation at the edge, based on user-defined penalty functions consisting of one to three attributes: from latency (ms), throughput (Mbps), and monthly cost ($). When handling two or more attributes, the penalty function gives weight (through user-defined policies) to the normalized values of each of the metrics (e.g., throughput and monthly cost) across each one's value range (e.g., from 0 to 12000 Mbps; from 0$ to 1800$, respectively). In the following figures, the more accurate (and thus better performant) *Évora* is regarding the user's intent, the lower the penalty function is, and the darker the circles are (so, lower penalty and darker circles are better). These darker circles should thus occupy regions of the graphs where the (possibly combined) metrics of interest to the user exhibit better values.

Two Variable Attributes in the User Policies

Figure 8 depicts the performance of *Évora*, calculating a composite penalty value from two variable attributes (each with an equal weight of 1.0) and minimizing the penalty value for a resource allocation adhering to the user-defined policy. The Figures show darker circles where the penalty function is lower (and therefore indicating closeness to what it captures as the user's intent).



(a) Max. Throughput and Min. Cost **(b)** Max. Throughput and Min. Latency **(c)** Min. Latency and Min. Cost

FIGURE 8 *Évora* policies with two attributes of equal weight.

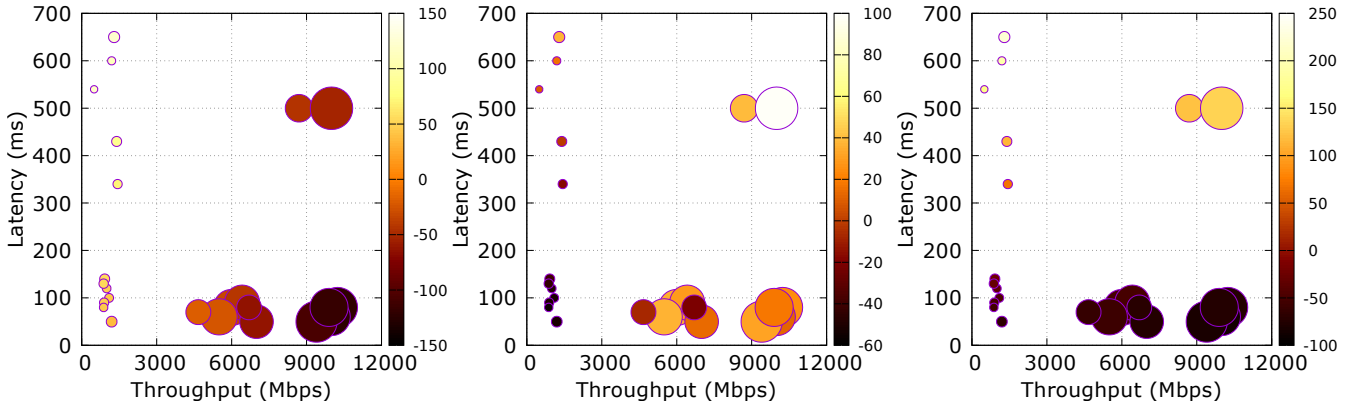
In Figure 8 a we can see that most darker circles are closer to the highest values of throughput (one aspect of the user's policy) but in the top-right corner (higher cost). Nonetheless, we note that there are also darker circles (two) on areas with lower cost ($1000, i.e., with significant savings from $1600) than other circles that offer lower bandwidth with actually higher cost (i.e., lighter circles on the $1400 line). This observation indicates that our penalty function can award some of the lowest values (and therefore identify those as of higher interest to the user) to the combination of 9000 Mbps and around $1000 cost. This seems indeed the best combination for the user (remember she gives equal weight to monetary cost and throughput).[2] Figure 8 b shows that the darker cycles are more present in the lower-right area of the chart. Thus, the penalty function can identify how to maximize throughput (well over 9000 Mbps) while minimizing latency (with equal weights) with great effectiveness (once again, any of values with the lowest distance to the lower-right corner would be optimal). Similarly, in Figure 8 c, the darker circles are predominantly in the lower-left region, showing that the penalty function can capture solutions that minimize cost (lower than $400), with very reduced latency (as good as with solutions costing $1000 and downwards).

Three Variable Attributes in the User Policies

We then evaluated the VNF allocation considering all three attributes: latency, throughput, and monthly cost. Figure 9 illustrates the performance of *Évora* giving prominence to one of the three attributes while considering the others as a secondary preference, in a three-dimensional space. The attribute with the highest prominence is given the weight of 1.0, while the other two are given the weight 0.3. The monthly cost of the NSC is depicted by the radius of the circles, while the throughput and latency are illustrated by the x and y-axes respectively. Similarly, Figure 10 illustrates the performance of *Évora* giving prominence to two or all of the three attributes equally.

In Figure 9 a we can see that most darker circles are closer to the highest values of throughput (higher than 9000 Mbps). We also note that with equal throughput, the darkest circles are at the bottom-right, indicating lower latency in addition to the high throughput. As the cheaper services offered lower throughput, they are not chosen by *Évora*. Figure 9 b shows that the dark cycles are those with the smallest radius, indicating how the cost is minimized. Moreover, still, the darkest circles are found at the bottom of the plot, indicating the minimal latency. Since higher throughput services typically entailed a higher cost, it was a trade-off that lower throughput services were chosen in favor of a lower cost. Similarly, in Figure 9 c, the darker circles are

[2] It is an extra step, not hard under these circumstances, to identify this more adequate value (even though, for completeness, this could entail, e.g., exploring the Pareto frontier of the optimal solutions, that we do not consider here and leave for future work).



(a) Max. Throughput **(b)** Min. Cost **(c)** Min. Latency

FIGURE 9 *Évora* policies considering three attributes with prominence to one of the three attributes. The radius of the circles represents the cost.

predominantly in the bottom region, showing the penalty function can capture solutions that minimize latency (lower than 200 ms). Among those, the circles with smallest radius (cheapest services) and those in the bottom-right (higher throughput) remain the darkest, indicating the efficacy of *Évora* in adhering to the user policies.

Figure 10 a, Figure 10 b, and Figure 10 c give prominence to two of the three attributes equally (1.0), while still considering the third attribute (with a weight of 0.3). Figure 10 d gives equal prominence to all three attributes (1.0). Figure 10 a displays the darkest circles at the highest values of throughput (higher than 9000 Mbps), while also having darker circles with smaller radius indicating the cheaper services (as found in a cheaper service offering around 6000 Mbps). Since latency was not given prominence, it can be noted that a smaller circle (lower cost) with high throughput (9000 Mbps) still has a dark complexion though it had a high latency (500 ms). Figure 10 b shows the darkest circles in the bottom-right corner, showing high throughput (higher than 9000 Mbps) and lower latency (less than 100 ms), albeit with higher prices as the cost was not given prominence. Figure 10 c contains the circles with the smallest radius on the bottom as the darkest, indicating the success in identifying choices of minimal cost and minimal latency (less than 150 ms). However, as a trade-off, here the services with lower throughput were chosen. Figure 10 d attempts to give equal weight to all the properties. In doing so, it manages to maximize the throughput (typically above 9000 Mbps) and minimize the latency (typically below 100 ms). It also has dark circles with low radius, indicating the preference for cheaper services.

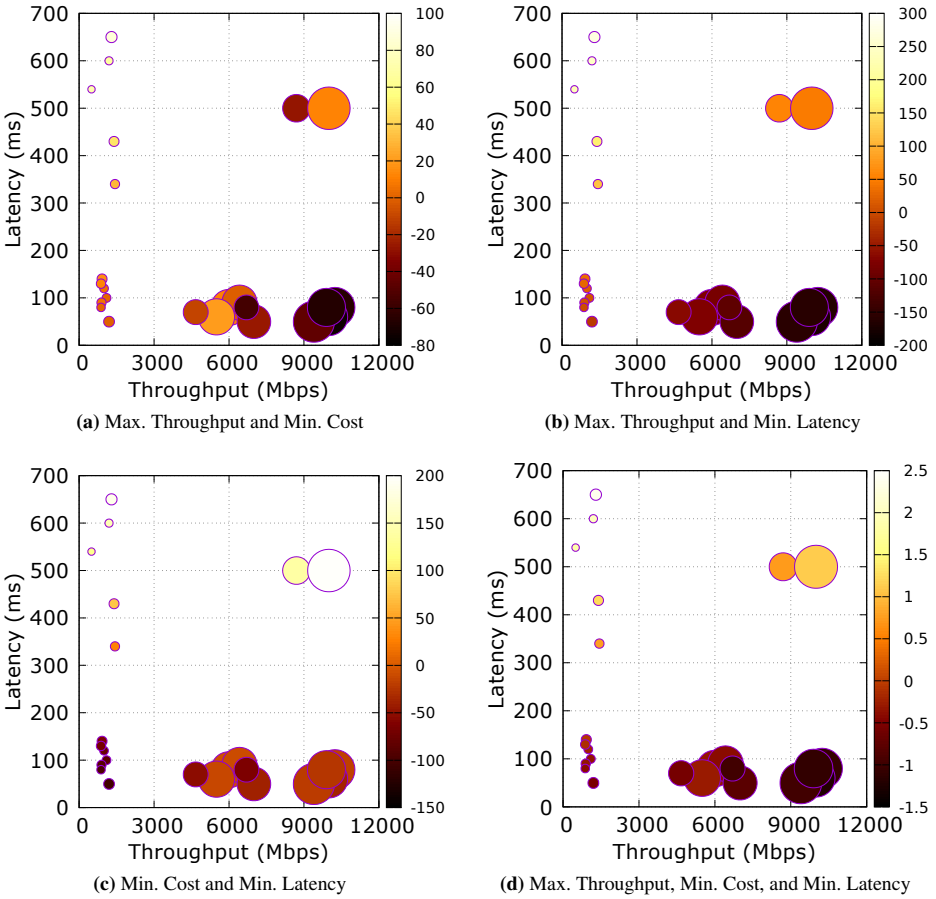
Our evaluations highlighted how *Évora* supports user-driven policies effectively in selecting the NSCs, considering one to three attributes with user-specified policies (including various attributes, normalized with weights). Considering more attributes as n-dimensional environments, *Évora* can handle complex user policies with more than three attributes defining the penalty value that needs to be minimized.

7 | RELATED WORK

In this section, we will look into the state of the art and related work to *Évora* on, i) VNFs in edge environments, ii) VNF placement, resource allocation, and network softwarization for NSC, and iii) Orchestration of network services.

7.1 | VNFs in Mobile and Edge Environments

The recent surge in network softwarization has enabled a fine-grained automated control of data center and cloud networks. Light-weight container-based frameworks have been proposed for service provisioning with minimal service migration time in the MEC environments[28]. Several solutions have been proposed for context-aware execution of mobile applications at the edge, with focus on bandwidth, latency, and jitter[43]. However, the current solutions fail short in addressing optimality due to the scale and variety of devices and users in the MEC environments. By leveraging its SDN-based architecture and algorithms leveraging the edge environments, *Évora* offers both optimality and scalability to the VNFs in the MEC environment.



(a) Max. Throughput and Min. Cost

(b) Max. Throughput and Min. Latency

(c) Min. Cost and Min. Latency

(d) Max. Throughput, Min. Cost, and Min. Latency

FIGURE 10 *Évora* policies considering three attributes with prominence to two or all of the three attributes. The radius of the circles represents the cost.

NetFATE[44] offers a VNF-as-a-Service (VNFaaS), virtualizing network services of both Customer Premises Equipment (CPE) devices and Provider Edge (PE) nodes. Though NetFATE can be extended to support NSCs, it focuses on a network provider offering the VNFs, rather than facilitating user-defined NSC over third-party edge VNFs. *Évora* supports user NSCs across the edge VNFaaS and has a broader problem space. Leveraging network softwarization, system administrators may configure the networks to adhere to a specific set of policies, such as energy and carbon efficiency[45]. The system and user goals can be complementary or even contradictory to one another. For example, one user's requirement for latency-aware execution may lead to a higher energy cost for the network. Thus, the research proposes VNF placement in distributed edge data centers considering various heuristics[46]. These policies can easily be incorporated into the MILP models of *Évora*, though *Évora* focuses on NSCs from an independent end-user perspective.

Évora identifies that controlling MEC and 5G networks requires a dynamic collaboration of controllers, with protected access to each other. The research proposes fair resource allocation algorithms for Lambda architecture to support a seamless deployment of services as functions across various infrastructures[47]. Mobile Service Sharing Community (MSSC)[48] addresses the service composition in dynamic mobile environments with both the mobile web service provider and consumer being non-stationary. SDS frameworks have been proposed for efficient execution of MEC applications[49]. CHIEF[50] designs a community network with several edge nodes with a federated deployment of SDN controllers. While these IoT approaches facilitate VNFs in dynamic MEC environments, they are not optimized to chain the VNFs in a bandwidth-efficient manner. *Évora* addresses these shortcomings, thus accommodating the latency-sensitive nature of VNFs.

7.2 | VNF Placement for NSC

Network softwarization brings down the time to develop an NSC from the scale of hours or a few days down to the range of seconds or minutes[51]. SDN control plane for data centers has been designed with hypergraph models[52]. NSCs have been optimally provisioned across the selected VNF nodes using ILP[53] and MILP models[54]. Using a Column Generation (CG) decomposition[55] along with an ILP, previous research minimizes the bandwidth consumption in the software-defined data center networks of NSCs[53]. This research has been extended concerning geographically distributed cloud environments to have minimal inter-cloud traffic and response time. An affinity-based approach (ABA) minimizes the total delays and resource cost in NSCs, solving more complex problems faster than ILP in the distributed cloud environment[56]. These works share the approach with *Évora*, though they do not extend to the edge environments which are more diverse. The previous research[31] also looks at the optimization problem from the perspective of a VNF provider, while *Évora* looks at it from the user perspective to identify the best options for the user from multiple edge VNF providers.

Virtual Network Embedding (VNE)[57] is a core concept in multi-tenant virtual networks, as the network resources are allocated to various tenants that share the network. ViTeNA[58] proposes algorithms for VNE in multi-tenant data centers. ESCAPE is a scalable architecture for NSC orchestration, which leverages Mininet network emulator[59], ClickOS[60] virtualized software middlebox platform, NetConf configuration protocol[61], and POX SDN controller[62], to support VNF implementation, traffic steering, and VNE[63]. OpenDaylight extends its controller to offer SFC as an incubation project. UNIFY programmatic framework performs NSCs in a data center environment based on OpenStack[64] and OpenDaylight through joint virtualization and control[65]. It presents an NSC control plane to integrate various VNF execution environments, supporting dynamic VNF creation and orchestration in cloud networks[66]. However, NSC at the edge has more dimensions than data centers and clouds, as edge environments consist of several nodes of different scales and numerous services. *Évora* presents an adaptive approach for VNF selection at the edge for user NSCs.

Mayan[67] is a service composition framework that exploits SDN and MOM to choose the best service deployments among the available alternative deployments for a context-aware composition of web services in a wide area network. NSC is a specific case of service compositions, as service compositions chain services with one's output as the input of another. However, *Évora* is the first to extend this approach for a resilient and adaptive execution of NSCs at the edge. A hybrid network with dedicated physical/hardware network functions along with VNFs can offer elasticity to the network functions and support bursts in demand. Efficient placement of such a hybrid deployment can improve the throughput of NSCs, as shown through ILP models in previous research[68]. *Évora* can extend these hybrid networks to the edge and enhance their efficiency.

7.3 | Service Orchestration

Service awareness and autonomic computing, including self-healing, self-protection, and self-optimization, have long been proposed to evolve service layer cost-efficiently for the telecommunication industry[69]. These early research efforts were driven significantly by research and did not materialize in the mainstream telecommunication enterprise. Currently, the industrial implementation efforts are catching up on the VNF orchestration front with the move towards the 5G networks. ONAP, Open Network Automation Platform[70] is a Linux Foundation project to manage, control, and orchestrate the network services end-to-end. Lifecycle Service Orchestration (LSO) encompasses the network softwarization advances of SDN and NFV for end-to-end management, orchestration, and control of network services[71]. Efforts on LSO and TOSCA (OASIS Topology and Orchestration Specification for Cloud Applications)[72] are working towards making complete standardization and end-to-end workflow orchestration a reality. Cloudify[73] is a model-driven microservices-based cloud orchestration framework with pluggable NFV components. Cloudify is interoperable with other VNF providers through its TOSCA-compliance.

MEF (formerly known as Metro Ethernet Forum) operates Third Network on Carrier Ethernet 2.0[74] with the vision of offering a comprehensive framework for LSO. Ciena Blue Planet[75] provides a microservices architecture with orchestration capabilities, offering a vendor-agnostic platform for LSO. T-NOVA[76] is a research effort on end-to-end management and orchestration of network services composed by VNF-as-a-Service. Cisco Tail-f[77] provides network orchestration with support towards achieving LSO. MEF 3rd network and ONAP collaborate to deliver orchestrated services over automated networks consisting of multiple providers and technology domains. Cross-layer management optimizations such as LSO facilitate composing and migrating NSCs seamlessly across platforms and vendors, towards end-to-end automation and management of network services. LSO focuses on complementing SDN and NFV to automate the network services with open APIs across multiple provider networks and technology domains. *Évora* expands the existing research on network softwarization and LSO into edge computing environments which are more dynamic than the traditional cloud and data centers. MEF 3rd network and Open Network Automation

Platform (ONAP)[70] collaborate for orchestrated services over automated networks, with a goal of offering service orchestration across multiple providers and technology domains. While these projects aim to provide a seamless inter-cloud service orchestration, their adoption to edge environments is still yet to be realized.

OSM is an open source implementation of MANO (NFV management and orchestration), to coordinate and manage the resources in a cloud data center[78]. MANO has been extended with a network slicing support, for the execution of VNFs in the LTE networks[79]. These research efforts are promising. However, due to the heterogeneous and multi-vendor nature of the edge, a messaging-based approach without a static control hierarchy (such as the one followed by *Évora*), will better fit NSC at the edge environments. There have been research and industrial efforts on service placements at the edge, and software-defined approaches to facilitate service compositions. However, *Évora* is the first to propose an extended software-defined approach with SDN and MOM for optimal NSCs at the edge.

8 | CONCLUSION

Emerging trends in telecommunications show that network softwarization and MEC will continue to influence and drive the innovation in 5G. VNFs are placed at the edge to support the overwhelming demand for latency-aware service execution. However, resource allocation at the edge for NSCs is an NP-hard problem as various user policies must be satisfied to assure QoE while choosing the VNFs.

In this paper, we presented MILP models and a graph-based architecture to compose NSCs by leveraging the third-party edge VNFs from multiple providers. We designed *Évora* extending SDN and MOM for an adaptive execution of NSCs. *Évora*, with its software-defined approach, spawns instances of VNFs in each edge node that offers the particular services and thus helps to compose NSCs that are policy and latency-aware. We evaluated the *Évora* models and algorithms for NSCs with various user-defined policies. Our evaluations highlight how *Évora* provides resilience and agility to the NSCs in the MEC environments through its orchestration of multiple execution paths spanning the edge nodes.

As a future work, we envision *Évora* supporting NSCs in hybrid networks consisting of both physical and virtual network functions. The VNFs are spawned when there is a spike in demand for the hardware network functions. This approach offers both the performance of the hardware network functions as well as the scalability and configurability of the VNFs. Thus, *Évora* could be extended with backward-compatibility for legacy physical network functions that are agnostic to network virtualization.

References

1. Kirilenko Andrei, Kyle Albert S, Samadi Mehrdad, Tuzun Tugkan. The flash crash: The impact of high frequency trading on an electronic market. *Available at SSRN.* 2011;1686004.

2. Claypool Mark, Claypool Kajal. Latency can kill: precision and deadline in online games. In: Proceedings of the first annual ACM SIGMM conference on Multimedia systems:215–222ACM; 2010.

3. Anvari Mehran, Broderick Tim, Stein Harvey, et al. The impact of latency on surgical precision and task completion during robotic-assisted remote telepresence surgery. *Computer Aided Surgery.* 2005;10(2): 93–99 .

4. AWS . *AWS Global Infrastructure.* Available at https://aws.amazon.com/about-aws/global-infrastructure/; 2017.

5. Khosravi Atefeh, Garg Saurabh Kumar, Buyya Rajkumar. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In: European Conference on Parallel Processing: 317–328 Springer; 2013.

6. Ahmed Ejaz, Rehmani Mubashir Husain. *Mobile edge computing: opportunities, solutions, and challenges.* 2017.

7. Bonomi Flavio, Milito Rodolfo, Zhu Jiang, Addepalli Sateesh. Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing: 13–16 ACM; 2012.

8. 365DataCenters . *365 - U.S. Data Center Locations.* Available at http://www.365datacenters.com/data-centers/; 2018.

9. vXchnge . *vXchnge Markets.* Available at http://www.vxchnge.com/markets/; 2018.

10. EdgeConneX . *Edge Data Center Locations.* Available at http://www.edgeconnex.com/edge-data-center-locations/; 2018.

11. De Brito MS, Hoque S, Steinke R, Willner A, Magedanz T. Application of the Fog computing paradigm to Smart Factories and cyber-physical systems. *Transactions on Emerging Telecommunications Technologies.* 2018;29(4):e3184.

12. Zhang Ben, Mor Nitesh, Kolb John, et al. The Cloud is Not Enough: Saving IoT from the Cloud.. In: HotCloud; 2015.

13. Shi Weisong, Cao Jie, Zhang Quan, Li Youhuizi, Xu Lanyu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal.* 2016;3(5): 637–646 .

14. Al-Hammouri Ahmad T, Al-Ali Zaid, Al-Duwairi Basheer. ReCAP: A distributed CAPTCHA service at the edge of the network to handle server overload. *Transactions on Emerging Telecommunications Technologies.* 2018;29(4):e3187.

15. 365DataCenters . *365 - Managed Services.* Available at https://www.365datacenters.com/services/managed-services/; 2018.

16. Wichtlhuber Matthias, Reinecke Robert, Hausheer David. An SDN-based CDN/ISP collaboration architecture for managing high-volume flows. *IEEE Transactions on Network and Service Management.* 2015;12(1):48–60.

17. Manzalini Antonio, Saracco Roberto. Software Networks at the Edge: a shift of paradigm. In: Future Networks and Services (SDN4FNS), 2013 IEEE SDN for: 1–6 IEEE; 2013.

18. Halpernj Pignataroc. RFC7665, Service Function Chaining (SFC) architecture. *IETF Datatracker.* 2015;.

19. Sayadi Bessem, Gramaglia Marco, Friderikos Vasilis, et al. SDN for 5G Mobile Networks: NORMA perspective. In: International Conference on Cognitive Radio Oriented Wireless Networks: 741–753 Springer; 2016.

20. Zhang Ying. *Network Function Virtualization: Concepts and Applicability in 5G Networks*. John Wiley & Sons; 2017.

21. Rossenhövel Carsten, Price Chris, Váncsa Ildikó. *The Interoperability Challenge in Telecom and NFV Environments.* : The Linux Foundation; 2017. Available at https://events.linuxfoundation.org/sites/events/files/slides/The_Interoperability_Challenge_in_Telecom_and_NFV_Environments.pdf.

22. Yaseen Qussai, Albalas Firas, Jararwah Yaser, Al-Ayyoub Mahmoud. Leveraging fog computing and software defined systems for selective forwarding attacks detection in mobile wireless sensor networks. *Transactions on Emerging Telecommunications Technologies.* 2018;29(4):e3183.

23. RAD . *D-NFV Alliance.* Available at https://www.rad.com/content/d-nfv-alliance; 2017.

24. Chaudhary Rajat, Kumar Neeraj, Zeadally Sherali. Network Service Chaining in Fog and Cloud Computing for the 5G Environment: Data Management and Security Challenges. *IEEE Communications Magazine.* 2017;55(11): 114–122 .

25. Curry Edward. Message-oriented middleware. *Middleware for communications.* 2004;:1–28.

26. García-Pérez César Augusto, Merino Pedro. Experimental evaluation of fog computing techniques to reduce latency in LTE networks. *Transactions on Emerging Telecommunications Technologies.* 2018;29(4):e3201.

27. Cziva Richard, Pezaros Dimitrios P. On the Latency Benefits of Edge NFV. In: Architectures for Networking and Communications Systems (ANCS), 2017 ACM/IEEE Symposium on: 105–106 IEEE; 2017.

28. Farris Ivan, Taleb T, Flinck H, Iera A. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies.* 2017;.

29. Lim Sharon, Ha J, Kim H, Kim Y, Yang S. A SDN-oriented DDoS blocking scheme for botnet-based attacks. In: Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on: 63–68 IEEE; 2014.

30. Yan Gongjun, Rawat Danda B, Bista Bhed Bahadur. Provisioning vehicular ad hoc networks with quality of service. *International Journal of Space-Based and Situated Computing.* 2012;2(2): 104–111 .

31. Bari Md Faizul, Chowdhury Shihabur Rahman, Ahmed Reaz, Boutaba Raouf. On orchestrating virtual network functions. In: Network and Service Management (CNSM), 2015 11th International Conference on: 50–56 IEEE; 2015.

32. Batalla Jordi Mongay, Mastorakis George, Mavromoustakis Constandinos X, Dobre Ciprian, Chilamkurti Naveen, Schaeckeler Stefan. Network Services Chaining in the 5G Vision. *IEEE Communications Magazine.* 2017;55(11): 112–113 .

33. McKeown Nick, Anderson Tom, Balakrishnan Hari, et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review.* 2008;38(2): 69–74 .

34. Jain Sushant, Kumar Alok, Mandal Subhasree, et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review.* 2013;43(4): 3–14 .

35. Parekh Ojas. Iterative packing for demand and hypergraph matching. In: International Conference on Integer Programming and Combinatorial Optimization: 349–361 Springer; 2011.

36. Vinoski Steve. Advanced message queuing protocol. *IEEE Internet Computing.* 2006;10(6).

37. Banks Andrew, Gupta Rahul. MQTT Version 3.1.1. *OASIS standard.* 2014;29.

38. Magnoni L. Modern messaging for distributed sytems. *Journal of Physics: Conference Series.* 2015;608(1): 12–38 .

39. Hapner Mark, Burridge Rich, Sharma Rahul, Fialli Joseph, Stout Kate. Java message service. *Sun Microsystems Inc., Santa Clara, CA*. 2002;:9.

40. Saint-Andre Peter, Smith Kevin, Tronçon Remko. *XMPP: the definitive guide*. O'Reilly Media, Inc.; 2009.

41. Medved Jan, Varga Robert, Tkacik Anton, Gray Ken. Opendaylight: Towards a model-driven sdn controller architecture. In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a: 1–6 IEEE; 2014.

42. Snyder Bruce, Bosnanac Dejan, Davies Rob. *ActiveMQ in action*. Manning Greenwich Conn.; 2011.

43. Orsini Gabriel, Bade Dirk, Lamersdorf Winfried. CloudAware: Empowering context-aware self-adaptation for mobile applications. *Transactions on Emerging Telecommunications Technologies.* 2018;29(4):e3210.

44. Lombardo Alfio, Manzalini Antonio, Schembra Giovanni, Faraci Giuseppe, Rametta Corrado, Riccobene Vincenzo. An open framework to enable NetFATE (network functions at the edge). In: Network Softwarization (NetSoft), 2015 1st IEEE Conference on: 1–6 IEEE; 2015.

45. Marotta Antonio, Zola Enrica, D'Andreagiovanni Fabio, Kassler Andreas. A fast robust optimization-based heuristic for the deployment of green virtual network functions. *Journal of Network and Computer Applications.* 2017;95:42–53.

46. Lange Stanislav, Grigorjew Alexej, Zinner Thomas, Tran-Gia Phuoc, Jarschel Michael. A Multi-objective Heuristic for the Optimization of Virtual Network Function Chain Placement. In: Teletraffic Congress (ITC 29), 2017 29th International, vol. 1: : 152–160 IEEE; 2017.

47. HoseinyFarahabady MohammadReza, Taheri Javid, Tari Zahir, Zomaya Albert Y. A Dynamic Resource Controller for a Lambda Architecture. In: Parallel Processing (ICPP), 2017 46th International Conference on:332–341IEEE; 2017.

48. Deng Shuiguang, Huang Longtao, Taheri Javid, Yin Jianwei, Zhou MengChu, Zomaya Albert Y. Mobility-aware service composition in mobile communities. *IEEE Transactions on Systems, Man, and Cybernetics: Systems.* 2017;47(3):555–568.

49. Jararweh Yaser, Alsmirat Mohammad, Al-Ayyoub Mahmoud, et al. *Software-defined system support for enabling ubiquitous mobile edge computing.* 2017.

50. Kathiravelu Pradeeban, Veiga Luís. Chief: Controller farm for clouds of software-defined community networks. In: Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on: 1–6 IEEE; 2016.

51. Association 5G-PPP, others . Contractual Arrangement: Setting up a Public-Private Partnership in the Area of Advance 5G Network Infrastructure for the Future Internet between the European Union and the 5G Infrastructure Association. *Dec.* 2013;7(20):3.

52. Xia Yiting, Ng TS. A cross-layer sdn control plane for optical multicast-featured datacenters. In: Proceedings of the third workshop on Hot topics in software defined networking: 229–230 ACM; 2014.

53. Huin Nicolas, Jaumard Brigitte, Giroire Frédéric. Optimization of network service chain provisioning. In: IEEE International Conference on Communications 2017; 2017.

54. Trivisonno Riccardo, Vaishnavi Ishan, Guerzoni Riccardo, et al. Virtual links mapping in future sdn-enabled networks. In: Future Networks and Services (SDN4FNS), 2013 IEEE SDN for: 1–5 IEEE; 2013.

55. Desaulniers Guy, Desrosiers Jacques, Solomon Marius M. *Column generation*. Springer Science & Business Media; 2006.

56. Bhamare Deval, Samaka Mohammed, Erbad Aiman, Jain Raj, Gupta Lav, Chan H Anthony. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications.* 2017;102: 1–16 .

57. Fischer Andreas, Botero Juan Felipe, Beck Michael Till, De Meer Hermann, Hesselbach Xavier. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials.* 2013;15(4): 1888–1906 .

58. Caixinha Daniel, Kathiravelu Pradeeban, Veiga Luís. ViTeNA: An SDN-based virtual network embedding algorithm for multi-tenant data centers. In: Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on: 140–147 IEEE; 2016.

59. De Oliveira Rogério Leão Santos, Shinoda Ailton Akira, Schweitzer Christiane Marie, Prete Ligia Rodrigues. Using mininet for emulation and prototyping software-defined networks. In: Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on: 1–6 IEEE; 2014.

60. Martins Joao, Ahmed Mohamed, Raiciu Costin, et al. ClickOS and the art of network function virtualization. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation: 459–473 USENIX Association; 2014.

61. Enns Rob. NETCONF configuration protocol. 2006;.

62. Kaur Sukhveer, Singh Japinder, Ghumman Navtej Singh. Network programmability using POX controller. In: ; 2014.

63. Sahhaf Sahel, Tavernier Wouter, Czentye János, et al. Scalable architecture for service function chain orchestration. In: Software Defined Networks (EWSDN), 2015 Fourth European Workshop on: 19–24 IEEE; 2015.

64. Sefraoui Omar, Aissaoui Mohammed, Eleuldj Mohsine. OpenStack: toward an open-source solution for cloud computing. *International Journal of Computer Applications.* 2012;55(3).

65. Elek Janos, Jocha David, Szabo Robert. Network Function Chaining in DCs: The Unified Recurring Control Approach. In: Software Defined Networks (EWSDN), 2015 Fourth European Workshop on: 13–18 IEEE; 2015.

66. Sonkoly Balázs, Czentye János, Szabo Robert, et al. Multi-domain service orchestration over networks and clouds: A unified approach. *ACM SIGCOMM Computer Communication Review.* 2015;45(4): 377–378 .

67. Kathiravelu Pradeeban, Grbac Tihana Galinac, Veiga Luís. Building blocks of mayan: Componentizing the escience workflows through software-defined service composition. In: Web Services (ICWS), 2016 IEEE International Conference on:372–379IEEE; 2016.

68. Moens Hendrik, De Turck Filip. VNF-P: A model for efficient placement of virtualized network functions. In: Network and Service Management (CNSM), 2014 10th International Conference on: 418–423 IEEE; 2014.

69. Manzalini Antonio, Zambonelli Franco. Towards autonomic and situation-aware communication services: the cascadas vision. In: Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. DIS 2006. IEEE Workshop on: 383–388 IEEE; 2006.

70. ONAP . *ONAP Open Network Automation Platform.* Available at https://www.onap.org/; 2017.

71. MEF . *Lifecycle Service Orchestration | Third Network.* Available at https://www.mef.net/third-network/lifecycle-service-orchestration; 2017.

72. Lipton Paul. *OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC.* : OASIS; 2017. Available at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.

73. Cloudify . *Cloudify: Cloud & NFV Orchestration Based on TOSCA.* Available at http://cloudify.co; 2017.

74. MEF . *MEF, Metro Ethernet Forum.* Available at https://www.mef.net/; 2017.

75. Ciena . *Blue Planet - transformational software solutions from Ciena.* Available at http://www.blueplanet.com/technology; 2017.

76. Carapinha Jorge, Di Girolamo Marco, Monteleone Giuseppe, Ramos Aurora, Xilouris George. VNFaaS with End-to-End Full Service Orchestration. In: Software-Defined Networks (EWSDN), 2016 Fifth European Workshop on: 57–58; IEEE; 2016.

77. Boyd Robb. *Network Service Orchestration enabled by Tail-f.* Available at https://blogs.cisco.com/cin/tail-f; 2015.

78. OSM . *Open Source MANO.* Available at https://osm.etsi.org/; 2017.

79. Katsalis Kostas, Nikaein Navid, Edmonds Andy. Multi-Domain Orchestration for NFV: Challenges and Research Directions. In: Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), International Conference on: 189–195 IEEE; 2016.

How to cite this article: P. Kathiravelu, P. Van Roy, and L. Veiga (2018), Composing Network Service Chains at the Edge: A Resilient and Adaptive Software-Defined Approach, *Transactions on Emerging Telecommunications Technologies (ETT)*, *2018*.