ANI: Abstracted Network Inventory for Streamlined Service Placement in Distributed Clouds

Danny Alex Lachos Perez
University of Campinas
Sao Paulo, Brazil
dlachosp@dca.fee.unicamp.br

Christian Esteve Rothenberg
University of Campinas
Sao Paulo, Brazil
chesteve@dca.fee.unicamp.br

Mateus Santos
Ericsson Research
Sao Paulo, Brazil
mateus.santos@ericsson.com

Pedro Henrique Gomes
Ericsson Research
Sao Paulo, Brazil
pedro.henrique.gomes@ericsson.com

***Abstract*—Scenarios for distributed cloud with multiple edge clouds and centralized data centers are being investigated as the computing and networking underpinnings of next-generation network services such as augmented reality, self-driving vehicles, drones, and more. In such distributed environments, service providers will typically face tens, hundreds, or thousands of compute location candidates (edge, regional, and central) where network service components can be placed. To take optimized placement decisions of network services and execute the management workflows, orchestration systems require up-to-date and accurate resource availability representation, in the form of a network inventory that can be immense in distributed cloud scenarios. As a result, the service management and placement problems may become not tractable. In this work, we propose the Abstracted Network Inventory (ANI) component to generate service-optimized network views over the same network inventory. ANI implements a novel abstraction method where network service requirements are used as an input to generate an optimized abstract network inventory representation, called Logical Network Inventory (LNI). We also provide a formal definition of the network model and problem statement along with the development of three algorithms to efficiently build an LNI. Results show the potential benefits of using an LNI to streamline service management and placement: (i) the relationship between compute nodes and links (*i.e.*, density) in an LNI is reduced between 1.8-2.7x compared to a full network inventory topology; and (ii) up to 50% of time can be saved for service placement after abstracting around 20% of the compute nodes.**

***Index Terms*—Network Inventory, Distributed cloud, Central Cloud, Regional Cloud, Edge Cloud**

I. Introduction

Emerging use cases like virtual and augmented reality, autonomous vehicles, smart cities, and drones call for transforming the way telecommunications operators deploy new network services, shifting from a manual and long process to a more flexible and programmable way [5, 9]. In this context, cloud computing [14, 21], Software Defined Networking (SDN) [11, 13], and Network Function Virtualization (NFV) [16, 18] arise as technological pillars to achieve the necessary flexibility and programmability during the provision of such network services. By softwarizing a network service, Network Functions (NFs) are separated from the hardware and offered through virtualized services that can be instantiated on data centers (as any other cloud applications) with the adequate connectivity.

978-1-7281-5684-2/20/$31.00

A distributed cloud is a cloud execution environment for NFs or applications that is distributed across multiple cloud sites (edge, regional, and central), with the required connectivity (networking) between them [6]. Distributed cloud deployment models keep latency-sensitive applications closer to the edges of the network (close to users), and move non-real-time applications to centralized data centers [2].

In such distributed cloud environments with edge and more centralized computing facilities, multiple cloud sites become candidate hosting targets for NFs and applications. Cloud sites are typically geographically distributed and interconnected through a Wide Area Network (WAN). Figure 1 shows interconnected edge cloud sites in which a centralized orchestrator, or Central Orchestrator (CO), can establish communication with Local Orchestrators (LOs) placed at individual edge cloud sites. An LO is also part of an edge cloud site so that some orchestration components can be deployed locally in a data center without always accessing the WAN. Eventually, regional cloud sites could also be deployed between central and edge cloud sites. Examples of open source projects that consider local and central orchestrators include Akraino[1] and ONAP[2].

To take optimized placement decisions, COs or LOs need to maintain an inventory of the network providing a real-time representation of the available resources in the network infrastructure along with their relationships. The size of a network inventory can become very large in distributed cloud scenarios because a typical service provider will have hundreds or thousands of edge cloud deployments. As a result, COs and LOs face scalability challenges when processing large amounts of data to decide where to instantiate a service or part of the service. A common system engineering principle to deal with scalability requirements is to introduce proper abstraction mechanisms that reduce the discovery time of resources while simplifying and optimizing their management.

Some examples of resource abstraction mechanisms are one-big-switch abstractions [1, 20], virtual-link abstractions [7, 8], and linear inequalities representation [22]. However, to the best of our knowledge, none of the existing approaches take into account service requirements to generate a logical network

[1]https://wiki.akraino.org/display/AK/Akraino+Edge+Stack
[2]https://wiki.onap.org/display/DW/Edge+Automation+through+ONAP

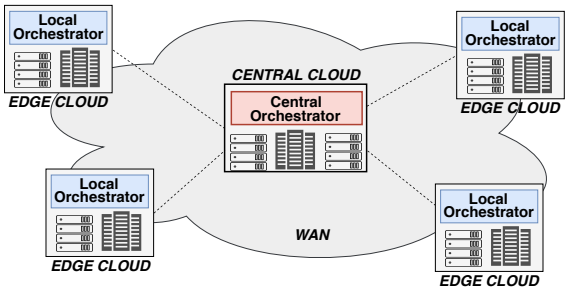


Fig. 1: A distributed cloud computing environment comprised by interconnected cloud sites and a centralized orchestrator communicating with local orchestrators in edge clouds.

inventory representation. The novel abstraction method presented in this paper is directed to the Abstracted Network Inventory (ANI) component (i) receiving services requirements from a catalog, (ii) receiving a network representation from a network inventory, and (iii) processing those inputs to generate an optimized abstract network representation, referred to as Logical Network Inventory (LNI). Using the LNI delivers two main advantages. First, a reduced time for placement of NFs since resource candidates for the placement are logically reduced, in terms of the number of compute nodes and links, in comparison to the original representation in the conventional network inventory. Second, a summarized topology per service can be used to simplify and optimize management of resources. An example of management is life-cycle operations in which service resources should be rearranged such as scaling NFs or workloads.

A challenging task for LNI generation is to find an optimal mapping of NFs within a network service to the components of a network inventory. To address this issue, we formalize a system model to solve the LNI generation problem based on network service requirements and infrastructure capabilities. Afterwards, we also develop three algorithms to build different types of LNIs efficiently: (i) Node-oriented LNI, (ii) Edge-oriented LNI, and (iii) Node/Edge-oriented LNI.
The main contributions of this paper are as follows:

- We propose the ANI as a novel component that allows the creation of service-optimized network inventory views in distributed cloud environments. To the best of our knowledge, ANI is the first approach that uses service requirements as an input to generate an abstract network inventory.
- We formally define a network model and problem statement along with the development of three algorithms to generate an LNI given the capacity-related resources and requirements of a network inventory and network service, respectively.
- We evaluate the proposed algorithms through extensive experiments using random and real-world topologies. Results show significant benefits of COs/LOs using an LNI to simplify (i) *service management*: the density in a network inventory is reduced between 1.8-2.7x; and (ii) *service placement*: up to 50% time saving rates after reducing less than 20% of the compute nodes.

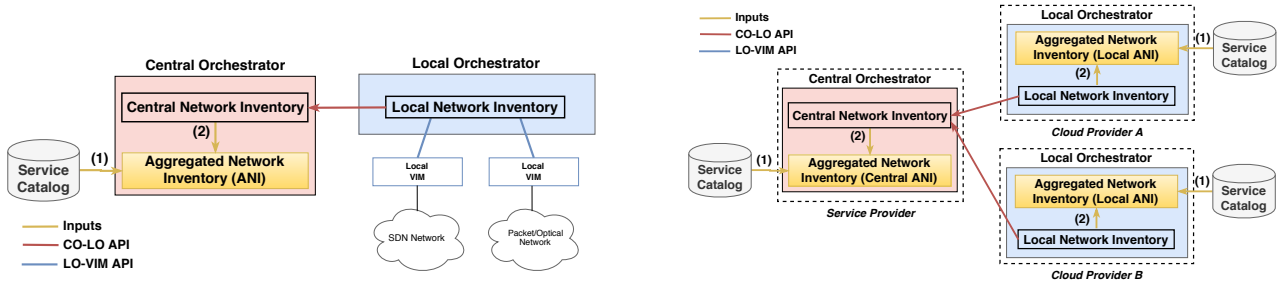
This paper is organized as follows. Section 2 covers related work for network inventory creation and abstraction. Section 3 describes in details the ANI component (key concepts, deployment, and LNI generation). Section 4 gives the formulation of the ANI model, including the problem statement. We introduce three algorithms for the LNIs generation in Section 5. Section 6 evaluates the LNI methods and its impact in the network service provisioning under two validation environments. Finally, we conclude the paper in Section 7 and point to our future work.

II. Related Work

Several solutions have been proposed for network inventory creation and abstraction [1, 7, 8, 15, 17, 19, 20, 22]. For example, ALTO [1] is a protocol that provides coarse-grained network information such as network locations and cost between them. ALTO uses maps to create abstract network topology representations and to express network costs between endpoints., UNIFY [20] provides an abstraction of type big-switch and big-software that includes compute and network resources. Solution in [22] uses linear inequalities to represent network resources availability in terms of bandwidth. However, all these solutions do not consider service requirements to generate a network representation. In addition, network representations are typically either coarse- or fine-grained. The former does not provide enough information from the infrastructure for placement decisions [20]. Fine-grained methods are too costly for a distributed cloud environment, meaning that since the number of infrastructure resources such as switches and compute devices can be very large in a distributed cloud environment. Solution in [22] does not consider network services in terms of NFs since it is a solution for workloads with available bandwidth requirements.

Likewise, [7, 8] provide different abstraction models (big-switch, virtual link with single weights, virtual link with multiple weights, and optical transport transformation) of optical transport networks, focusing especially on centralized radio access networks (C-RANs). However, both solutions also do not consider the use of service requirements as an input to generate an abstract network representation. In addition, abstraction models for cloud environments are out of scope, as stated by the authors.

Work in [15] uses abstraction strategies described in [8] for distributed data center network scenarios, however, it does not provide new abstraction mechanisms to represent information related to the network infrastructure. [19] supports grouping by joining entities within hypernodes or hyperedges. Our proposal has two main differences (i) filtering instead of aggregation as abstraction mechanism and (ii) service requirements as an additional input. Finally, ONAP AAI [17] is a component that provides real-time network inventory of infrastructure resources, but there is no defined an abstraction process.



(a) (b)

Fig. 2: Exemplary system architectures in which the ANI component is (a) executed as part of a CO component or (b) in a distributed manner across several orchestrators components, *i.e.*, CO and LOs.

III. Abstracted Network Inventory (ANI)

A. Basic Definitions

Network Inventory. Network infrastructure resources are represented in the form of a network inventory. A network inventory may comprise nodes[3] and links[4] each providing a particular capacity. Examples of node-related capacities are number of CPUs, amount of RAM (*e.g.*, 32 GB of RAM), and amount of disk space (*e.g.*, 10 TB of disk space). Examples of link-related capacities include bandwidth characteristics (*e.g.*, 100 Gbps) and latency characteristics (*e.g.*, 1 ms RTT).

Network Service. A network service, or simply service, is deployed or instantiated over the infrastructure resources. A service specifies one or more nodes (a set of required NFs) as well as links (how NFs are connected) [10]. Nodes include resource demands (*e.g.*, CPU, memory, storage) and links contain performance objectives (*e.g.*, latency, bandwidth).

B. ANI Component & Deployment

The ANI component proactively constructs multiple network views over the same network infrastructure, called LNIs. Each LNI is optimized to a service in terms of its requirements such as CPU, memory, latency, etc. Every new service in a catalog triggers the creation of another LNI that will be part of the optimized network inventory. As such, service requirements are used in the method to guide the right level of abstraction.

The ANI component may be executed as part of the CO (see Fig. 2a). As shown in the figure, the ANI receives two inputs: (i) services requirements from a catalog, and (ii) a central network inventory representation. The CO component may build a central network inventory based on local resource infrastructure information received from one or more LO, which maintain a local network inventory. In another variant (see Fig. 2b), the ANI component may be executed in a distributed manner across a service provider (hosting the CO component), and several cloud providers (hosting the LO component).

[3]The terms node and vertex are used interchangeably

[4]The terms link and edge are used interchangeably

C. Logical Network Inventory (LNI)

A two-step procedure is performed to generate a LNI. In the first step, a classification of a service is carried out to determine whether the service is node-oriented, edge-oriented, or node/edge-oriented. In the second step, the actual node and/or edge oriented mode is executed over the network inventory to generate an LNI.

Service Classification. This classification may be executed: (i) determining a reference value of node and/or link capacities in the network inventory. This value can be computed as the sum of available node/link capacities. (ii) calculating the total of required node and link capacities from the service, and (iii) comparing both values to determine a Provisioning Index (PI)[5]. If the calculated PI indicates node underprovisioning (*i.e.*, that not enough node resources are available for the service), it is determined that the service is classified as node-oriented. If, on the other hand, the PI value indicates overprovisioning (*i.e.*, that node resources are available in abundance), it is indicated that the service is classified as edge-oriented mode. Otherwise, the service is classified as node/edge oriented.

LNI Generation. The process of building up an LNI is in accordance with the service classification, so that we have three types of LNIs: (i) node-oriented LNI, (ii) edge-oriented LNI, and (iii) node/edge-oriented LNI. Section 4 provides more detailed information about this generation.

- *Node-oriented LNI.* A set of vertices in the network inventory are assigned to the LNI (links are discarded). Such selected vertices have to support the CPU capacity constraint of NF nodes in a service.
- *Edge-oriented LNI.* A set of edges in the network inventory are assigned to the LNI. Selected edges have to support the required bandwidth capacity of edges in a service.
- *Node/Edge-oriented LNI.* A set of vertices and edges in the network inventory are assigned to the LNI. Vertices and edges are selected according to the CPU and bandwidth capacity constraints in a service, respectively.

An illustrative example of this process is shown by Figure 3, where orange nodes (dashed borders) and blue links (dashed

[5]Different mechanisms may be used to compute a PI (out of the scope of the paper)

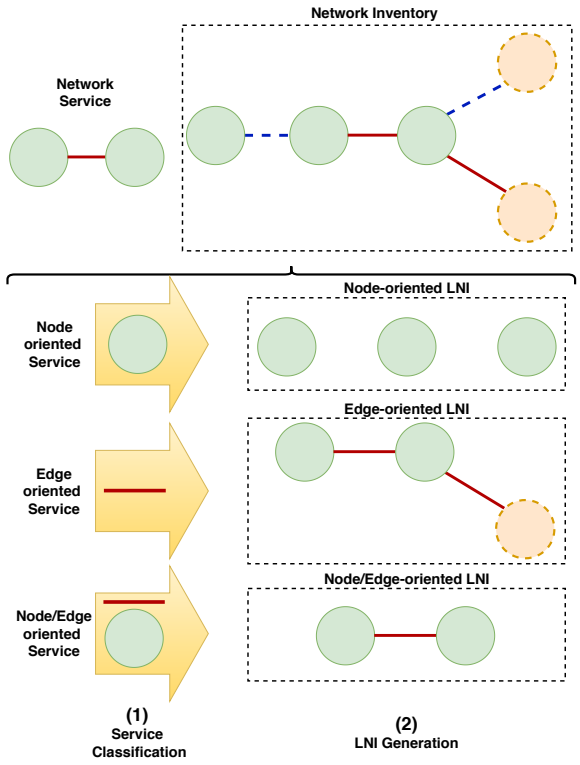


Fig. 3: Two-step procedure to create a Logical Network Inventory (LNI).

lines) could become logically discarded of the network inventory based on the network service demands (green nodes and red link) and service classification (node-oriented, edge-oriented, or node/edge-oriented).

IV. Network Model & Problem Statement

A. Network Model

Network Service. We model a network service as a directed graph denoted by $G_s = (V_s, E_s)$, where V_s is a set of NFs connected via a set of directed edges E_s. Each NF $v_s \in V_s$ is associated with a requested CPU capacity value cpu_{v_s}. Each edge $e_s(y, z) \in E_s$, connecting two NFs y and z, is associated with a requested bandwidth capacity value bw_{e_s}[6].

Network Inventory. In its basic form, a network inventory is modeled as an undirected graph $G_i = (V_i, E_i)$, where a vertex $v_i \in V_i$ has an available CPU capacity (CPU_{v_i}), and an edge $e_i(m, n) \in E_i$, between two vertices m and n , is associated with a bandwidth capacity BW_{e_i}. We also denote the set of all loop-free paths from the source vertex s to the destination vertex d by $P_i(s, d)$. Therefore, the available bandwidth capacity of a path $p_i \in P_i$ is given by:

$$BW(p_i) = \min_{e_i \in p_i} BW(e_i)$$

[6]We are considered a basic scenario, with only CPU and bandwidth constraints. Additional capacity constraints of nodes (*e.g.*, memory) and edges (*e.g.*, delay) can be easily extended in our model.

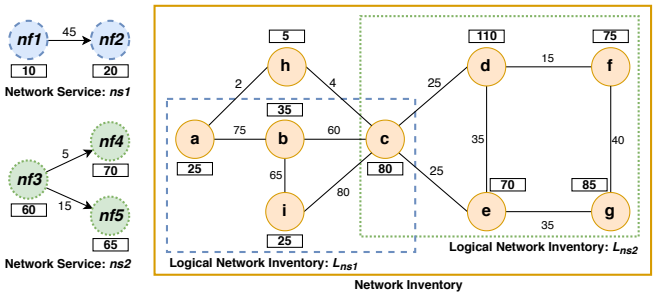


Fig. 4: An example of Network Service & Network Inventory & Logical Network Inventory (LNI).

In order to support graph collections, our network inventory model is extended to include a set of LNI graphs $G_i = (V_i, E_i, LNI_i)$, where LNI_i represents multiple possible views of the same network inventory.

We model a LNI graph $L_l \in LNI_i$ as an undirected graph denoted by $L_l = (V_l, E_l)$ where V_l is a subset of vertices such that $V_l \subseteq V_i$, and E_l is a subset of edges such that $E_l \subseteq E_i$. Besides, LNI graphs may overlap such that $\forall L_y, L_z \subseteq LNI_i : |V(L_y) \cap V(L_z)| \geq 0 \wedge |E(L_y) \cap E(L_z)| \geq 0$.

B. Problem Statement

Given a service request $G_s = (V_s, E_s)$ and a network inventory $G_i = (V_i, E_i, LNI_i)$, our objective is to determine a LNI graph $L_l \in LNI_i$ optimized for the service requirements in terms of vertex and edge constraints.

We illustrate our problem statement with an example given in Figure 4. The left side of this figure shows two network services, where the numbers in rectangles represent requested CPU capacity, and the numbers near the links represent required bandwidth capacity. The network service 1 (*ns1*) connects two NFs (*nf1* and *nf2*) with 45 units of bandwidth on the edge between them. The network service 2 (*ns2*) requires the bandwidth 5 over the edge (*nf2*, *nf4*) and 15 units over the edge (*nf3*, *nf5*), and the CPU resources 60, 70, 65 at NFs nodes, *nf3*, *nf4*, and *nf5*, respectively. Figure 4 (right side) also depicts a network inventory. The number near the links is the available bandwidth, and the numbers in rectangles represent the available CPU resources at the vertices. Once the process of determining an LNI is performed, the network inventory contains two logical graphs $LNI_i = \{L_{ns1}, L_{ns2}\}$, where each graph has a dedicated subset of vertices and edges, which represent an optimized network view to the requirements of each network service. Note that also a vertex (v_c) is overlapped since $V(L_{ns1}) \cap V(L_{ns2}) = \{v_c\}$.

V. Proposed Algorithms

The primary objective here is to design three algorithms that efficiently create LNIs. The ANI component considers three modes of LNIs generation:

Node-oriented LNI. It takes a network service $G_s = (V_s, E_s)$ and a network inventory $G_i = (V_i, E_i, LNI_i)$ as input and returns a set of vertices $V(L_y)$ such that $L_y \subseteq LNI_i$.

Algorithm 1: Node-oriented LNI

```
Input:
G_s = (V_s, E_s): Network Service;
G_i = (V_i, E_i, LNI_i): Network Inventory;
Output:
V(L_y), ∀L_y ∈ LNI_i and V_l ⊆ V_i;
1 foreach v_s in V_s do
2     foreach v_i in V_i do
3         if CPU(v_i) ≥ cpu(v_s) then
4             V(L_y) ← V(L_y) ∪ {v_i};
```

For each service node v_s, the algorithm searches all nodes v_i in the network inventory and adds nodes with CPU capacity greater than or equal to the CPU requirement (Line 3) into the subset of vertices $V(L_y)$ (Line 4). The pseudo-code of this mode is provided in Algorithm 1.

Edge-oriented LNI. Algorithm 2 provides an overview of this proposed mode. It also takes a network service $G_s = (V_s, E_s)$ and a network inventory $G_i = (V_i, E_i, LNI_i)$ as inputs. However, it returns a set of vertices $V(L_y)$ and a set of edges $E(L_y)$ as output.

For each service edge e_s, we first iterate all the possible network inventory nodes to get a source node $v_{i_{src}}$ and a destination node $v_{i_{dst}}$ (Lines 1-6). After, the algorithm finds the paths from $v_{i_{src}}$ to $v_{i_{dst}}$ (Line 7) and only considers those paths that respect the bandwidth requirement in a service link $bw(e_s)$ (Line 8). Finally, all the nodes and edges which are in a path p_i (Line 9) will be part of the subset of vertices $V(L_y)$ and edges $E(L_y)$ (Lines 10-12).

Node/Edge-oriented LNI. This algorithm (see Algorithm 3) is quite similar to Algorithm 2 except that the CPU constraints of service nodes should also be satisfied by the CPU capacity of network inventory nodes (Lines 4 and 7).

Algorithm 2: Edge-oriented LNI

```
Input:
G_s = (V_s, E_s): Network Service;
G_i = (V_i, E_i, LNI_i): Network Inventory;
Output:
V(L_y), ∀L_y ∈ LNI_i and V_l ⊆ V_i
E(L_y), ∀L_y ∈ LNI_i and E_l ⊆ E_i;
1  foreach e_s(src, dst) in E_s do
2      v_{i_src} ← ∅, v_{i_dst} ← ∅;
3      foreach v_i in V_i do
4          v_{i_src} ←v_i;
5          foreach v_i in V_i do
6              v_{i_dst} ←v_i;
7              foreach p_i in P_i(v_{i_src}, v_{i_dst}) do
8                  if BW(p_i) ≥ bw(e_s) then
9                      foreach e_i(s, d) in p_i do
10                         V(L_y) ← V(L_y) ∪ s;
11                         V(L_y) ← V(L_y) ∪ d;
12                         E(L_y) ← E(L_y) ∪ e_i;
```

Algorithm 3: Node/Edge-oriented LNI

```
Input:
G_s = (V_s, E_s): Network Service;
G_i = (V_i, E_i, LNI_i): Network Inventory;
Output:
V(L_y), ∀L_y ∈ LNI_i and V_l ⊆ V_i
E(L_y), ∀L_y ∈ LNI_i and E_l ⊆ E_i;
1  foreach e_s(src, dst) in E_s do
2      v_{i_src} ← ∅, v_{i_dst} ← ∅;
3      foreach v_i in V_i do
4          if CPU(v_i) ≥ cpu(src) then
5              v_{i_src} ←v_i;
6              foreach v_i in V_i do
7                  if CPU(v_i) ≥ cpu(dst) then
8                      v_{i_dst} ←v_i;
9                      foreach p_i in P_i(v_{i_src}, v_{i_dst}) do
10                         if BW(p_i) ≥ bw(e_s) then
11                             foreach e_i(s, d) in p_i do
12                                 V(L_y) ← V(L_y) ∪ s;
13                                 V(L_y) ← V(L_y) ∪ d;
14                                 E(L_y) ← E(L_y) ∪ e_i;
```

VI. Experimental Evaluation

In this section, we analyze the performance of our proposed algorithms. Two evaluation environments are considered: (*i*) quality of the LNI quality using randomly generated network inventory topologies, and (*ii*) LNI impact on network service provisioning time using a real graph dataset. For each experiment, we first describe the experimental setup, and then present and discuss the evaluation results.

The platform used in all the experiments is an Intel® Core™ I7-4790 @ 3.60GHz x 8 with 16GB RAM, running Ubuntu 14.04LTS (Linux) 64-bit. For reproducibility purposes, all supporting codes are publicly available in our research group repository.[7]

A. LNI Quality Evaluation

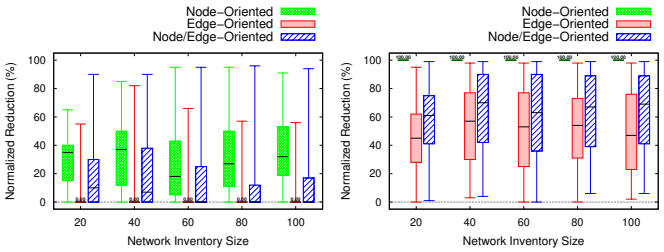
This experiment mainly serves to illustrate the performance of our three proposed algorithms to generate a LNI.

Simulation Setup. Network inventory topologies are randomly created using Networkx library in Python. The number of nodes varies between 20 and 100, and each pair of nodes are randomly connected with probability 0.5. The CPU and bandwidth capacity is a number uniformly distributed between 1-16 and 1-100, respectively. LNIs are created from service requests. Each service has 2 NFs (*i.e.*, 2 nodes). The CPU demand of each NF is normally distributed between 1 and 16 and the bandwidth requirement of each link is a number between 1 and 100, uniformly distributed.

Performance Metrics. We use two measures of nodes/edges reduction and degree to evaluate the quality of LNIs generated by the three different algorithms: (i) Node-oriented LNI (N-LNI), (ii) E-oriented LNI (E-LNI), (iii) Node/Edge-oriented LNI (N/E-LNI).

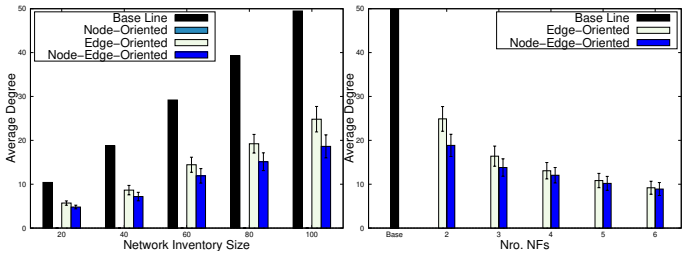
For each series of experiments, we randomly generate 100 service requests with two NFs. A new LNI is generated from a service request and then the percentual node and edge reduction and average degree that a LNI topology obtains, using the three algorithms, is compared to the same metrics in the full network inventory topology (used as a baseline). In case of the average degree evaluation, we are considering two variations: (i) different network inventory topology sizes with a service composed of two NFs and (ii) same network

[7]https://github.com/intrig-unicamp/ani



(a) Nodes Reduction (b) Edge Reduction

Fig. 5: Normalized reduction in nodes (a) and edges (b) using the three algorithms: Node-oriented LNI, Edge-oriented LNI, and Node/Edge-oriented LNI



(a) Degree (b) Degree: 2-6 NFs

Fig. 6: The average degree at 95% of confidence level obtained from three algorithms (Node-oriented LNI, Edge-oriented LNI, and Node/Edge-oriented LNI) with different topology sizes (a) and with different amount of NFs (b).

inventory topology size (100 nodes) but increasing the number of NFs from 2 to 6.

Simulation Results. Figures 5 shows the normalized reduction of nodes and edges (as candlesticks with median, quartiles, and max/min values) with different network topology sizes by N-LNI, E-LNI, and N/E-LNI. The node reduction (see Fig. 5a) by N-LNI achieves the higher values, and its edge reduction is 100%. This is because N-LNI only considers nodes supporting the CPU constraints, and edges are discarded. Besides, E-LNI achieves the lowest node reduction value since, it traverses all the nodes without restrictions. On the other hand, the edge reduction measure (see Fig. 5b) by N/E-LNI is higher than E-LNI. This behavior is expected because N/E-LNI only considers nodes and edges supporting CPU and bandwidth constraints, respectively.

Figure 6a shows the average degree comparison between the three algorithms with different topology sizes. The degree measure is always 0 for N-LNI because edges are no longer considered, and therefore all the nodes are isolated. N/E-LNI achieves a lower degree that E-LNI. This is because, as we just mentioned, N/E-LNI combines both nodes and edge restrictions. Also, generated network inventories are densely connected (see baseline values); however, E-LNI and N/E-LNI always achieve much lower degree values (1.8-2.7x) than the baseline.

To further test the effect of the number of NFs on the LNI quality, we set services with different amounts of NFs (2-6). Figure 6b shows the degree comparison between E-LNI and N/E-LNI on a network inventory topology with 100 nodes. As shown in the figure, the value of the degree is decreasing as the number of NFs increases. This is because a new NF adds new CPU and bandwidth constraints, therefore we can infer that the number of NFs decreases the nodes and/or edges supporting such requirements.

B. Network Service Provisioning

We now focus on quantifying the time it takes for a CO, in terms of execution time, to map a requested network service considering an optimized network inventory (*i.e.*, an LNI).

Simulation Setup. We consider a real wide-area network topology obtained from the Internet Topology Zoo project [12]. Specifically, we extend the *Interoute* topology — one of Europe's largest cloud service providers[8] — to represent a full network inventory. This network topology is composed of 110 nodes connected through 148 links[9]. As with the previous experiment, node and link capacities are uniformly distributed between 1-16 and 1-100, respectively.

In case of the CO, we opt to re-use an exiting open-source software tool, called ESCAPE (Extensible Service ChAin Prototyping Environment)[10]. ESCAPE [4] is a framework that supports the development of several parts of the service chaining architecture and it can run a number of emulated LOs with emulated interfaces to load information about a local network inventory. ESCAPE also includes a simple service layer where users can request services.

Performance Metrics. ESCAPE uses a heuristic-based greedy backtracking algorithm to map service requests to a network inventory topology. In this experiment, we measure the ESCAPE's time saving rate for mapping of a service. This value is defined as a fraction of the amount of saved time using a LNI topology (generated by Node/Edge-oriented LNI algorithm) out of running time using a full network inventory topology. We generate 50 different LNIs from 50 service requests. Each service is with two NFs where the CPU and bandwidth demands are normally distributed between 1 and 16 and between 1 and 100, respectively.

Simulation Results. Figure 7 presents the average saving time at 95% of confidence level according to the percentage of reduced nodes and edges when using a Node/Edge-oriented LNI and when using a full network inventory. Results indicate significant improvements in terms of saving time when the orchestrator uses LNI compared to the approach in which a full topology is used. More specifically, the key observation here is: a CO can achieve up to 50% time saving for service deployment with the reduction of less than 20% of nodes in a network inventory. Even, with a reduction of nodes and edges by less than 55%, it is possible to obtain up to 75% time saving rate.

[8]Interoute was acquired by GTT Communications in 2018 [3]
[9]http://www.topology-zoo.org/files/Interoute.gml
[10]https://github.com/5GExchange/escape

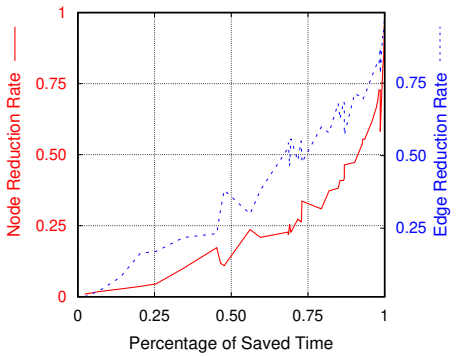


Fig. 7: Time saving rate for mapping a service according to the percentage of reduced nodes and edges using a Node/Edge-oriented LNI.

VII. Conclusion & Future Work

One of the foremost challenges for management systems in distributed cloud environments is how to effectively handle the scale and complexity of network service placement and management considering actual resource inventories. This paper contributes with a novel component called Abstracted Network Inventory (ANI) that generates optimized network views called Logical Network Inventory (LNI) based on network service requirements and network inventory capabilities. Our results reveal that, when using an LNI methodology, we can reduce the time to place network services while optimizing the management of resources by following the principle of abstraction, *i.e.*, by logically reducing the sets of candidate resources in terms of compute nodes and links.

We reckon that, as the deployment size and heterogeneity complexity of softwarized networks increase, properly applying fundamental software principles like layering (*e.g.*, SDN controller foundations), indirection (*e.g.*, overlay tunnels), or abstraction (*e.g.*, LNI), just to cite a few examples, will be more important than ever to deliver manageable systems.

For future work, it will be beneficial to extend the three proposed offline algorithms (which typically comes at the expense of long runtimes) to online algorithms implemented via heuristics. Also, future activities include extending our network model to consider other decision variables such as cost and revenue. Embodiments of the proposed methods as extensions to the ALTO protocol and more specific use cases driven by industry partners are also in our roadmap.

Acknowledgment

This research was supported by the Innovation Center, Ericsson S.A., Brazil, grant UNI.64. The views expressed are solely those of the authors and do not necessary represent Ericsson's official standpoint.

References

[1] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Woundy. Application-layer traffic optimization (alto) protocol. RFC 7285, RFC Editor, September 2014. http://www.rfc-editor.org/rfc/rfc7285.txt.

[2] AT&T. AT&T Edge Cloud (AEC) - White Paper, 2017.

[3] GTT Communications. GTT Completes Acquisition of Interoute. https://www.gtt.net/gb-en/news/press-releases/gtt-completes-acquisition-of-interoute/, 2018.

[4] Attila Csoma, Balázs Sonkoly, Levente Csikor, Felicián Németh, Andràs Gulyas, Wouter Tavernier, and Sahel Sahhaf. Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox. *SIGCOMM Comput. Commun. Rev.*, 44(4):125–126, August 2014.

[5] Nathan F Saraiva de Sousa, Danny A Lachos Perez, Raphael V Rosa, Mateus AS Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *Computer Communications*, 2019.

[6] Ericsson. Distributed cloud – a key enabler of automotive and industry 4.0 use cases. https://www.ericsson.com/en/ericsson-technology-review/archive/2018/distributed-cloud, 2018.

[7] Matteo Fiorani, Ahmad Rostami, Lena Wosinska, and Paolo Monti. Transport abstraction models for an sdn-controlled centralized ran. *IEEE Communications Letters*, 19(8):1406–1409, 2015.

[8] Matteo Fiorani, Ahmad Rostami, Lena Wosinska, and Paolo Monti. Abstraction models for optical 5g transport networks. *IEEE/OSA Journal of Optical Communications and Networking*, 8(9):656–665, 2016.

[9] Riccardo Guerzoni, Ishan Vaishnavi, David Perez Caparros, Alex Galis, Francesco Tusa, Paolo Monti, Andrea Sganbelluri, Gergely Biczók, Balasz Sonkoly, Laszlo Toka, et al. Analysis of end-to-end multi-domain management and orchestration frameworks for software defined infrastructures: an architectural survey. *Transactions on Emerging Telecommunications Technologies*, 28(4):e3103, 2017.

[10] J. Halpern and C. Pignataro. Service function chaining (sfc) architecture. RFC 7665, RFC Editor, October 2015.

[11] Yosr Jarraya, Taous Madi, and Mourad Debbabi. A Survey and a Layered Taxonomy of Software-Defined Networking. *IEEE Communications Surveys & Tutorials*, 16(4):1955–1980, 2014.

[12] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[13] Diego Kreutz, Fernando M V Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, 1 2015.

[14] Nam Tuan Le, Mohammad Arif Hossain, Amirul Islam, Do-yun Kim, Young-June Choi, and Yeong Min Jang. Survey of promising technologies for 5g networks. *Mobile information systems*, 2016, 2016.

[15] Melissa Licciardello, Matteo Fiorani, Marija Furdek, Paolo Monti, Carla Raffaelli, and Lena Wosinska. Performance evaluation of abstraction models for orchestration of distributed data center networks. In *2017 19th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE, 2017.

[16] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *Communications Surveys Tutorials, IEEE*, 2015.

[17] ONAP Platform. Active and Available Inventory Project. https://wiki.onap.org/display/DW/Active+and+Available+Inventory+Project, 2018.

[18] Farah Slim, Fabrice Guillemin, Annie Gravey, and Yassine Hadjadj-Aoul. Towards a dynamic adaptive placement of virtual network functions under onap. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on*. IEEE, 2017.

[19] Thomas Soenen, Sahel Sahhaf, Wouter Tavernier, Pontus Sköldström, Didier Colle, and Mario Pickavet. A model to select the right infrastructure abstraction for service function chaining. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 233–239. IEEE, 2016.

[20] Balázs Sonkoly, János Czentye, Robert Szabo, Dávid Jocha, János Elek, Sahel Sahhaf, Wouter Tavernier, and Fulvio Risso. Multi-domain service orchestration over networks and clouds: a unified approach. In *ACM SIGCOMM Computer Communication Review*. ACM, 2015.

[21] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z Sheng, and Rajiv Ranjan. A Taxonomy and Survey of Cloud Resource Orchestration Techniques. *ACM Computing Surveys*, may 2017.

[22] Qiao Xiang, J Jensen Zhang, X Tony Wang, Y Jace Liu, Chin Guok, Franck Le, John MacAuley, Harvey Newman, and Y Richard Yang. Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pages 27–29. ACM, 2018.