Necklace: An Architecture for Distributed and Robust Service Function Chains with Guarantees

(Article begins on next page)

19 April 2024

Necklace: An Architecture for Distributed and Robust Service Function Chains with Guarantees

Flavio Esposito* Maria Mushtaq* Michele Berno† Gianluca Davoli‡ Davide Borsatti‡ Walter Cerroni‡ Michele Rossi†

*Saint Louis University, USA   † University of Padova, Italy   ‡ University of Bologna, Italy

Abstract—The service function chaining paradigm links ordered service functions via network virtualization, in support of applications with severe network constraints. To provide wide-area (federated) virtual network services, a distributed architecture should orchestrate cooperating or competing processes to generate and maintain virtual paths hosting service function chains while guaranteeing performance and fast asynchronous consensus even in the presence of failures. To this end, we propose a prototype of an architecture for robust service function chain instantiation with convergence and performance guarantees. To instantiate a service chain, our system uses a fully distributed asynchronous consensus mechanism that has bounds on convergence time and leads to a $(1 - 1/e)$-approximation ratio with respect to the Pareto optimal chain instantiation, even in the presence of (non-byzantine) failures. Moreover, we show that a better optimal chain approximation cannot exist. To establish the practicality of our approach, we evaluate the system performance, policy tradeoffs, and overhead via simulations and through a prototype implementation. We then describe our extensible management object model and compare our asynchronous consensus's overhead against Raft, a recent decentralized consensus protocol, showing superior performance. We furthermore discuss a new management object model for distributed service function chain instantiation.

Index Terms—Network Virtualization, Service Function Chains, NFV, Consensus Algorithms, Guarantees.

I. Introduction

THE growing number of networked services and applications increased the complexity of network management. Network virtualization has however simplified many aspects of such complexity, while also enabling new business models; the key idea is to abate infrastructure and service providers' operational costs [1]. The research community has also gained interest in this technology as it allows simpler management of multi-tenant, wide-area and complex networks; see, for example, large scale virtual network testbeds such as Chameleon [2], or Fabric [3]. To offer such wide-area network services, physical resources from multiple federated providers or research institutions are required. Managing these services is, however, very complex. The management complexity is brought by the scale and constraints required by the applications, as well as by multiple virtual network functions that coexist to build a *service function chain*. Such functions range from management tasks, such as load balancing (sometimes also considered a data plane task), firewalls, intrusion detection systems, network address translators or deep-packet inspection [4] and Quality-of-Service [?], [5]–[7], to data plane tasks such as congestion control, and network scheduling [8], [9].

To orchestrate such complex environments, chain policies need to be instantiated on multiple (physical or virtual) hosting machines, spanning multiple federated providers, and being dynamically reprogrammed to quickly adapt to the dynamic nature of the service. By policy, we mean a variant aspect of any network mechanism, *e.g.*, a desirable high-level goal dictated by users, applications, infrastructure or service providers.

Many systems for centralized service function chain orchestration are actively being developed, see *e.g.*, [10], [11]. It is still unknown, however, how multiple (virtualized) instances of the infrastructure could cooperate or compete to orchestrate several network management mechanisms to offer a wide-area service function chain.

In this paper, we focus on a subset of all possible (distributed) chain orchestration mechanisms. In particular, we present the design and implementation of *Necklace*, an architecture that solves the distributed chain instantiation problem with performance guarantees via distributed asynchronous consensus, even in the presence of a small number of random failing processes or communication links. Necklace solves the problem of reaching a distributed allocation agreement among processes running on all chain hosting nodes, while maximizing all providers' utilities.

Motivating applications. Solutions that provide resilient decentralized asynchronous consensus that could be used for the chain instantiation problem already exist, see *e.g.*, the Paxos consensus algorithm [12], a version of which is used even by Google data centers [13], or the more recent Raft [14], adopted by the Open Network Operating System (ONOS) [10]. The design behind these protocols is sound, but although these approaches have been subject to recent optimizations and improvements (see *e.g.*, [15], [16]), none of them simultaneously provides: (i) guarantees on the Pareto optimality of the elected leader, (ii) bounds on the agreement convergence time, and (iii) resilience to random failures of processes and/or communication links, via a fully distributed solution (as opposed to a decentralized one as in [14]). Our Necklace architecture bridges this gap with the following three contributions.

Architectural contributions. Leveraging stochastic optimization theory, we identify the mechanisms and the interactions within what we call the *complete resilient service function chain instantiation problem*. The three necessary and sufficient sets of invariances to instantiate a chain are: (i) state retrieval, (ii) chain mapping, and (iii) resource binding. Necklace connects these three mechanisms via a Chain Instantiation Protocol (CIP) that modifies states within each phase.

Algorithmic contributions. To solve the service chain mapping problem, we propose a fully Distributed Asynchronous Chain Consensus Algorithm (DACCA) that is guaranteed to converge and has probabilistic guarantees (*i.e.*, guarantees on the expected value) on the quality of the chain instantiation, with respect to a Pareto optimal network utility.

System contributions. We analyze some policy tradeoffs of the DACCA mechanism with simulations, and we confirm the simulated results over a prototype implementation of Necklace within the Mininet virtual network testbed [17]. Using a large available dataset of requests to the Facebook datacenter, we also compare the performance of several predictors used to provision a chain, and deploy such chains after running DACCA using an OpenSource MANO (OSM) physical network testbed using production-level hardware to establish the practicality of our approach.

Paper Organization. The rest of the paper is organized as follows: in the next Section II we relate our contributions to existing work; in Section III we define our complete resilient chain instantiation problem as a stochastic optimization problem, that served as architecture design tool; our Necklace architecture and its core mapping mechanism DACCA are then described in Sections IV and V, respectively, while the details of the management object model, a superset of the Chain Instantiation Protocol are described in Section IV. The analytical results are highlighted in Section VI. In Section VII, instead, we present the traffic forecast analysis, performed using the Facebook dataset [18]. Finally, we present our prototype evaluation results in Section VIII and draw our conclusions in Section IX.

II. RELATED WORK

Service chain orchestration systems. The idea of providing a resilient chain in a distributed environment has been floated at the ETSI NFV ISG group [19], proposing a system in which chains are deployed with automatic failover and reinstallation of failed instances. Moreover, their proof-of-concept include redundancy with middleware that provides state replication and synchronization services between chain instances. Necklace does not yet support chain reinstallation upon failover but its focus is on an algorithm for instantiating a chain, that is resilient during the instantiation phase. Necklace's asynchronous consensus mechanism does not distinguish failed hosting nodes from silent nodes unable to take the "hosting leadership". Moreover, ETSI's solution does not allow each provider to specify their own instantiation policies (utility or voting strategy) privately, and their system does not have bounds on suboptimalities. Although also without guarantees, a discussion of an optimal distribution of chains in a multi-provider environment is discussed in [20]. A use case by the IETF service function chaining working group [21] considers instantiating chains across federated domains; however, no system has been implemented yet in support of those use cases. MIDAS [22] and Cloud4NFV [23] are other two approaches for multi-cloud environments chain provisioning (MIDAS also considers discovery). Their approaches are sound but many of

them are ad-hoc, *i.e.*, they do not allow policy tuning on the chain (middlebox) instantiation mechanism and do not discuss any guarantees.

Several other NFV orchestration system designs have been proposed [24], see, *e.g.*, OpenBaton [25], FROG [26], and several other solutions [27] have been built upon the service function chain working group proposal, to explore, expand and identify new abstractions, especially at the intent-specification level [28]. Differently than OpenBaton, FROG, nf.io and others, the design of Necklace does not depend on the (flawed [29]) naming and addressing architecture, *i.e.*, it does not need to use IP addresses. Finally, Necklace shares the distributed and unifying approach design principles with [27], but the implementation of our prototype is limited to the management of the lifetime of a service function chain; other proposed orchestrators (*e.g.*, OpenBaton [25]) have other very useful features. OpenNF [30] does service chain orchestration using the forwarding mechanism. It assumes that dynamic service chaining is provided by updating how SDN switches forward packets. OpenNF key point is efficiency, and ability to coordinate control of forwarding changes and middlebox state migration, so that middleboxes can be replaced quickly and safely. The more recent Dysco [31] like the Chain Instatiation Protocol (CIP) of Necklace, is a session protocol and places no constraints on the choice of the control plane (*i.e.*, it works also without SDN), hence avoiding performance risk problems with (OpenNF) controllers because they are responsible for packet buffering. Other architectures, including [32], focus on the deployment of a pre-allocated chain, without covering resource allocation mechanisms. To conclude, none of these proposed architectures or orchestrators is based on a mechanism design able to provide guarantees on both convergence of a chain instantiation, and guarantees on performance utilization of a set of chains to be instantiated.

Constrained path finder algorithms. Since a chain is a (directed) constrained path, approaches to find a (resilient) physical path with multiple constraints are also related. Due to its NP-complete [33] nature, the problem of finding a constrained path has inspired many heuristics. As in our approach, most of these heuristics group multiple metrics into a single function to reduce the optimization problem to a single constrained formulation, see *e.g.*, [34], and then solve it using, *e.g.*, Lagrangian relaxation. Our architecture uses a mechanism that performs a fully distributed resource discovery with the voting system, before mapping the path on the hosting infrastructure, and then a subsequent phase assigns the best candidate, given by the Pareto optimal mechanism design.

Other path finder suboptimal solutions that use k-shortest paths like Necklace also exist: in [35] *e.g.*, the authors propose a k-path constraints heuristic solution. Moreover, the exact pseudo-polynomial algorithm proposed by Jaffe *et al* [33] offers a distributed path finder alternative, but restricted to a two-paths constraint. Differently from all these solutions, to our knowledge, we are the first to propose a constrained path finder mechanism that is policy-based, fully distributed, and has guarantees with respect to a Pareto optimal chain instantiation.

III. MODEL AND ARCHITECTURE DESIGN

In this section, we describe the complete resilient chain instantiation as a centralized stochastic optimization problem. We then use our model to design a system architecture in support of its resilient system implementation. In particular, we model each subproblem individually as a building block of our architecture, and we model the interfaces among such mechanisms with coupling (or complicating) variables.

Before running a chain of (virtual) services on a shared (physical) infrastructure, service and infrastructure providers need to cooperatively or independently run three interacting mechanisms: (i) chain state retrieval, (ii) service chain mapping, and (iii) resource binding.

The *state retrieval* mechanism involves querying a subset of all the available hosting resources, among those that satisfy the constraints of the requested service chain. These resources are physical, in case an infrastructure provider is processing the instantiation, or virtual, in case a service provider acts as a broker and rents resources on top of a virtual overlay. We refer to such underlying resources simply as *hosting* nodes or links.

If a set of hosting resources has been found, a resource mapping protocol has to be executed. It is known that finding unconstrained shortest paths can be solved in polynomial time [36]. However, due to the combination of node and link constraints, this path finding problem is the most complex step in the chain instantiation setup, and it has been shown to be NP-hard when several path Service Level Objective (SLO) constraints are enforced [37]. A SLO is a technical requirement within a Service Level Agreement (the full legal contract). Underlying or requested SLO constraints include intra-node, *e.g.*, desired physical location, processor speed or storage capacity, as well as inter-node constraints, *e.g.*, physical network topology.

Before data plane packets can flow on the newly instantiated service chain, an additional hosting-hosted resource binding step is necessary to reserve the resources and update all the network states. A system component dedicated to this phase has to check for additional capacity constraints or topological dependencies, as described in the RFC [38], and choose among all solutions available, if any, from the previous two phases.

In this work, we model a chain with a constrained virtual path, for ease of notation denoted as a (directed) graph with a linear topology $H = (V_H, E_H, C_H)$, to be instantiated on a physical network graph $G = (V_G, E_G, C_G)$, where V is a set of nodes, E is a set of links, and each node or link $e \in V \cup E$ is associated with a capacity constraint $C(e)$. [1]

A. Hosting Node State Retrieval (Resource Discovery)

A chain hosting element is available if a discovery (or state retrieval) operation is able to find it, given a set of protocol parameters. For example, an application may wish to find the k-shortest paths from node n to node m, or it may wish to find as many available Virtual Machines (VMs)

---

[1] Each constraint could be an (ordered) set of constraints containing, *e.g.*, restrictions on location, delays, or even node capabilities, such as installed packages or firewall rules.

---

capable of hosting a given middlebox service, *e.g.*, a NAT or a parental control application, within a given number of hops to minimize latency. We assume that a chain request j is a path that contains $\eta_j > 0$ (virtual) nodes (we omit the trivial case of a chain composed of a single virtual machine), and $\eta_j - 1$ virtual links. We limit the discovery overhead of physical nodes and paths with parameters $A_{Nj} \geq \eta_j$ and $A_{Pj} \geq \eta_j - 1$, respectively. This means that at least one candidate for each resource to be instantiated within the chain needs to be found. Otherwise, the chain allocation process gets rejected. Since Necklace handles multiple simultaneous chain instantiations, we identify with $j \in J$ each request, where J is the set of all Service Function Chains (SFC) to instantiate. Among all possible resources, the state retrieval phase returns the subset that maximizes a given notion of utility. Such utilities may have the role of selecting resources that are closer — with respect to some notion of distance — to the given set of constraints $\mathcal{C}_j(e)$.

Let us introduce two sets of binary variables, n_{ij}^P and p_{kj}. n_{ij}^P is equal to 1 if the ith physical node, on which the chain instance subset may be deployed is available, and zero otherwise. Similarly, p_{kj} is equal to 1 if the kth physical path, is available to host the direct virtual link, and zero otherwise. If we denote by $u_{ij} \in \mathbb{R}$ and $\omega_{kj} \in \mathbb{R}$ the utility of hosting nodes and paths, respectively, then the state retrieval mechanism can be modeled as follows:

$$
\begin{aligned}
\underset{n^P, p}{\text{maximize}} \quad & \sum_{j \in J} \left(\sum_{i \in V_G} u_{ij} n_{ij}^P + \sum_{k \in \mathcal{P}} \omega_{kj} p_{kj} \right) \\
\text{subject to} \quad & \sum_{i \in V_G} \sum_{j \in J} n_{ij}^P - \sum_{j \in J} A_{Nj} \leq 0, \\
& \sum_{k \in \mathcal{P}} \sum_{j \in J} p_{kj} - \sum_{j \in J} A_{\mathcal{P}j} \leq 0, \\
& n_{ij}^P, p_{kj} \in \{0, 1\}, \quad \forall i, j, k,
\end{aligned}
\tag{1}
$$

where $\mathcal{P}$ is the set of all physical paths in G. After this phase is completed, architecturally, the set of available physical resources $\{n_{ij}^P, p_{kj}\}$ are passed to the SFC mapper via an interface.

B. Service Function Chain (SFC) Mapping

Among the subset of all physical resources potentially available to host a node of the chain, we now seek a non-empty subset of all feasible hosting paths. Note that finding a constrained (shortest) path may have polynomial or exponential time complexity, depending on the number of physical path and physical link constraints [37]. The *chain mapping* is the problem of finding a matching of H in G, such that each service instance in H is mapped onto one hosting node (underlying server or VM), and each virtual link is mapped onto at least a physical path p. Note that since hosting paths may span across different federated providers, packets may be forced to pass through a third party firm offering deep packet inspection as a service. Formally, the mapping is a function $\mathcal{Q} : H \to (V_G, \mathcal{P})$, where $\mathcal{Q}$ is called a *valid mapping* if all constraints of H are satisfied, and for each virtual link $l^H = (s^H, r^H) \in E_H$, $s^H, r^H \in V_H$, $\exists$ at least one physical path (i.e., sequence of physical nodes) $p = (s^G, \ldots, r^G)$ such

that $p \in \mathcal{P}$ and virtual nodes s^H and r^H are mapped to physical nodes s^G and r^G, respectively.

We model the sets of available paths and nodes at the time of the chain instantiation request j with $\mathcal{P}'_j \subseteq \mathcal{P}$ and $V'_{G_j} \subseteq V_G$, respectively. The mapper returns a list of candidate nodes and links to the resource binder which decides the final binding between hosting and hosted resources.

The mapping phase can hence be modeled by the following optimization problem:

$$
\begin{aligned}
\underset{n^V, l}{\text{maximize}} \quad & \mathbb{E}\left[\sum_{j \in J}\left(\sum_{i \in V'_{G_j}} \Theta_{ij} n^V_{ij} + \sum_{k \in \mathcal{P}'_j} \Phi_{kj} l_{kj}\right)\right] \\
\text{subject to} \quad & \sum_{i \in V'_{G_j}} n^V_{ij} = \rho_j, \quad \forall j \in J, \\
& \sum_{k \in \mathcal{P}'_j} l_{kj} = \gamma_j, \quad \forall j \in J, \\
& n^V_{ij}, l_{kj} \in \{0, 1\}, \quad \forall \; i, j, k,
\end{aligned}
\tag{2}
$$

where Θ_{ij} is the utility that the system would get if the chain request j gets assigned to virtual node i, and Φ_{kj} is the system's utility when j gets the virtual link k, $\rho_j > 0$ is the number of virtual nodes, and $\gamma_j \geq 0$ the number of virtual links, respectively, requested with the chain request j. Note that ρ_j and γ_j differ from A_{N_j} and A_{P_j} defined in the resource discovery problem (1), as we may not need to use all physical nodes or physical paths that we have discovered in the mapping phase. Also, n^V_{ij} and l_{kj} are binary variables that are set to one if virtual node i or virtual link k have been assigned to chain request j, and zero otherwise.

The first two constraints enforce that all the virtual resources requested by each user are mapped, $i.e.$, at least one hosting node (path) is going to be assigned to each virtual node (link) of the requested chain, respectively. The third constraint ensures that the one-to-one mapping between virtual and hosting nodes is satisfied.

C. Chain Resource Binding

After all mapping candidates have been identified, Necklace solves a bin packing problem considering both chain priorities and additional physical constraints. As multiple chains for multiple tenants may be simultaneously requested, the SFC instantiation solver needs to invoke the appropriate policies on each individual request. We model this policy enforcement in this last phase. Each resource type may have its own binding policy ($e.g.$, it could follow either a guaranteed 99.999% Service Level Agreement model or a best-effort allocation). This phase only ensures that chain requests will be unable to exceed physical limits or their authorized resource usage. The weight w_j assigned to each chain request j, represent the policy used ($e.g.$ in first-come first-serve, $w_j = w \; \forall \; j$), or the priority or importance of allocating chain j for the system or application. For example, the system may assign null weight to a request that has not yet been authorized, even though the resources to map it exist.

Similarly to a standard set packing problem [39], we model the resource binding phase as follows:

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{j \in J} w_j x_j \\
\text{subject to} \quad & \sum_{j \in J} n^V_{ij} x_j \leq C^n_i, \quad \forall i \in V'_{G_j}, \\
& \sum_{j \in J} l_{kj} x_j \leq C^l_k, \quad \forall k \in \mathcal{P}'_j, \\
& x_j \in \{0, 1\}, \quad \forall j,
\end{aligned}
\tag{3}
$$

where C^n_i and C^l_k represent the capacities, $i.e.$ the number of virtual nodes and links, respectively, that can be simultaneously hosted on the hosting node i and physical path k, respectively. The binary variable x_j instead is equal to 1 if chain j has been successfully bind and zero otherwise. Note that it is possible for a chain request to be denied given the scarce resource or its lowest priority.

$$
\begin{aligned}
\underset{n^P, p, n^V, l, x}{\text{maximize}} \quad & \mathbb{E}\left[\sum_{j \in J} \sum_{i \in V_H} U_i(n^P_{ij}, p_{kj}, n^V_{ij}, l_{kj}, x_j)\right] \\
\text{subject to} \quad & \sum_{i \in V_G} \sum_{j \in J} n^P_{ij} - \sum_{j \in J} A_{Nj} \leq 0, \tag{4a} \\
& \sum_{k \in \mathcal{P}} \sum_{j \in J} p_{kj} - \sum_{j \in J} A_{\mathcal{P}_j} \leq 0, \tag{4b} \\
& \sum_{i \in V'_{G_j}} n^V_{ij} - \rho_j = 0 \;\; \forall j, \tag{4c} \\
& \sum_{k \in \mathcal{P}'_j} l_{kj} - \gamma_j = 0 \;\; \forall j, \tag{4d} \\
& n^V_{ij} - n^P_{ij} \leq 0 \;\; \forall i \;\; \forall j, \tag{4e} \\
& l_{kj} - p_{kj} \leq 0 \;\; \forall k \;\; \forall j, \tag{4f} \\
& \sum_{j \in J} n^V_{ij} x_j - C^n_i \leq 0 \;\; \forall i, \tag{4g} \\
& \sum_{j \in J} l_{kj} x_j - C^l_k \leq 0 \;\; \forall k, \tag{4h} \\
& x_j - \frac{1}{\rho_j} \sum_{i \in N_p} n^V_{ij} \leq 0 \;\; \forall j, \tag{4i} \\
& x_j - \frac{1}{\gamma_j} \sum_{k \in \mathcal{P}} l_{kj} \leq 0 \;\; \forall j, \tag{4j} \\
& x_j, n^P_{ij}, p_{kj}, n^V_{ij}, l_{kj}, \in [0, 1] \;\; \forall \; i, j, k. \tag{4k}
\end{aligned}
$$

D. Modeling chain subproblems interactions

Building on previous optimization problems, we now formulate a unified centralized stochastic optimization problem that considers the various aspects of the SFC instantiation problem. The optimization problem also provides insights on the interactions among each phase and how they may impact efficiency in network virtualization. In formulating a unified centralized stochastic optimization problem, we assume that processes cooperate to instantiate as many chain as the infrastructure can host seeking a Pareto optimality. Informally, a chain instantiation is optimal if no infrastructure provider hosting at least a virtual node has a better utility with another mapping. We give more details on the utility function in Section V. The optimization problem is stochastic since we

assume failures. This means that the Pareto optimal chain instantiation is not always achievable, due to suboptimality in the greedy leader election protocol, and due to system and link (temporarily) failures. In particular, we model the three phases of the *complete resilient chain instantiation problem* in Problem (4).

We model with n_{ij}^P (or p_{kj}) the probability of hosting node i (or hosting path k) being available after a state retrieval operation; n_{ij}^V represents the probability that a virtual instance of node i is selected as a valid mapping solution relative to the chain request j, while l_{kj} represents the probability that a virtual instance of the path k is selected to be hosting a virtual link within chain request j. Finally, variable x_j represents the probability of chain j being successfully allocated on the hosting infrastructure; C_i^n and C_k^l represent the hosting capacity vectors, *i.e.* the number of virtual nodes and links, respectively, that can be simultaneously hosted on node i and physical path k, respectively. ρ_j is the number of virtual nodes, and γ_j the number of virtual links, respectively, requested within chain j. Constraints (4a) and (4b) represent the physical state retrieval constraints, (4c) and (4d) refer to the mapping subproblem, while (4g) and (4h) describe the binding phase. Constraints (4i) and (4j) instead capture the final binding subproblem: these constraints ensure that each chain request is not considered for binding unless all virtual nodes and all virtual links requested in the path can be (if no failures occurs) successfully mapped ($n_{ij}^V = 1$ and $l_{kj} = 1$). In optimization theory, constraints like (4e), (4f), (4i) and (4j) are often called *complicating constraints*, as they "complicate" the problem binding the three mechanisms together; without those constraints, each of the chain instantiation mechanism could be solved independently from the other two, *e.g.*, by a different architecture component.

IV. NECKLACE PROTOTYPE ARCHITECTURE

In this section we introduce the architecture of our Necklace system, as depicted in Figure 1. Being located between the network operating system and the application, Necklace manages all chain instantiation states, shared across federated domains, via a novel management protocol that we call CIP (Chain Instantiation Protocol.) The message design of CIP is not only used to implement the consensus strategy used during the crucial chain mapping phase, but to manage the inter-process communication among Necklace processes. CIP is inspired by classical network management protocols [40]–[42]; it is simpler, and so less general than CMIP [42] or HEMS [41], but more complex than SNMP [40], and specific to the service chain instantiation problem.

The core components of Necklace refer to the three coupled mechanisms necessary and sufficient to instantiate a (resilient) chain, as described in Section III: state retrieval (or resource discovery), chain mapping and resource binding. All other architecture components support such mechanisms along with their interactions, as well as the interface with the underlying network operating system and the application policy or *intent* requests. In the rest of this section we describe the building blocks of Necklace (Figure 1).
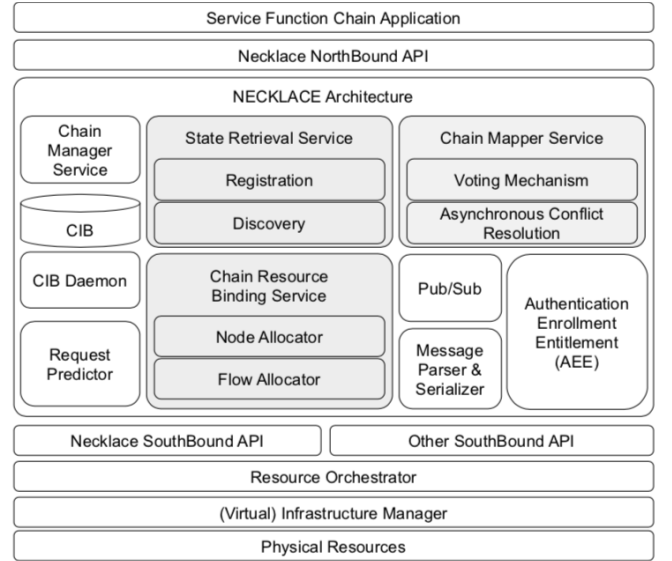

Fig. 1: Necklace architecture overview: the core components support the three coupled mechanisms for a resilient Service Function Chain Instantiation, and their interactions with the network operating system and the applications.

Management Object Model. Historically [40]–[42], a management object model has been characterized by a (i) set of objects along with their attributes, to define the manageable states, (ii) an interface to modify such object attributes locally, and a (iii) set of protocol messages, to modify attributes remotely. Our approach to define a chain instantiation management object model was no exception. Every process participating in the distributed chain instantiation protocol stores all relevant states locally into its *Chain Information Base (CIB)*. The CIB is similar in design to the Forward Information Base (FIB), a Routing Information Base (RIB) or a more general Management Information Base (MIB). Instead of storing, *e.g.* routing states as in a RIB, the CIB stores in a partially replicated database all the states necessary to instantiate a chain. The *Chain Information Base Daemon* task is to keep the CIB consistent across instances belonging to the same network participating to the chain instantiation process. The daemon works together with the *message parser and serializer*, an interpreter for the CIP protocol that uses Google Protocol Buffers [43]. We describe in details the distributed consensus chain instantiation protocol (implemented using the CIP management protocol messages) in Section V.

Necklace was designed and built to orchestrate virtual function chains, but its architecture is extendable to other virtual (network) function management mechanisms by merely extending the object model. New objects would have to be then managed by the *Chain Instantiation Protocol (CIP)* and by the application logic. To define our objects, our protocol implementation uses the Google Protocol Buffers [43] (GPB) as abstract syntax notation, and JSON for our OpenSource Mano (OSM) deployment, where processing efficiency is not as crucial. Many systems today use XML to define policies, intents or objects, but XML is a text-based syntax notation, and so much less efficient than binary-based alternative like GPB and BSON. BSON was also a very good option but the compiler (that converts string objects into binary and API)
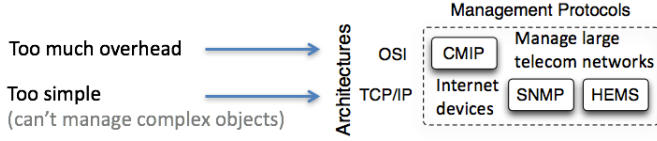
Fig. 2: Existing network management protocols are suboptimal for virtual network function chain management given their low expressiveness or their high overhead.

is yet unavailable; also, protocols written in GPB appear to be easily extendable. An abstract syntax notation is needed to serialize and de-serialize messages in a protocol. This is needed so that a system prototype is both language-neutral and platform-neutral, that is, different processes may implement our protocol to host chain nodes using *e.g.*, Java, C++, Python, on machines whose processor architectures are little-endian or big-endian, without needing additional code refactoring.

Chain Manager Service. In our Necklace architecture, this component is similar to a classical Network Management System (NMS), and it is responsible for the monitoring, naming and addressing resolution mechanisms. By monitoring, we mean the ability to parametrize and interpret keep-alive messages. The Chain Manager Service orchestrates also the authentication, enrollment and entitlement mechanisms, that we represent in a separate block in Figure 1.

Authentication, Entitlement and Enrollment. Before a chain can be instantiated, the processes participating in the leader election protocol need to authenticate. To do so, they enroll in an overlay with private addressing scheme, and subscribe to each-other updates. The Authentication, Entitlement and Enrollment (AEE) component deals with such authorizations mechanisms. Currently, the architecture only supports a basic user and password authentication, but we modularized this block to allow independent evolution.

In the upcoming sections we describe the distributed asynchronous max-consensus, core mechanism of the Necklace architecture, used as a chain mapping mechanism. We then describe how, under realistic assumptions, it guarantees performance and convergence even in the presence of non-byzantine failures during a chain instantiation process.

Why another network management protocol? As shown in Figure 2, existing management protocols such as CMIP [42] or HEMS [44] could have been used as well, but they are either too complex for merely a chain instantiation, and so they would bring unnecessary complexity and overhead, or, like SMNP [40], are not expressive enough and they do not support authentication (that our CIP protocol enrollment phase provides).

V. DISTRIBUTED ASYNCHRONOUS CHAIN CONSENSUS ALGORITHM (DACCA)

In this section we describe our proposed chain mapping mechanism *viz. Distributed Asynchronous Chain Consensus Algorithm* (DACCA). The mechanism's goal is to reach in a distributed fashion an agreement on which network resource will host the chain, weather such resources are controlled by a single, or by multiple infrastructure providers. The main idea

behind the chain mapping procedure is to have (federated) hosting nodes independently run an election process using a private utility function, that we formally define later in this section. Such utility is a policy, *i.e.*, a variant aspect of the invariant mapping mechanism. Although our Necklace architecture supports any utility function, we give recommendations on which utility processes running DACCA should use to obtain bounds on convergence time and guarantees with respect to the Pareto optimal chain instantiation.

DACCA overview. Consider a chain request by an application or a service provider, where each potentially hosting node belongs to a different infrastructure provider. The service provider may send to (a subset of) all hosting nodes a request for the entire chain, or, if it has preferences, it may split the request into multiple contiguous subsets. Each potentially hosting node receiving the request then uses a private function to decide what is its utility in hosting the chain (subset). After a first asynchronous voting procedure (in which each host tries if possible to elect himself as a leader or remains silent), each voter exchanges its utility values (votes) *only* with its first-hop neighbors, for a distributed leader election. An asynchronous conflict resolution phase is then run to propagate only the highest utility on each chain element, considering also times at which the votes were generated (as opposed to receiving time). In our system implementation each hosting process runs a standard time synchronization protocol (TSP) set with a third party unique server. DACCA can simultaneously elect multiple leaders, maximizing the sum of the utilities (Pareto optimality) of the hosting nodes. The elected leaders communicate the mapping to the service provider that, if possible, releases the next chain request, if any, or the next virtual node of the service chain.

Before describing DACCA in details, we need few definitions that we will use in the rest of the paper:

Definition 1. *(utility function $\mathbf{f}_i$). Given a chain H to be instantiated on a hosting network G with a voting procedure among $|V_G|$ hosting nodes, we define utility function of voter i, and we denote it with $\mathbf{f}_i \in \mathbb{R}_+^{|V_H|}$, the utility that hosting node i assigns to chain node j during the chain mapping phase. Notation $\mathbb{R}_+^{|V_H|}$ represents a vector of positive real numbers with size $|V_H|$.*

The value assumed by the utility function eventually becomes a vote. One example of f_{ij} is the residual capacity (stress profile) on the hosting node, that we define as:

$$f_{ij} = \frac{(T_i - S_{ij})}{\delta + \sum_{i \in \mathcal{N}} S_{ij}} \cdot \frac{1}{\mathbb{T}_i} \qquad (5)$$

where δ is a small positive constant, $\frac{1}{\mathbb{T}_i}$ is a normalization constant, S_{ij} represents the stress on the host node i, namely, the sum of the chain node (CPU) capacity already allocated on the i, including chain node j; $\mathbb{T}_i$ is the desired load on host node i. A service chain provider may not necessarily want to balance the load: *e.g.*, it may be desirable to force mapping on data center nodes where energy is cheaper. Note how, due to the normalization factor $\mathbb{T}_i$, in this particular case $f_{ij} \in [0, 1]$.

Definition 2. *(vote vector $\mathbf{v}_i$). Given a chain H to be instantiated on a hosting network G with a leader election process among $|V_G|$ hosting nodes, we define as vote vector $\mathbf{v}_i \in \mathbb{R}_+^{|V_H|}$, where $i \in \mathcal{I} \triangleq \{1,\ldots,|V_G|\}$, the vector of current winning votes on chain nodes, i.e., each element v_{ij} is a positive real number representing the highest vote known by hosting node i, made so far on chain node $j \in \mathcal{J} \triangleq \{1,\ldots,|V_H|\}$.*

Note that even though $\mathbf{v}_i$ could contain votes for at most $|V_H|$ virtual nodes, we leave its maximum length as a policy.

Until an agreement on the chain instantiation is reached, DACCA iterates multiple voting and consensus phases asynchronously, in what we call a round. Hosting nodes act upon messages received at different time during each consensus phase, and messages may arrive out of order. In the rest of our paper, we denote those *rounds or iterations* with t.

Definition 3. *(eligible resource indicator $\mathbf{h_i}$). Given a chain instantiation of H on a hosting network G with a multi-election process among $|V_G|$ hosting nodes, a private utility function f_{ij} and a vote v_{ij} of hosting node i on chain element j, we define $\mathbf{h_i}(t) = (h_{ij}(t) \mid h_{ij}(t) = \mathbb{I}(f_{ij}(t) > v_{ij}(t)) \ \forall j \in V_H)$, that is, the list of chain elements eligible to receive votes. $\mathbb{I}(\cdot)$ is the indicator function that is unitary if the argument is true and zero otherwise.*

Definition 4. *(assignment vector $\mathbf{a}_i$). Given a set of processes V_G voting on a set of chain elements V_H, we define the assignment vector $\mathbf{a}_i \in V_G^{|V_H|}$, as the vector containing the latest information that $i \in V_G$ has on the current assignment of all chain elements.*

Depending on the level of information that it contains, the assignment vector $\mathbf{a}_i$ may assume two different policies — *single or least informative* and *multiple, or most informative*. In its least informative form, $\mathbf{a}_i$ is a vector storing the probability that process i hosts chain element j. In its most informative form, $\mathbf{a}_i$ is the vector of winning voters so far, *i.e.*, each element represents the identifier of the hosting process that has the highest utility so far to host element j.

When $\mathbf{a}_i$ is in its most informative policy form, the vector gives to a hosting process i information on which is the winner of the leader election, where in its least informative policy, process i only knows if it is the winner of a resource j or not. In the evaluation section we show how the assignment vector policy generates an interesting tradeoff analysis.

Definition 5. *(bundle vector $\mathbf{m}_i(t)$). Given a process $i \in V_G$ voting on a set of chain elements V_H, and a bundle target size $\mathbb{T}_i$ that is, the maximum number of elements that i is require or willing to host, we define the bundle vector $\mathbf{m}_i(t) \in V_H^{\mathbb{T}_i}$ to be the list of chain element identifiers that hosting node i is currently voting on during iteration t. We use $\mathbb{C}(\mathbf{m}_i(t))$ to denote the cardinality of $\mathbf{m}_i(t)$, i.e., the number of chain nodes currently in the bundle $\mathbf{m}_i$ at iteration t.*

Note how both the bundle target size $\mathbb{T}_i$ and $\mathbb{C}(\mathbf{m_i}(t))$ can be expressed either in terms of total number of virtual nodes, *e.g.*, $\mathbb{C}(\mathbf{m_i}(t)) = |\mathbf{m_i}(t)|$ or in terms of sum of their requested

---

Algorithm 1: DACCA for process i at iteration t

1: **for all** $k \in \mathcal{N}_i$ **do**
2:    `voting`$(\mathbf{a}_i(t-1), \mathbf{v}_i(t-1), \mathbf{m}_i(t-1))$
3:    `send`(k,t) $\forall k \in \mathcal{N}_i$ // vote message for round t
4:    `receive`(k,t) $\forall k \in \mathcal{N}_i$
5:    `agreement`(k,t)
6: **end for**

Algorithm 2: `voting` for process i at iteration t

1: **Input:** $\mathbf{a}_i(t-1)$, $\mathbf{v}_i(t-1)$, $\mathbf{m}_i(t-1)$,
2: **Output:** $\mathbf{a}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{m}_i(t)$,
3: $\mathbf{a}_i(t) = \mathbf{a}_i(t-1)$
4: $\mathbf{v}_i(t) = \mathbf{v}_i(t-1)$
5: $\mathbf{m}_i(t) = \mathbf{m}_i(t-1)$
6: **while** $(\mathbb{C}(\mathbf{m}_i(t)) < \mathbb{T}_i)$ **do**
7:    **if** $\mathbf{h}_i \neq 0$ **then**
8:      `vote`$(\mathbf{a}_i(t), \mathbf{v}_i(t), \mathbf{m}_i(t))$
9:    **end if**
10: **end while**

---

capacity: $\mathbb{C}(\mathbf{m_i}(t)) = \sum_{j \in \mathbf{m}_i(t)} C_{ij}(t)$, where $C_{ij}(t)$ is the requested capacity of virtual node j to hosting node i at iteration t.

We are now ready to describe the consensus-based distributed chain mapping mechanism: each hosting node $i \in V_G$ performs a voting phase; then, the vote vector $\mathbf{v}_i$ and, when specified by the policy, the allocation vector $\mathbf{a}_i$ are exchanged among neighboring nodes for a distributed consensus-baesd winner determination (Algorithm 1).

A. Phase 1: DACCA Voting Phase

After the initialization of both vectors $\mathbf{a}_i$, $\mathbf{v}_i$ and $\mathbf{m}_i$ to the current iteration t (Algorithm 2, lines 3-5), each potentially hosting node checks its current available capacity $\mathbb{C}(\mathbf{m_i}(t))$, given what it already has in its bundle, to verify if the target capacity has been reached. Note that this does not mean that there is room to host another chain node. In fact, hosting node i may have still residual capacity, but not to host any chain node in the current round. Assume for example that $\mathbb{C}(\mathbf{m}_i(t)) = \mathbb{T}_i - \epsilon$, where $\epsilon > 0$ and that the requested capacity $C_j > \epsilon$ for every virtual node j still to be mapped. If not, the voting phase terminates (line 6), otherwise the hosting node verifies if it can vote higher than some other hosting node. If there is at least an acquirable chain node (*i.e.*, if $\mathbf{h_i} \neq 0$), the function `vote`$(\cdot)$ registers a vote for the chain node whose reward is the highest in the vector $\mathbf{v}_i$, updates the assignment vector $\mathbf{a}_i$ with itself (node i) as a winner of that virtual node, and finally, the chain node's identifier is *appended* to the bundle vector *i.e.*, $\mathbf{m}_i \leftarrow \mathbf{m}_i + j$. At the end of this phase, the current winning vector $\mathbf{v}_i$ and, when the policy allows it, the assignment vector $\mathbf{a}_i$ are exchanged with each neighbor $\mathcal{N}_i$.

Remark. *Note that bundle $\mathbf{m}_i$ is an ordered list where the order represent the preference of hosting node i. If multiple chain nodes are allowed to be elected simultaneously, a vote on the highest rewarding chain node is inserted first in $\mathbf{m}_i$, while subsequent chain nodes are assigned with the new value*

Algorithm 3: `agreementPhase` of i at iteration t

1: **Input:** $\mathbf{a}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{m}_i(t)$
2: **Output:** $\mathbf{a}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{m}_i(t)$
3: **for all** $k \in \mathcal{N}_i$ **do**
4:   **for all** $j \in V_H$ **do**
5:     **if** `IsUpdated`(v_{kj}) **then**
6:       `update`$(\mathbf{a}_i, \mathbf{v}_i, \mathbf{m}_i)$
7:     **end if**
8:   **end for**
9: **end for**

---

of utility recomputed assuming that former entries in the bundle will be won (elected). If we use the residual capacity as utility (and voting function), this means that hosting nodes are only allowed to vote using their residual capacity to acquire subsequent resources. This in turn means that their votes is a diminishing marginal gain function. A relaxed version of this diminishing marginal gain property is a key notion that we use to show bounds on optimality in the next section.

B. Phase 2: DACCA Consensus

In this phase hosting nodes make use of a max-consensus strategy to converge to the winning vote vector $\bar{\mathbf{b}}$, and to compute the allocation vector $\bar{\mathbf{a}}$ (Algorithm 3).

The standard definition of *max-consensus* [45], applied to the chain instantiation problem becomes:

Definition 6. *(max-consensus). Given a hosting network G, an initial vector of hosting nodes $\mathbf{v}(0) := (\mathbf{v}_1(0), \dots, \mathbf{v}_{|V_G|}(0))^T$, and the consensus algorithm for the communication instance $t + 1$*

$$\mathbf{v}_i(t+1) = \max_{j \in \mathcal{N}_i \cup \{i\}} \{\mathbf{v}_j(t)\} \;\; \forall i \in \mathcal{I}, \tag{6}$$

max-consensus *among the hosting nodes is said to be achieved if $\exists\, l \in \mathbb{N}$ such that $\forall t \geq l$ and $\forall i, i' \in \mathcal{I}$,*

$$\mathbf{v}_i(t) = \mathbf{v}_{i'}(t) = \max\{\mathbf{v}_1(0), \dots, \mathbf{v}_{|V_G|}(0)\}. \tag{7}$$

The agreement (or consensus) for each hosting node i, for example on the vector $\mathbf{v}_i$ received from each hosting node k in the neighborhood of i, is performed comparing v_{ij} with v_{kj} for all k members of $\mathcal{N}_i$. This evaluation is performed by the function `IsUpdated`$(\cdot)$ (line 5). In case the policy requires consensus only on a single chain node at the time *i.e.* $|\mathbf{m}_i| = 1$, the function `IsUpdated`$(\cdot)$ returns always true, since when a hosting nodes i receives from k an higher utility for a chain node j $(v_{ij} < v_{kj})$, the receiver hosting node is always required to update its vote vector $\mathbf{v}_i$ $(v_{ij} \leftarrow v_{kj})$. When instead hosting nodes are allowed to vote on multiple chain nodes in the same election round —the size of the bundle $|\mathbf{m}_i| > 1$, even if the received utility for a chain node is higher than what is currently known, the information received may not be up to date. In other words, the standard max-consensus strategy may not work. Each hosting node is in fact required to evaluate the function `IsUpdated` that compares the time-stamps of the received vote and updates the bundle, the utility

and the assignment vector accordingly (Algorithm 3 line 6). Note that a requirement of the DACCA protocol is to forbid hosting nodes to vote (for itself) again after it has received a valid higher utility from another hosting node. Malicious nodes may abuse of this feature to attack the protocol; we leave the question on how to secure a correct functionality of DACCA even in the presence of byzantine failures open and outside the scope of this work.

Remark: *If a hosting node i received a more recent higher vote for chain node j, deleting node j from its bundle $\mathbf{m}_i$ is not enough: all utilities and winners of subsequent nodes in the bundle (built appending subsequent allocation attempts) need to be released as they were obtained using an out of date utility (e.g., the residual capacity was out-of-date.)*

Note also that our protocol does not violate the FLP impossibility result [46] (no consensus can be guaranteed in an asynchronous communication system in the presence of any failures). Our assumption is that our asynchronous consensus is achieved among the processes that are participating before a tunable timeout, started after receiving the first vote.

C. Pseudo sub-modular utility functions

As we will see in Section VI, our DACCA mechanism guarantees convergence allowing hosting nodes to use their own utility function as a private policy, as long as the function *appears* to be sub-modular to other bidders [47]. Sub-modularity is a well studied concept in mathematics [48], and applied to the distributed chain instantiation problem, can be defined as follows:

Definition 7. *(sub-modular function.) The marginal utility function $U(j, \mathbf{m})$ obtained by adding a virtual resource j to an existing bundle $\mathbf{m}$, is sub-modular if and only if*

$$U(j, \mathbf{m}') \geq U(j, \mathbf{m}) \;\forall\; \mathbf{m}' \mid \mathbf{m}' \subset \mathbf{m}. \tag{8}$$

This means that if a hosting node uses a sub-modular utility function, a value of a particular virtual resource j cannot increase because of the presence of other chain nodes in the bundle.

Why do we need sub-modular functions? Consider two hosting nodes PN1 and PN2 trying to instantiate two chain nodes C1 and C2 using (the same) sub-modular utility function to vote. In the first voting phase, the bundle $\mathbf{m}$ and voting vector $\mathbf{v}$ for PN1 and PN2 are:

$$\mathbf{m}_{PN1} = (C1, C2), \;\; \mathbf{v}_{PN1} = (v, v - \epsilon)$$
$$\mathbf{m}_{PN2} = (C2, C1), \;\; \mathbf{v}_{PN2} = (v, v - \epsilon).$$

After the consensus phase, PN1 releases C2 and PN2 releases C1 as they are outbid and we have a converge to an assignment. If instead PN1 and PN2 use a non sub-modular function, after the first voting phase they could end up with the following bundle and bid vectors:

$$\mathbf{m}_{PN1} = (C1, C2), \;\; \mathbf{v}_{PN1} = (v, v + \epsilon)$$

$$\mathbf{m}_{PN2} = (C2, C1), \quad \mathbf{v}_{PN2} = (v, v + \epsilon).$$

After exchanging their bid vectors, the agreement phase would require PN1 and PN2 to reset their bundle, and a second voting phase would bring exactly the same initial case. This cycle would repeat forever breaking convergence.

Although having sub-modular utility functions may be realistic in many resource allocation problems [49], in the distributed chain instantiation problem this assumption may be too restrictive, as the value of a chain node may increase as new resources are added to the bundle, *e.g.* the cost of mapping a directed virtual link between two virtual nodes part of the chain decreases if a hosting node is elected leader on both virtual source and destination.

For this reason, we use the notion of *pseudo sub-modularity*, that is, each hosting node may use any utility function, as long as the bids communicated to the other bidders appear as if they were obtained using a sub-modular function.

To guarantee convergence without using a sub-modular utility function, we let each hosting node communicate its vote on virtual node j obtained from a vote "bending" function:

$$\mathcal{B}_{ij}(U_{ij}, \mathbf{v}_i) = \min_{z \in \{1, \dots, |\mathbf{v}_i|\}} \{\mathcal{B}_{iz}, U_{ij}\}, \tag{9}$$

where $\mathcal{B}_{iz}$ is the value of the bending function for the zth element of $\mathbf{v}_i$. Note how by definition, applying the function $\mathcal{B}$ to the vote before sending it is equivalent to communicating a bid that is never higher than any previously communicated bids. In other words, bids *appear* to other hosting nodes to be obtained from a sub-modular utility function.

D. Contiguous Virtual Path Mapping Policy

In this section, we discuss the problem of a chain mapping request that result in logical allocation of middleboxes that are physically far away. When a chain of virtual network functions is instantiated in a distributed fashion, hosting nodes located on hubs *i.e.*, highly central node, may quickly congest the entire hosting network. Moreover, if the chain request is mapped on a path that would include physical nodes not hosting any virtual node, two problems may arise: (i) the hosting network will map virtual links on hosting paths that are unnecessarily long, with a consequent over-provisioning, additional delays or energy waste; (ii) perhaps more importantly, in a federated chain instantiation, virtual link requests may require to be hosted between non-neighboring hosting nodes belonging to different providers, expecting intermediate hosts to relay data traffic. This may be acceptable for some applications or merely undesirable for others.

Necklace handles this potential problem by supporting *virtual path auction* policies. Such policyallows processes to instantiate a chain in a distributed mapping, by avoiding relays, *i.e.*, attempting to host either contiguous or non-contiguous virtual paths. By contiguous virtual path we mean that neighboring chain nodes are mapped to neighboring (or identical) hosting nodes. This means that each virtual link is allocated on a single (physical) link, as opposed to being allocated on any generally longer hosting path. This path restriction obviously forces additional pruning but may save infrastructure providers additional costs, for example, those arising when we leave the mapping decision to a more classical k-shortest path. During the voting phase, hosting nodes applying the *contiguous virtual path* policy are allowed to attempt hosting a chain node j only if the chain nodes adjacent to j are currently mapped by the node itself, or by an adjacent hosting node. By enforcing the *contiguous virtual path* policy, a chain of length $L > 0$ will be mapped on physical paths of length at most L, avoiding node relays.

E. Handling Failures

In this subsection we analyze the resiliency rationale behind Necklace. In particular, we describe how Necklace considers and handles node and link failures.

Necklace is based on the DACCA mechanism, an asynchronous max-consensus protocol. Fischer, Lynch and Paterson showed that it is impossible to achieve asynchronous consensus within a system with failing processes. (FLP impossibility result [50]). Many practical solutions have been proposed to cope with the FLP impossibility result: from using randomized algorithms to failure detectors [45], to name a few; we employ an engineering approach and merely adopt a timeout to forcefully terminate the asynchronous max-consensus even if we have not received the minimum number of messages to ensure that we have converged to a solution; this is to avoid having to wait indefinitely for messages that are delayed by lossy or congested paths. The DACCA timeout could be estimated using an average of round-trip-time values across the overlay, similarly to how TCP does it [51], or it could be dynamically set by the SDN controller; in our prototype, however, we statically set it from with a configuration file. In DACCA, new votes received from first-hop neighbors propagate hop by hop traversing the entire (virtual) network overlay. When a resource cannot be outvoted, nodes remain silent. This means that by merely consider application-level states, DACCA nodes hence cannot distinguish between silent and unavailable nodes. This is an advantage as it means that the mechanism is resilient to silent (unavailable) nodes.

Note how timing-out in a fully distributed max-consensus approach as DACCA does not require majority consensus; this is different than Paxos-like protocols [12], [15], [16] or Raft [14] that need both a leader and a majority of node participating to the consensus, *i.e.*, they are intolerant to failures of the majority of nodes.

F. Traffic Forecast to Provision NFV Chain Requests

Real system measurements can be exploited to extract knowledge about future traffic patterns. These insights can in turn be utilized to effectively provision the underlying infrastructure resources and accommodate all NFV chain requests. A responsive and effective network management requires a sharp decision-making process, especially in dynamic settings. An accurate estimation of the incoming future traffic is valuable to provide such responsiveness within dynamic SFC provisioning, to optimize node utilization and link capacities. Traffic volume awareness allows a responsive proactive provisioning approach. To this aim the *Request Predictor* block of our architecture deals with traffic forecasts. We validated a few traffic prediction policies over a real dataset of requests [18] in Section VII.

VI. Convergence and Performance Guarantees

In this section we show results on the convergence properties of the DACCA mechanism adopted by Necklace. By convergence we mean that a valid mapping (Section V-B) is found in a finite number of steps (Definition 6). Moreover, leveraging well-known results on sub-modular functions [48], [52], we show that under the assumption of pseudo sub-modularity of the utility function, DACCA guarantees an optimal $(1 - \frac{1}{e})$-approximation, that is, a better approximation does not exist unless $P = NP$.

Convergence Analysis. A necessary condition for convergence of the max-consensus is that, to make all hosting nodes aware of what is the node that has the highest (maximum) utility on each single chain node, this information needs to traverse all the physical network, which we assume has diameter D. Our convergence results (Theorem VI.1) states that, in absence of failures, this single hosting network traversing is also sufficient. This claim is inspired by Theorem 1 in [53], which deals with a distributed task allocation problem for a fleet of robots. We relax, however, their Diminishing Marginal Gain (DMG) property in [53] since in our problem a hosting node utility does not depend on the order of insertion of chain nodes in the bundle, while ordering is crucial in the mission allocation problem in robotic networks.

Theorem VI.1. *(Convergence of synchronous DACCA). Given a chain of virtual network functions H of length $|V_H|$ and a hosting network with diameter D, the utility function of each hosting node is pseudo sub-modular, and the communications occur over reliable channels, then the DACCA mechanism converges in a number of iterations bounded above by $D \cdot |V_H|$.*

Proof. (sketch) We use $\mathcal{B}_{ij}(U_{ij}, \mathbf{v}_i)$ as a voting function (pseudo sub-modular by definition). If a vote is generated with a pseudo sub-modular function, then it appears to be DMG and so sub-modular to other processes. From [53] we know that by induction, a consensus-based auction run by a fleet of N_u processes, each assigned at most T tasks, so as to allocate N_t tasks, converges in at most $N_{min} \cdot D$ where $N_{min} = \min\{N_t, N_u \cdot T\}$. Since for DACCA to converge, every chain node needs to be assigned, $N_{min} = N_t \equiv |V_H|$, and therefore we have the claim. $\square$

As a direct corollary of Proposition VI.1, we compute a bound on the number of messages that hosting nodes have to exchange in order to reach an agreement on a chain instantiation. Because we only need to traverse the hosting network once for each node, the following result holds:

Corollary VI.1. *(DACCA Communication Overhead) The number of messages exchanged to reach an agreement with reliable channels and non-failing hosts using the DACCA mechanisms is at most $D \cdot |S_G| \cdot |V_H|$, where D is the diameter of the hosting network, $|S_G|$ is the number of edges in the hosting network minimum spanning tree, and $|V_H|$ is the length of the chain.*

Performance Guarantees. The following results holds:

Theorem VI.2. *(Asynchronous DACCA Approximation). The DACCA consensus algorithm yields an $(1 - \frac{1}{e})$-approximation (circa 0.63) with respect to the optimal chain assignment.*

Proof. The DACCA asynchronous consensus algorithm assumes that each hosting node i does not vote on a chain node j unless it brings a positive utility, therefore U_{ij} and so $\mathcal{B}_{ij}$ are positive. Moreover, if we append to the vector $\mathbf{v}_i$ an additional set of chain nodes $\mathbf{v}$ resulting in vector $\mathbf{v}'_i$, we have:

$$\mathcal{B}_{ij}(U_{ij}, \mathbf{v}'_i) \le \mathcal{B}_{ij}(U_{ij}, \mathbf{v}_i) \ \forall \ \mathbf{v} \neq \emptyset \qquad (10)$$

which means that $\mathcal{B}_{ij}$ is monotonically non-increasing.

Since the sum of the utilities of each hosting node, and since the bending function $\mathcal{B}_{ij}(U_{ij}, \mathbf{v}_i)$ of DACCA is a positive, monotone (non-increasing) and sub-modular function, theorem 1 in Nemhauser *et al.* [48] on sub-modular functions holds. Therefore the claim holds. $\square$

Moreover, the following approximation bound holds:

Theorem VI.3. *(Approximation Bound). The DACCA approximation bound of $(1 - \frac{1}{e})$ is optimal, unless $P = NP$.*

Proof. To show that the optimal chain instantiation cannot be approximated in polynomial time within a ratio of $(1 - \frac{1}{e} - \epsilon) \ \forall \ \epsilon > 0$, we use a recent result by Feige [52]. The result shows that it is NP-hard to achieve a $(1 - \frac{1}{e} + \epsilon)$-approximation $\forall \epsilon > 0$ for the *maximum k-coverage* problem [54]. Given m subsets $V_1, \ldots, V_m$ of V and k processes with different weight functions $U_i : V \rightarrow \mathbb{R}_+$, the maximum k-coverage is the problem of allocating each set V_j to some process i, in order to maximize $\sum_{i=1}^{k} U_i(\bigcup_{j \in S_i} V_j)$, where S_i are the indices of sets allocated to hosting node i. We reduce the DACCA assignment problem from the maximum k-coverage problem by considering $V_1, \ldots, V_m$ to be subsets of bundles that any k hosting node wins according to their voting function. Note that all final assigned bundles are necessarily disjoint by definition of consensus, *i.e.*, there cannot be two hosting nodes for each node of the chain. The maximum k-coverage is a special case of the *maximum coverage problem* for monotone sub-modular functions, a problem for which the approximation bound for the greedy heuristic was proven [48]. As the DACCA consensus strategy is a greedy heuristic that maximizes a monotone sub-modular function, the max-consensus greedy heuristic is the best approximation algorithm for the node mapping phase that we can possibly hope for, unless $P = NP$. $\square$

VII. Traffic Prediction Analysis

In this section, we evaluate the performance of the predictor used in our implementation, a specific module of our architecture. Since our system uses this predictor as a policy, we separate the evaluation of this component from the evaluation of the entire system (Section VIII).

We used a dataset composed by real traffic requests to the Facebook datacenter to assess the forecasting capabilities of several learning algorithms. This forecast is an input to our DACCA algorithm that runs chain allocation request prior to their potential arrival and pre-launches real physical instances, when needed.
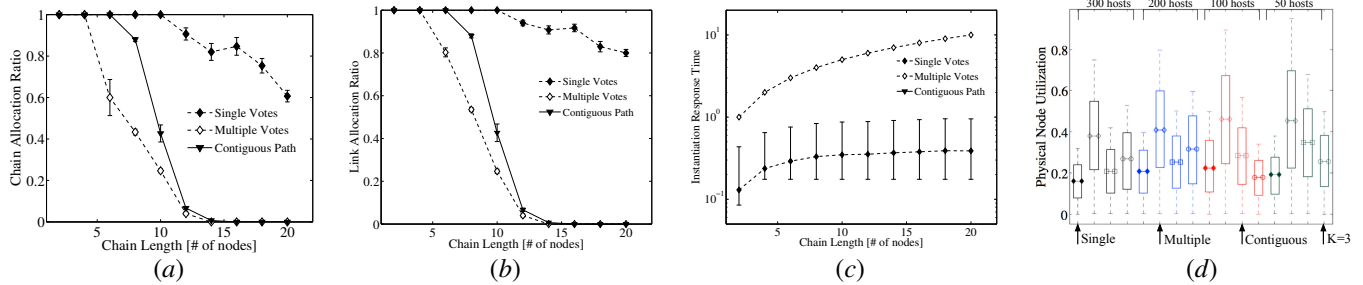
Fig. 3: (abc) Prototype. (d) Simulations. (a) Hosting single chain nodes helps balancing the load and hence accepting more chain. (b) Similar results were obtained at a larger scale in simulations. (c) Single allocation helps faster response time. (d) Host utilization tradeoffs across different hosting network sizes.
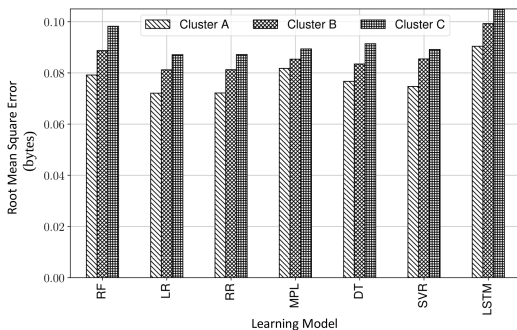

Fig. 4: **Forecasting capability (quantitative):** it is reported the RMSE of the predictions for all the 7 models tested (1-step ahead predictions).

Fig. 5: **Forecasting capability (qualitative):** predictions of the average packet length obtained using MLP, SVR, and LR over a period of 300 minutes.

Facebook dataset description. We used the data collected from the Facebook datacenter located in Altoona [55] to predict traffic size and hence requests. The dataset contains traffic from three separate clusters that refer to requests to Database, Web servers, and Hadoop.

Learning models. We tested seven forecast models applying a standard 70% train, 30% test ratio. All tested models and their acronyms are denoted as follows: Random Forest (RF), Linear Regressor (LR), Ridge Regressor (RR), Multi-layer Perceptron (MLP), Decision Tree (DT), Support Vector Regression (SVR) and Long Short-Term Memory (LSTM)-neural network. In Figure 4, we report the Root Mean Square Error (RMSE) obtained by the seven algorithms with a 1-step ahead prediction model. Note that signals' range is in $[0, 1]$: feature scaling is commonly applied to avoid issues during model training.

We observe that the normalized RMSE of all but LSTM is less than 0.1 showing that the maximum forecasting error is 10% of the real values. In Figure 5, we show the predicted and real values of the average packet length over the next minute, for a period of 300-minutes. Due to their similarities, we only show the three representative (and best performing) prediction algorithms: multilayer perceptron, support vector regressor, and linear regressor over a 300-minute period within Cluster-B.

VIII. EVALUATION

To test our proposed distributed chain instantiation protocol we developed our own event-driven simulator and we implemented a prototype of Necklace. We also assessed the deployment time of service chains on a real-hardware testbed using the OpenSource MANO (OSM) orchestrator.
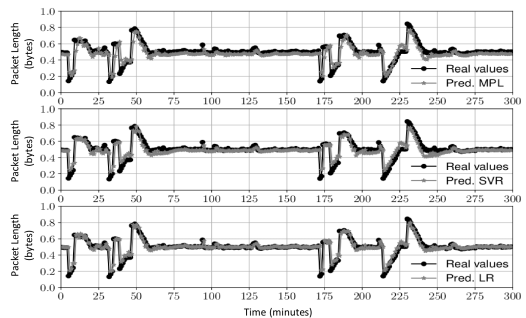
Hosting Network Model: With the BRITE topology generator [56], we obtain a topology that we use as hosting processes. We use the top-down generation model of BRITE which is based on two phases. In the first phase, an Autonomous-System (AS) topology is generated using the Barabasi-Albert model with incremental growth type and preferential connectivity. In the second phase, a node level topology is generated for each AS, where hosting nodes are placed randomly on the 2D-plane and connected using the Waxman model. The sizes of our chains (virtual network with a linear topology) were obtained from sampling a distribution uniformly distributed at random with an average 50 nodes. The physical network, synthetically generated with BRITE has 500 nodes.

We compared three policies of our DACCA's mechanism: (i) single if a single chain node is allowed to be mapped at a time (i.e., $|\mathbf{m}_i| = 1$), (ii) multiple when multiple chain nodes are allowed to be mapped simultaneously, i.e., $|\mathbf{m}_i| > 1$ (even if we have to create Generic Routing Encapsulation (GRE) tunnels among elected leaders), and (iii) *contiguous*, when relays are forbidden (see Section V-D). Unless otherwise stated, we used a k-shortest path with $k = 2$ to map virtual endpoints.

Chain Model: Even though a chain request may have several constraints, we focus our evaluation solely on CPU and bandwidth constraints. We synthetically generate requests for chain instantiations sampling uniform distributions ranging between 1 and 20 for the number of virtual CPU units, and between 1 and 50 Mbps for the requested bandwidth, respectively. We ensure that each chain node is able to support at least the capacity of its adjacent virtual links, and we impose ratios between chain-host node and link capacity of 200 and 500,

respectively.

Evaluation metrics. Our evaluation results quantify the benefits of our approach across two metrics: mapping efficiency and time to find a solution, and across two platforms: a simulator and our Necklace prototype tested over Mininet [17]. In particular, the time to find a solution is evaluated through the response time, namely, the number of one-hop communications that the DACCA protocol needs to realize a chain can or cannot be instantiated (since we found bounds on convergence time). Efficiency instead is evaluated through the chain allocation ratio within our prototype, namely, the ratio between the number of chains requested and those successfully instantiated, and with resource utilization in our simulations, namely, the hosting node capacity utilized in the mapping.

Evaluation Results. We attempt to map 100 chains with increasing lengths over a testbed of 50 (mininet) hosting nodes running a Necklace prototype (Figure 3a-c). We then repeat the same experiments with a simulated (more scalable) version of DACCA and evaluate the physical (hosting) network utilization. We present our results by summarizing the key observations. Moreover, results are shown with a 95% confidence interval.

Physical testbed deployment over OpenSource MANO (OSM). Our testbed implemented on real-hardware is managed by the OpenSource MANO (OSM) orchestrator (Release 7). The orchestrator controls three OpenStack clusters (Stein Release), each including two compute nodes, for a total of six individually-addressable nodes on which the chain can be mapped. Although from the algorithm's point of view this is a much smaller scenario than the one emulated with mininet, it is still scaled down within a single order of magnitude with respect to the emulated testbed (50 mininet nodes vs. 6 hardware nodes). Moreover, the physical testbed is composed of full-fledged servers, making their performances comparable to those that would be observed in a production environment.

We defined a format for the exchange of information between the allocation algorithm and the hardware orchestrator, using the JSON formalism:

```
{ "sfc-id": "id_value",
  "vnfs": [
    {
      "type": "type_value",
      "node": "node_value"
    }, ...]
}
```

where: `sfc-id` is a unique identifier associated with the specific chain, in the form of a text string `id_value`; `vnfs` is the ordered list of service functions forming the chain; each service function is described by fields `type` and `node`, with `type_value` and `node_value` being strings identifying a predefined Virtual Network Function Descriptor in the orchestrator, and the physical node the service function has been mapped onto, respectively.

(1) Multiple virtual resources allocated on the same hosting node lead to smaller chain allocation ratios. This is because central hosting paths become quickly congested,

leaving no room for future chain requests (Figures 3a and 3b).

(2) Allowing multiple chain nodes to be mapped onto the same hosting node leads to a slower convergence time. Although we showed that the worse case convergence bound is the same, our results show that practically there is a difference. This can be understood in an extreme case where a single chain node cannot be allocated, hence the entire chain request is rejected, without trying to elect a leader to host other elements of the chain (Figure 3c). The contiguous path instantiation policy has the same response time as the multi-node mapping policy. This is because the response time does not depend on the constraints on the links, but merely on the number of nodes (see how the curve in Figure 3c related to multiple votes overlaps with the contiguous path).

(3) Instantiating contiguous virtual paths may increase the chain acceptance rate when multiple chain nodes are allowed to be hosted on the same host. This result is surprising and counterintuitive. We were expecting that by adding the contiguous additional constraint, on average, instantiating multiple chains on the same distributed multi-provider infrastructure would prune solutions. Our evaluation results instead explained that, when multiple chain nodes are allowed to be hosted by the same hosting node, it is better (in terms of allocation ratio) if virtual paths are allocated on contiguous (physical) paths, avoiding long path stretches. This is because, intuitively, a couple of hosting nodes that are distant in terms of number of hops but both with higher similar residual capacity may end up winning alternatively multiple chain nodes, forcing multiple longer hosting paths (Figure 3ab). Figure 3b shows that the main responsibility for non allocating chains is due to a lower link mapping, *i.e.*, node utilities that do not take into account full paths may lead to suboptimal chain allocations. This in turn suggests that further exploration of distributed constraint path vectors is needed to instantiate a chain, and that node agreement strategies alone are insufficient to guarantee optimal chain instantiations.

(4) With Necklace, providers can tune load profiles by balancing the load or packing hosts. Allowing a single chain node per host balances the overall host network load and lowers utilization. These results were confirmed across different hosting network sizes (Figure 3d). On the contrary, allowing multiple chain nodes on the same host packs resources and has a "bin packing" effect, potentially reducing energy costs by leaving unused hosting machines idle. The load packing effect is partially mitigated when we increase the virtual link mapping policy k, allowing virtual link splitting across up to $k = 4$ hosting paths when available. Our experiments shows that (in BRITE generated topologies) there are very rarely no more than 4 disjoint paths available.

(5) Necklace outperforms the (decentralized) Raft consensus protocol in terms of overhead, with or without failures. As we show in Figure 6a, Raft has higher overhead with respect to our DACCA, as we increase the number of processes that need to reach an asynchronous consensus. Note that DACCA
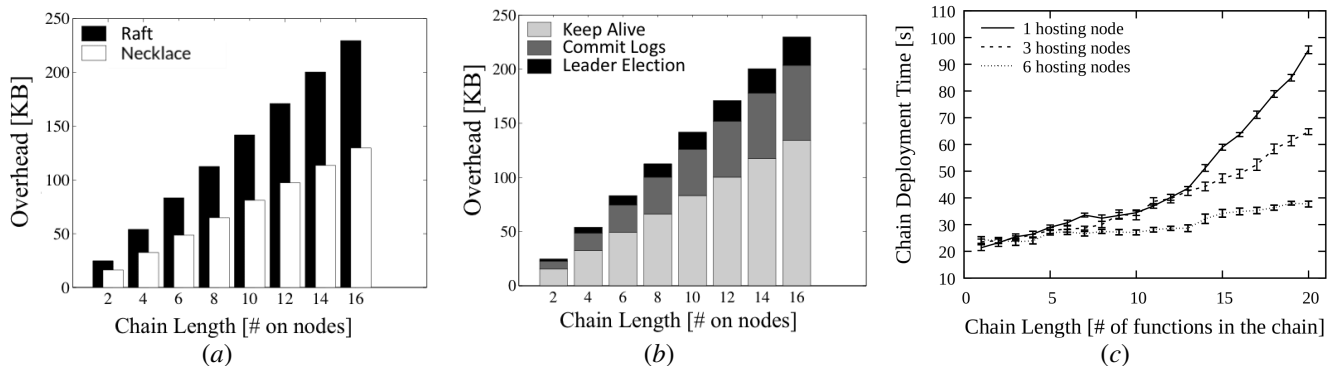
Fig. 6: (a) Overhead of our DACCA consensus protocol compared to Raft: DACCA's overhead is lower, especially as the number of hosting nodes that need to reach consensus grows. (b) Raft overhead dissected by phase: DACCA does not have to commit logs and does not have to elect a leader. (c) OpenSource MANO (OSM) deployment time as a function of chain length and number of hosting nodes.

inherits the properties of the max-consensus and so converges with the minimum possible number of messages, but more importantly, when a leader fails in Raft (as well as in similar consensus protocols such as Paxos [12] or its improvements), processes have to re-elect a new leader. Our fully distributed consensus-based auction instead, does not need a leader and so it has less overhead in case any of the processes fails. Raft (or Paxos) were not designed to deal with resilient network function orchestration and Necklace could work even replacing DACCA with Raft; by doing so, however, also the performance guarantees would be lost. Finally, Raft was not designed to be partition tolerant, while DACCA can still instantiate a chain as long as the majority of hosting nodes agree on a maximum objective vote set. Figure 6b shows how the overall overhead of Raft is higher as the hosting nodes that need to reach consensus grow. On the right plot we dissect the overhead of Raft: DACCA does not have to commit logs and does not have to elect a leader, hence it has a lower overhead. In summary, being a leader-based decentralized protocol and not fully distributed, Raft has higher overhead compared to DACCA because not only the maximum vote needs to propagate, but the entire log needs to be pushed from the leader to all the followers.

(6) A larger number of hosting nodes ensures a faster deployment. In our physical experimental testbed using OSM, we injected a progressively larger number of chain deployment requests, allowing service functions to be mapped onto one, then three, then all six physical nodes. For each case, we submitted requests for twenty different values of chain length, ranging from 1 to 20. We measured the deployment time of each chain, then removed the chain before running the next experiment, so as to have independent measurements. We performed 30 measurements per chain length, and plotted the average deployment time with confidence intervals. Each measurement campaign took anywhere between 8 and 14 hours. As Figure 6c shows, having more hosting nodes results in a better chain resource management, as service functions can be allocated on more nodes thus balancing loads, therefore resulting in a shorter chain deployment time. While this is an expected result, it demonstrates that the proof-of-concept implementation with OSM is functional, and hence Necklace can be used to deploy service chains on real systems within

reasonable time scales (tens of seconds).

IX. CONCLUSION

In this paper we proposed Necklace, an architecture for service function chain instantiation with programmable policies. Through this architecture, we modeled the separation and interaction of functionalities of the SFC instantiation problem via a stochastic optimization model, and then used such model to design our Necklace system. DACCA, the core mechanisms of Necklace, is a decentralized max-consensus protocol that allows providers to cooperatively instantiate wide-area service function chains. Hosting processes running DACCA have guarantees on convergence and on performance bounds with respect to a Pareto optimal instantiation, even in the presence of (non-byzantine) failures. Our Necklace's performance evaluation showed a few surprising results, among which the higher acceptance rate when the chain request is released for instantiation sequentially, *i.e.*, each hosting process should allocate a single chain node at a time. Our Necklace prototype can be used by the community interested in providers mechanism design or tradeoff analysis. Our work leaves several open questions, *e.g.*, how to design a path vector protocol with the same allocation guarantees, while being fully distributed (not decentralized) and resilient to byzantine failures as well.

REFERENCES

[1] F. Esposito, I. Matta, and V. Ishakian, "Slice Embedding Solutions for Distributed Service Architectures," *ACM Computing Surveys*, vol. 46, no. 2, pp. 6:1–6:29, March 2014.
[2] The Chamaleon Testbed. [Online]. Available: https://ben.renci.org
[3] The Fabric Testbed Initiative. (2020) http://www.whatisfabric.net.
[4] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proc of the 9th USENIX Conf. on OS Design and Implementation*, ser. OSDI'10, 2010, pp. 1–16.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 242–253.

[6] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *In Proc. of USENIX NSDI '13*, Lombard, IL, 2013, pp. 171–184.

[7] F. Esposito, D. Di Paola, and I. Matta, "On distributed virtual network embedding with guarantees," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 569–582, 2016.

[8] C. Gkantsidis, T. Karagiannis, P. Key, B. Radunovic, E. Raftopoulos, and D. Manjunath, "Traffic management and resource allocation in small wired/wireless networks," in *Proc. of CoNEXT '09*, 2009, pp. 265–276.

[9] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. of the ACM SIGCOMM 2012*, pp. 127–138.

[10] Berde, Pankaj et al., "onos: Towards an open, distributed sdn os."

[11] Linux Foundation. (2015) The OpenDayLight Project. http://opendaylightproject.org/.

[12] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998. [Online]. Available: http://acm.org/10.1145/279227.279229

[13] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '07, 2007, pp. 398–407.

[14] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Jun. 2014, pp. 305–319.

[15] L. Lamport, "Fast paxos," *Distributed Computing*, vol. 19, no. 2, pp. 79–103, 2006.

[16] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos: A high-throughput atomic broadcast protocol," in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, June 2010.

[17] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proc. of ACM SIGCOMM Hotnets-IX*, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[18] Facebook Dataset, https://www.facebook.com/network-analytics.

[19] ETSI NFV ISG, "Demonstration of multi-location, scalable, stateful Virtual Network Function," Dec 2014.

[20] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, and A. Erbad, "Multi-cloud distribution of virtual functions and dynamic service deployment: Open adn perspective," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, March 2015, pp. 299–304.

[21] S. Kumar et. al. - IETF SFC WG, "Service Function Chaining Use Cases In Data Centers," Jan 2015.

[22] A. Abujoda and P. Papadimitriou, "MIDAS: Middlebox discovery and selection for on-path flow processing," in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2015, pp. 1–8.

[23] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, Feb 2015.

[24] Charalampos Rotsos et al., "Network service orchestration standardization: A technology survey," *Computer Standards and Interfaces*, vol. 54, pp. 203 – 215, 2017, sI: Standardization SDN and NFV.

[25] OpenBaton, "http://openbaton.github.io/," 2017.

[26] Cerrato, Ivano et al., "Toward dynamic virtualized network services in telecom operator networks," *Comput. Netw.*, vol. 92, no. P2, pp. 380–395, Dec. 2015.

[27] B. Sonkoly, J. Czentye, R. Szabo, D. Jocha, J. Elek, S. Sahhaf, W. Tavernier, and F. Risso, "Multi-domain service orchestration over networks and clouds: A unified approach," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 377–378. [Online]. Available: http://doi.acm.org/10.1145/2785956.2790041

[28] G. Davoli, W. Cerroni, S. Tomovic, C. Buratti, C. Contoli, and F. Callegati, "Intent-based service management for heterogeneous software-defined infrastructure domains," *International Journal of Network Management*, vol. 29, no. 1, p. e2051, 2019.

[29] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, "On Supporting Mobility and Multihoming in Recursive Internet Architectures," *Computer Communications*, vol. 35, no. 13, pp. 1561 – 1573, 2012.

[30] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, Aug. 2014.

[31] Zave, Pamela et al., "Dynamic service chaining with dysco," in *Proceedings of SIGCOMM '17*, 2017, pp. 57–70.

[32] D. Borsatti, G. Davoli, W. Cerroni, and F. Callegati, "Service Function Chaining leveraging Segment Routing for 5G Network Slicing," in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–6.

[33] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, no. 1, pp. 95–116, 1984. [Online]. Available: http://dx.doi.org/10.1002/net.3230140109

[34] P. Khadivi et al, "Multi-constraint qos routing using a new single mixed metric," in *Communications, 2004 IEEE International Conference on*, vol. 4, June 2004, pp. 2042–2046 Vol.4.

[35] X. Yuan and X. Liu, "Heuristic algorithms for multi-constrained quality of service routing," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2001, pp. 844–853 vol.2.

[36] Dijkstra, "A note on two problems in connexion with graphs," *E.W. Numer. Math.*, vol. 269, 1959.

[37] D. Chemodanov, P. Calyam, F. Esposito, and A. Sukhov, "A general constrained shortest path approach for virtual path embedding," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016, pp. 1–7.

[38] Editors P. Quinn and T. Nadeau, *RFC 7498 - Problem Statement for Service Function Chaining*. IETF, April 2015. [Online]. Available: https://tools.ietf.org/html/rfc7498#section-2.1

[39] S. S. Skiena., *Set Packing. The Algorithm Design Manual.*, 1997.

[40] J. Case, "A Simple Network Management Protocol (SNMP). ARPA Request For Comment (RFC) - 1157," May 1990. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc1157.txt

[41] C. Partridge and G. Trewitt, "High-level Entity Management Protocol (HEMP)," RFC 1022, October 1987.

[42] ISO/IEC 9596-1, "Information Technology - OSI, Common Management Information Protocol (CMIP) - Part 1: Specification," 1991, Also CCITT X.711.

[43] Google Protocol Buffers. (2013) Developer Guide http://code.google.com/apis/protocolbuffers.

[44] C. Partridge and G. Trewitt, "High-level Entity Management Protocol (HEMP)," RFC 1022, October 1987.

[45] N. A. Lynch, *Distributed Algorithms*, 1st ed. Morgan K., Mar. 1996.

[46] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, p. 374–382, Apr. 1985. [Online]. Available: https://doi.org/10.1145/3149.214121

[47] L. Johnson, H.-L. Choi, S. Ponda, and J. How, "Allowing Non-submodular Score Functions in Distributed Task Allocation," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, 2012, pp. 4702–4708.

[48] G. Nemhauser, L. Wolsey, and M. Fisher, "An Analysis of Approximations for Maximizing Submodular Set Functions," *Mathematical Programming*, vol. 14, no. 1, p. 265–294, 1978.

[49] A. Kulik, H. Shachnai, and T. Tamir, "Maximizing Submodular Set Functions Subject to Multiple Linear Constraints," ser. SODA '09, Philadelphia, PA, USA.

[50] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985. [Online]. Available: http://doi.acm.org/10.1145/3149.214121

[51] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Addison Wesley, 2009.

[52] U. Feige, "A Threshold of ln n for Approximating Set Cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, Jul. 1998.

[53] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based Decentralized Auctions for Robust Task Allocation," *IEEE Transaction of Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[54] D. S. Hochbaum, "Approximation Algorithms for NP-hard Problems." Boston, MA, USA: PWS Publishing Co., 1997, pp. 94–143.

[55] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[56] A. Medina, A. Lakhina, I. Matta, and J. W. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS*, 2001.

Flavio Esposito (flavio.esposito@slu.edu) is an Assistant Professor with the Department of Computer Science at Saint Louis University (SLU), with a second affiliation in Electrical and Computer Engineering withing the Parks College of Engineering at SLU. He received the M.Sc. degree in telecommunication engineering from the University of Florence, Italy, and the Ph.D. in computer science from Boston University in 2013. His research interests include network management, network virtualization, artificial intelligence, and distributed systems. Dr. Esposito is a currently a principal investigator of four National Science Foundation grants, and one grant from the International Center for Responsible Gaming. Dr. Esposito was awarded with the Comcast Innovation Fund Award in 2020, and is the recipient of three best paper awards from the IEEE.

Maria Mushtaq is a Graduate Research Assistant with the Department of Computer Science at Saint Louis University. She received a B.S. degree in Computer Science from Karachi Institute of Economics and Technology (PAF-KIET), Pakistan in 2015. Followed by a M.S. degree in Project Management from Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Pakistan in 2019. Her research interests include Machine Learning and its application in the field of Networking.

Michele Berno received the B.Sc. degree in Information Engineering and the M.Sc.degree (Hons.) in Telecommunication Engineering from the University of Padova, Italy, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree in ICT Engineering with the SIGNET research group in the Department of Information Engineering (University of Padova). During 2019, he was a research scholar with the Department of Computer Science, Saint Louis University (SLU), Missouri, US. He was a recipient of two best paper awards (FMEC 2019, IEEE MobileCloud 2020). His research interests include sustainable edge/cloud computing, distributed optimization, and machine learning.

Gianluca Davoli received his M.Sc. Degree in Telecommunications Engineering from the University of Bologna, Italy, in 2017, and he currently is a PhD candidate and Research Fellow there. His fields of interest revolve around communication networks, focusing on the new approaches to programmability, management and monitoring of software-based network infrastructures.

Davide Borsatti received his B.S. and M.S. in Telecommunications Engineering from University of Bologna in 2016 and 2018 respectively. He is currently enrolled in the Electronics, Telecommunications, and Information Technologies Engineering PhD program from the University of Bologna. His research interests include NFV, SDN, Intent Based Networking, MEC and 5G Network slicing.

Walter Cerroni [M'01, SM'16] (walter.cerroni@unibo.it) is an Associate Professor of communication networks at the University of Bologna, Italy. His recent research interests include software-defined networking, network function virtualization, service function chaining in cloud computing platforms, intent-based northbound interfaces for multi-domain/multi-technology virtualized infrastructure management, modeling and design of inter- and intra-data center networks. He co-authored more than 130 articles published in the most renowned international journals, magazines and conference proceedings. He serves/served as Series Editor for the IEEE Communications Magazine, Associate Editor for the IEEE Communications Letters, and Technical Program Co-Chair for IEEE-sponsored international workshops and conferences.

Michele Rossi (SM'13) is a Full Professor of Telecommunications in the Department of Information Engineering (DEI) at the University of Padova (UNIPD), Italy, teaching courses within the Master's Degrees in ICT for internet and Multimedia at DEI (http://mime.dei.unipd.it/ and Data Science, offered by the Department of Mathematics (DM) at UNIPD (https://datascience.math.unipd.it/). Since 2017, he has been the Director of the DEI/IEEE Summer School of Information Engineering (http://ssie.dei.unipd.it/). His research interests lie in wireless sensing systems, green mobile networks, edge and wearable computing. In recent years, he has been involved in several EU projects on IoT technology (e.g., IOT-A, project no. 257521), and has collaborated with companies such as DOCOMO (compressive dissemination and network coding for distributed wireless networks) and Worldsensing (optimized IoT solutions for smart cities). In 2014, he was the recipient of a SAMSUNG GRO award with a project entitled "Boosting Efficiency in Biometric Signal Processing for Smart Wearable Devices". In 2016-2018, he has been involved in the design of IoT protocols exploiting cognition and machine learning, as part of INTEL's Strategic Research Alliance (ISRA) R&D program. His research is currently supported by the European Commission through the H2020 projects SCAVENGE (no. 675891) on "green 5G networks", MINTS (no. 861222) on "mm-wave networking and sensing" and GREENEDGE (no. 953775) on "green edge computing for mobile networks" (project coordinator). Dr. Rossi has been the recipient of seven best paper awards from the IEEE and currently serves on the Editorial Boards of the IEEE Transactions on Mobile Computing, and of the Open Journal of the Communications Society.