Received July 24, 2020, accepted August 7, 2020, date of publication September 7, 2020, date of current version September 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3021355

Deep Learning for Service Function Chain Provisioning in Fog Computing

NAZLI SIASI[1], (Member, IEEE), MOHAMMED JASIM[2], (Member, IEEE), ADEL ALDALBAHI[3], (Member, IEEE), AND NASIR GHANI[4], (Senior Member, IEEE)
[1]Department of Physics, Computer Science and Engineering, Christopher Newport University, Newport News, VA 23606, USA
[2]School of Engineering, University of Mount Union, Alliance, OH 44601, USA
[3]Department of Electrical Engineering, King Faisal University, Al-Ahsa 31982, Saudi Arabia
[4]Department of Electrical Engineering, University of South Florida, Tampa, FL 33620, USA

Corresponding author: Nazli Siasi (nazlisiasi@mail.usf.edu)

ABSTRACT Cloud and fog computing along with network function virtualization technology have significantly shifted the development of network architectures. They yield in reduced capital and operating expenditures, while achieving network flexibility and scalability to accommodate the massive growth in data traffic volumes from user terminals requesting various services and applications. Now cloud solutions here offer abundant computing and storage resources, at the detriment of high end-to-end delays, hence limiting quality of service for delay-sensitive applications. Meanwhile, fog solutions offer reduced delays, at the detriment of limited resources. Existing efforts focus on merging the two solutions and propose multi-tier hybrid fog-cloud architectures to leverage their both saliencies. However, these approaches can be inefficient when the applications are delay-sensitive and require high resources. Hence this work proposes a novel standalone heterogeneous fog architecture that is composed of high-capacity and low-capacity fog nodes, both located at the terminals proximity. Thereby, realizing a substrate network that offers reduced delays and high resources, without the need to relay to the cloud nodes. Moreover, the work here leverages and deploys a deep learning network to propose a service function chain provisioning scheme implemented on this architecture. The scheme predicts the popular network functions, and maps them on the high-capacity nodes, whereas it predicts the unpopular network functions and maps them on the low-capacity nodes. The goal is to predict the next incoming function and prefetch it on the node. Hence, when a future request demands the same function, it can be cached directly from the node, at reduced resources consumption, processing times, cost, and energy consumption. Also, this yields in higher number of satisfied requests and increased capacity. The deep learning network yields reduced loss model and high success rates.

INDEX TERMS Cloud computing, deep learning, fog computing, function popularity, long short-term memory, network function virtualization, service function chain, preteching and caching.

I. INTRODUCTION

Cloud computing technology provides a centralized paradigm for computationally intensive tasks for network providers. This is attributed to its abundant available resources at the network backbone. Here the computational complexity, power consumption at the network terminals are alleviated by eliminating local data processing [1], [2]. However, cloud computing introduces increased delays in the network due to the extended link propagation distances between terminals and stationary cloud nodes that are geographically distributed. This also leads to prolonged processing times of various tasks, end-to-end latencies, congestion and jitters, packet losses, and mobility management challenges, i.e., long handover times [3], [4]. This can further be aggregated by the massive growth of internet of things (IoT) terminals that require real-time and delay-sensitive applications. Hence cloud computing solutions can present a bottleneck, despite the abundant resources offered in accommodating massive traffic information.

Along these lines, fog computing paradigms are proposed to overcome the aforementioned cloud limitations. This is accomplished by transferring the computing and storage resources, relaying and caching services from the cloud to the

The associate editor coordinating the review of this manuscript and approving it for publication was Chunsheng Zhu.

edge of the network. Tasks are then processed and executed in a decentralized manner, intermediary between terminals and cloud nodes. This reduces delays and latencies and alleviates overloads at cloud nodes. Hence computational services are now deployed on fog nodes with location-aware and mobility support. Despite these saliencies, however fog computing can suffer from limited resources at high traffic volume scenarios, which impose excessive needs for scheduling and load balancing.

Furthermore, virtualization technologies, such as network function virtualization (NFV) and software defined network (SDN) [5], [6] facilitate accessing and sharing cloud or fog nodes resources and services by various applications. NFV here decouples network functions (NFs) such as firewall, DNS, caching, and evolved packets, from proprietary hardware (nodes) [7], and enables them as virtual network functions (VNFs) to run as software instances over dispersed (cloud or fog) nodes at various locations.

Research Problem: A major design challenge is VNF mapping onto the underlying nodes, in order to realize the network virtualization paradigm. Moreover, the VNFs are often chained together in a specific sequence specified by terminal requests, thereby forming a service function chain (SFC). Now performing efficient SFC provisioning on cloud or fog nodes is a challenging task, in particular achieving efficient resources utilization, reduced delays, energy and cost, and high network capacity.

Hence this paper aims to address the problems of VNF mapping and SFC provisioning in standalone fog computing. Note that the aforementioned efforts in NFV-based fog networks lack SFC provisioning (dependency) when mapping and embedding the VNFs onto the fog nodes. Moreover, existing fog architectures suffer from limited processing and storage capabilities, as opposed to nodes in the cloud domain. Hence ii ts important to propose fog architectures that can accommodate requests at reduced delays and high processing capabilities. Here it is also important to study key performance metrics, such as usage rate, network saturation and capacity, power and energy consumption and cost.

II. RELATED WORK

Various studies have looked at the SFC provisioning problem onto cloud and fog architectures. First, the work in [8] integrates NFV with fog computing for reduced overhead and network flexibility for fog access points (F-AP) with fixed catches to support a handover scheme for 5G systems. Similarly, the work in [9] presents a centralized VNF mapping approach to accommodate stringent delay constraints. Here the VNF mapping problem is formulated as a graph-clustering optimization model and a genetic algorithm is leveraged to minimize fronthaul cost. Moreover, authors in [10] propose a multi-tier architecture for platform as-a-service (PaaS) that splits VNF mapping onto cloud and fog nodes, as opposed to fog nodes to measure end-to-end delay. however, the work here is limited to a single VNF and does not consider mapping onto fog nodes only. Similarly, work in [11] supports provisioning for healthcare applications with components spanning both cloud and fog nodes. This work focuses only on mechanisms for providing control, signaling, and data interfaces between the cloud and fog nodes in a hybrid settings.

Also, authors in [12] present various methods to monitor and migrate applications components between cloud and fog nodes, where tradeoff between power consumption and delay is studied. Namely, the mapping problem here is formulated as an optimization model, and then an approximate solution is proposed to decompose the primary problem into three subproblems of corresponding subsystems (solved independently). Overall, the goal is to alleviate link bandwidth usage, reduce transmission latencies, at the detriment of high computing resources. Note that VNF mapping is not considered here. Additionally, work in [13] presents the benefits associated with service migration from cloud to multi-tier fog nodes based on SDN for video distribution. The required time for migration between the different tiers is studied in efforts to minimize traffic at the core network. However, the work here does not consider the resource limitations at highly congested multi-tier fog nodes and lacks the study on delay-tolerant applications with high capacity demands, as well as VNF mapping.

Moreover, an application component placement scheme for NFV-based in a single-tier fog architecture is proposed in [14], where placement decisions are determined from an ILP solver to achieve cost minimization. This work is limited to small-scale scenarios and considers a single VNF type in a single-tier fog nodes. Also, work in [15] presents a multi-tier fog architecture for video streaming applications. Namely, it is composed of three tiers that are classified in terms of coverage, computing and storage capacities. Nonetheless, the work here only demonstrates suitable services for multi-tier architectures. It also lacks SFC provisioning and it is limited to video-streaming applications. Moreover, the extended three fog tiers model highly increases realization costs.

Machine learning techniques have been leveraged to perform NFV placement and SFC provisioning. First, the work in [16] focuses on the reliability requirement of requested services when performing NFV placement. Authors deploy deep reinforcement learning to model the NFV placement problem considering the reliability requirement of the services, when failure occurs. The output of the introduced model determines optimal placement in each state at minimized placement cost and increased number of admitted services. Work here considers failure probabilities, where if any of the servers hosting the VNFs fails, the service is then disrupted. Similarly, authors in [17] also consider the reliability-aware NFV placement problem on servers to achieve higher admission ratio and reduced cost. Here a deep Q-network (DQN) is used instead to meet the reliability requirement of the incoming services, when performing the SFC provisioning.

Furthermore, the work in [18] formulates the VNF placement problem in SDN/NFV-enabled networks as a binary integer programming (BIP) problem. Then a deep reinforcement learning algorithm termed as, double deep Q-network based VNF placement algorithm (DDQN-VNFPA) is used to determine the optimal solution. The algorithm impact on the network performance is tested in terms of the reject number and reject ratio of the SFC requests, cost, throughput, delay, running time and load. However, the work does not specify the architecture of the nodes that host the VNFs, i.e., lacking the fog computing architecture attributes, in terms of delays, distribution, delay, energy, cost, and usage settings.

A. MOTIVATIONS

Existing efforts in NFV-based fog networks lack SFC provisioning (dependency) when mapping and embedding the VNFs onto the fog nodes. Moreover, existing fog architectures suffer from limited processing and storage capabilities, as opposed to nodes in the cloud domain. Hence the research problem here is developing SFC provisioning schemes in fog computing. Namely, it is important to propose fog architectures that can accommodate requests at reduced delays and high processing capabilities. Here it is also important to study key performance metrics, such as usage rate, network saturation and capacity, power and energy consumption and cost.

The main goal is to propose SFC provisioning for fog computing. A compressive framework includes the request models from terminals, the fog computing architecture, and the mapping/routing methods that couple the services onto the network infrastructure. The goal also enables SFC provisioning for requests demanding various practical applications (VNFs of various types), while taking into accounts different network settings and constraints, such as request delay bounds, resources requirements, and node processing capacities. This benefits the network operators to accommodate for multiple requests of different specifications, e.g., delay-sensitive, delay-tolerant, data-intensive, while accommodating high traffic volumes with reduced dropping rates.

Overall, aforementioned efforts lack SFC provisioning (dependency) when mapping the VNFs, with components splitting between multi-tier fog and cloud nodes. Hence, the goal is to achieve SFC provisioning for requests demanding several applications (VNFs of various types), while taking into accounts different network setting, such as request delay bounds, resources requirements, and node processing capacities. This allows the network to account for multiple requests, e.g., delay-sensitive, delay-tolerant, data-intensive, while accommodating high traffic volumes with reduced dropping rates.

B. CONTRIBUTIONS

This paper presents an initial study on leveraging deep learning networks for an efficient SFC provisioning scheme in fog computing. The work proposes a novel heterogeneous fog (HF) architecture, over which the deep learning scheme is implemented, in order to overcome limitations associated with existing multi-tier fog and cloud architectures.

TABLE 1. Parameter settings for the lstm network.

Parameter	Value
LSTM layers, number of cells	4, [50,50,50,50]
Dense layer: Activation function	ReLU
Batch size	16
Epochs	350
Dropout rate	0.2
Optimizer	RMSprop
Learning rate	0.001
Drop-out regularization rate	0.2

Namely, the proposed architecture possesses heterogeneous fog nodes of different resources at close proximity to terminals, as opposed to conventional homogeneous multi-tier fog solutions. These nodes include a set of high-capacity fog (HCF) nodes that host high-frequency (popular) VNFs, and low-capacity fog (LCF) nodes hosting low-frequency (unpopular) VNFs from, high and low incoming traffic volumes, respectively. The deep learning network continuously monitors incoming requests and their VNFs, and aims to predict the future incoming VNFs and their popularity classes. Namely, the popular and unpopular VNFs are predicted, then prefetched and mapped to nodes that are close to the terminals. Thereafter, when a new request demands the same VNF, it is cached from the same node for faster retrieval. This eliminates redundant VNF mapping and duplicate transmissions, which reduces the processing times, minimizes network traffic, and saves node resources (used instead to accommodate additional new requests). The deep learning network model yields high success rates with low mean square error, i.e., hence enhancing the accuracy of the proposed scheme. Overall, the SFC provisioning scheme features reduced network delays, resources, energy consumption, and realization costs, as well as achieving high number of satisfied requests.

This paper is organized as follows. The proposed HF architecture is first proposed in Section III. Then the network model incorporating the topology and request model is presented in Section IV. The proposed deep learning SFC provisioning scheme is detailed in Section V. This is followed by simulation results in Section VI, performance evaluation in Section VII, along with conclusion in Section VIII. Moreover, Tables 1,2,3 and 4 in this paper list the parameter settings for the LSTM-based deep learning network, simulation parameters assignment, acronyms & abbreviations, and key notations, respectively.

III. PROPOSED HETEROGENEOUS FOG ARCHITECTURE

Consider a geographical area composed of adjacent fog clusters, each comprises multiple high-capacity fog (HCF) nodes and low-capacity fog (LCF) nodes. This combination forms heterogeneous fog (HF) architecture that serves multiple terminals, e.g., pedestrians, vehicles, indoor users, as depicted in Figure 1. Consider the details.

Assumptions: Consider the following assumptions for the requests, architecture for the proposed provisioning scheme.

First, the SFC requests generated from the terminals vary in terms of the delay, processing, storage and cache

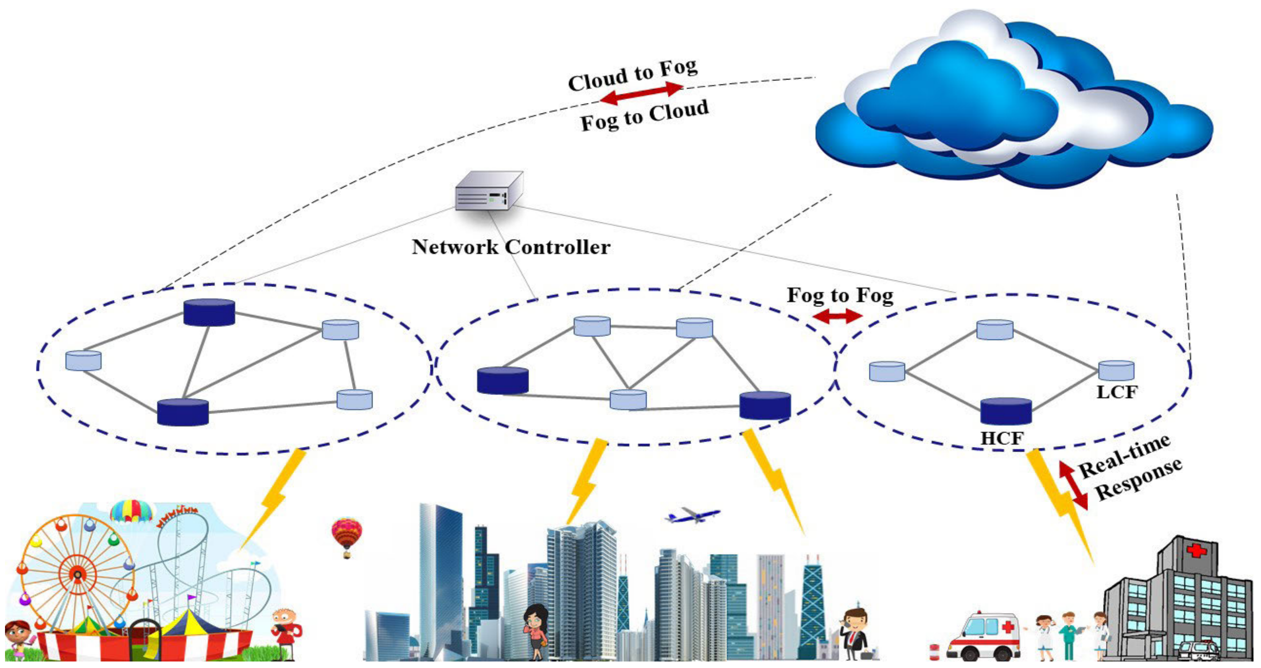


FIGURE 1. Heterogeneous fog (HF) architecture composed of high- and low-capacity nodes.

TABLE 2. Simulation parameters.

Parameter	Value
Num. of Nodes: N_m, N_{hc}, N_{lc}	[1,150], 25, 58
Num. of Nodes per cluster	[5,6]
Node processing capacity (GHz): $Q_{pr}(n_{lc})$, $Q_{pr}(n_{hc})$	5, 15
Node memory (GB): $Q_{me}(n_{lc})$, $Q_{me}(n_{hc})$	32, 64
Link bandwidth capacity (Gbps): $B(e_{m,lc})$, $B(e_{m,hc})$, $B(e_{lc,lc})$, $B(e_{lc,hc})$, $B(e_{hc,hc})$	0.5, 0.75, 1, 2, 3
Num. VNF / request	Rand[1-5]
Requested VNF processing, $Q_{pr}(v_u)$	1-3 (uniform)
Requested VNF memory, $Q_{me}(v_u)$	Rand [1-5]
Request bandwidth (Mbps), B_r	1-10 (uniform)
Number of requests per batch, R	150
Traffic load A_r (KB)	70
Edge transmission rate (rand[] ms)/traffic unit (GB): $\delta(e_{m,lc})$, $\delta(e_{m,hc})$, $\delta(e_{lc,lc})$, $\delta(e_{lc,hc})$, $\delta(e_{hc,hc})$	[0.5, 1.5], [0.5, 2.5], [2.5, 3.5], [3.5, 5], [5, 10]
Edge cost (\$/GB): $\rho(e_{m,lc})$, $\rho(e_{lc,lc})$, $\rho(e_{lc,hc})$, $\rho(e_{hc,hc})$	1, 2, 2.5, 3
VNF processing rate at nodes: $\delta(n_{lc})$, $\delta(n_{hc})$	0.03, 1.03
VNF license cost (\$): $\rho(v_u)$	100
Node cost per resource unit (\$/processor): $\rho_{pr}(n_{lc})$, $\rho_{pr}(n_{hc})$	5, 3
Node cost per resource unit (\$/memory): $\rho_{me}(n_{lc})$, $\rho_{me}(n_{hc})$	6, 4
Power parameters: $\epsilon(n)$, w_x, $w(n_{lc})$, $w(n_{hc})$, $w(n_{nc})$	0.7, 15W, 65W, 120W, 200W

TABLE 3. List of acronyms and abbreviations.

Term	Acronym
Authentication, Authorization & Accounting	AAA
Binary Integer Programming	BIP
Deep Q-Network	DQN
Double Deep Q Network	DDQN
Fog access points	F-AP
Heterogeneous Fog	HF
High-Capacity Fog	HCF
Internet of Things	IoT
Low-Capacity Fog	LCF
Long Short-Term Memory	LSTM
Mean Square Error	MSE
Network Controller	NC
Network Function	NF
Network Function Virtualization	NFV
Pico Base Station	PBS
Radio Remote Head	RRH
Software Defined Network	SDN
Service Function Chain	SFC
Platform-as-a Service	Paas
Virtual Network Function	VNF
VNF Placement Algorithm	VNFPA

requirements, types and numbers of VNFs, frequency (number of popular/unpopular requests from terminals). The architecture is composed of heterogeneous nodes of different resources and separation distances. The nodes and links possess finite resources in order to consider network congestion, saturation (working with high traffic volume).

A. HIGH-CAPACITY FOG (HCF) NODE

This node is located at the edge of the network in proximity with terminals. Thereby, services are provided at short separation distances and thus enabling low propagation times across links. Any HCF node is collocated with a high-power transmission unit, e.g., radio remote head (RRH). Hence providing high traffic capacity in congested zones with demands of different applications. A key saliency here is when terminals are clustered in small regions, then this node can support terminals with high processing and storage capacities, as well as high data rates, at reduced network delays.

Moreover, the high processing capabilities here enable elastic scalability to the network. In particular, when traffic volumes increase. Then the HCF node can provide simultaneous signaling services to terminals without the need for

TABLE 4. **Key notations and variables.**

Input Parameter	Notation
N	Set of all terminal and network nodes
E	Set of all network links
N_m, N_{lc}, N_{hc}	Number of terminals, LCF, HCF
$e_{m,lc}$, $e_{m,hc}$, $e_{lc,hc}$	Links between terminals & LCF/HCF nodes
$e_{lc,lc}$, $e_{lc,hc}$, $e_{hc,hc}$	Links between LCF and HCF nodes
$Q_{pr}(n_{lc})$, $Q_{pr}(n_{hc})$	LCF, HCF node computing resources
$Q_{me}(n_{lc})$, $Q_{me}(n_{hc})$	LCF, HCF node memory resources
$Q_{ch}(n_{lc})$, $Q_{ch}(n_{hc})$	LCF, HCF node cache capacity
$B(e)$	Link bandwidth capacity of link e
R	Number of requests per batch
U	Total number of VNF types
V_r	Set of required VNF types for request r
v_u	VNF of type $u \in U$
S_{v_u}	Set of VNF instances associated to v_u
Dep_r	SFC dependency order
$Q_{pr}(v_u)$, $Q_{me}(v_u)$, $Q_{ch}(v_u)$	Processing, memory and chase resources demands for v_u
$Q_{pr}(n_{hc})$, $Q_{me}(n_{hc})$,$Q_{ch}(n_{hc})$	Available processing, memory and chase resources at HCF node n_{hc}
$Q_{pr}(n_{lc})$, $Q_{me}(_{lc})$, $Q_{ch}(_{lc})$	Available processing, memory and chase resources at LCF node n_{lc}
ρ_r	Request delay bound
B_r	Request bandwidth
A_r	Traffic load for request r
L_r	Request lifetime
Y^i_{lc}, Y^i_{hc}	Captured power level of LCF, HCF nodes
$\Omega(n)$, $\Omega(e)$	Node and link weights
k, K	Path vector, set of path vectors between source and least-loaded candidate nodes.
$\delta(e)$	Edge transmission rate of link e
$\rho(e)$	Edge cost of link e
$\delta(n_{lc})$, $\delta(n_{hc})$	VNF processing rate at nodes
$\lambda^r_{v_u}$	Number of VNFs of a type u in r
Γ^r_e	Number of virtual mapped links in r
$\rho(v_u)$	License cost for v_u
$\rho_{pr}(n_{lc})$, $\rho_{pr}(n_{hc})$	Node cost per resource unit ($/processor)
$\rho_{me}(n_{lc})$, $\rho_{me}(n_{hc})$	Node cost per resource unit ($/memory)
T_{learn}, T_{train}	Learning period and training period
V_p, $V_{\overline{p}}$	Set of popular and unpopular VNFs
$\vartheta(n_{lc})$, $\vartheta(n_{hc})$	Number of popular and unpolular VNFs cached in the LCF and HCF
$V_{pref}(n_{lc})$,$V_{pref}(n_{hc})$	VNF preteching list on LCF and HCF
$V_{er}(n_{lc})$, $V_{er}(n_{hc})$	Erasing list for LCF and HCF
$D_{pr}(n')$	Processing delay of candidate node n'
$D_{tran}(e')$, $D_{prop}(e')$	Link transmission and propagation delays for candidate link e'
Υ	Total number of VNFs in the traffic

offloading to other clusters or relaying to the cloud node. This in turn alleviates signaling traffic on the network fronthaul, and reduces usage of cloud nodes. Note that each HCF possesses high processing, memory, and cache capacities, denoted by $Q_{pr}(n_{hc})$, $Q_{me}(n_{hc})$, $Q_{ch}(n_{hc})$, respectively. These capacities are dedicated to support high-frequency (popular) VNFs, as detailed later.

B. LOW-CAPACITY FOG (LCF) NODE

This node is also located at the terminals proximity, and it possesses processing, memory, and cache capacities, denoted by $Q_{pr}(n_{lc})$, $Q_{me}(n_{lc})$, $Q_{ch}(n_{lc})$, respectively. These resources are fold less than the HCF node resources, i.e., $Q_{pr}(n_{lc}) \ll Q_{pr}(n_{hc})$, $Q_{me}(n_{lc}) \ll Q_{me}(n_{hc})$, and $Q_{ch}(n_{lc}) \ll Q_{ch}(n_{hc})$. This is important for efficient resources utilization, minimizing cost and power consumption design goals, as these nodes are intended to support low-frequency (unpopular) VNFs in the incoming requests. Therefore, each LCF node is collocated with a low-power radio unit, e.g., pico base stations (PBS) to support low traffic volumes and seamless coverage in sparse terminals distribution. Moreover, the deployment of multiple LCF nodes in each cluster can provide ubiquitous coverage for terminals.

Note that the HCF locations are modeled as homogeneous Poisson point process distribution that act as parent point process to the LCF nodes in the cluster, which follow point process of symmetric normal distribution around the HCF nodes. Here the radio units are all responsible for radio interfacing, i.e., system broadcasting information, control and data signaling with terminals. For example, this includes association, mobility management, and quality of service control, and authentication, authorization, and accounting (AAA). Meanwhile, the fog nodes host the VNFs of the incoming requests. They monitor incoming traffic, generate database, report to the network controller (NC) and operate in learning and training modes during the SFC provisioning scheme, as detailed later.

C. CLOUD NODE

High-end cloud nodes are connected to the HCF and LCF by the NC through high bandwidth network backbone. Cloud nodes are responsible for migrating requested (network functions) NFs to the fog clusters, where it possesses high computing and memory resources to process and store enormous amounts of data, at the detriment of high end-to-end delays. However, provisioning in this work is performed solely on the fog clusters.

D. NETWORK CONTROLLER

This controller is utilized for optimal mapping management. It acts as the network manager that controls traffic assignment and broadcasting information between the fog (HCF, LCF) and cloud nodes. It maintains lists of the popular and unpopular VNFs, over which it broadcasts them to all participating fog nodes in clusters. Namely, it creates policies on whether VNFs are processed by a HCF or LCF node. This also includes processing, mapping, offloading, storing, prefetching and caching decisions. The policies are created based on the VNFs popularity classes developed during the in the deep learning network, as well as consideration for latency requirements, available resources, and energy consumption.

Overall, the NC maintains global knowledge of the entire network traffic collected and aggregated from the fog nodes. Meanwhile, fog nodes maintain local knowledge that relates to the traffic of their clusters.

As a result, each node is now aware of the mapped VNFs in neighboring nodes in the cluster. This avoids duplicate VNFs mapping, saves computing resources, eliminates redundant traffic flow and link congestion in the network. This also increases capacity by accommodating new VNFs of new requests.

IV. NETWORK MODEL

A. SUBSTRATE NETWORK TOPOLOGY

The substrate network for the proposed architecture is modeled as an undirected graph, $G = (N, E)$. It consists of vertices $n \in N$, $N = \{n : n_m, n_{lc}, n_{hc}\}$, where n can be either a terminal node $n_m \in N_m$, a LCF node $n_{lc} \in N_{lc}$, or HCF node $n_{hc} \in N_{hc}$, where N_m, N_{lc} and N_{hc} are the total number of terminals, LCF and HCF nodes in the network, respectively. Also, E represents the set of links in the network, i.e., $E = \{e : e_{m,lc}, e_{m,hc}, e_{lc,hc}\}$. Here a single link is denoted by e, and it can be either a link between a terminal and a LCF, $e_{m,lc}$ and HCF, $e_{m,hc}$, respectively, or between a LCF and HCF node, $e_{lc,hc}$.

Furthermore, each node can host one or more VNF, $v_u \in V_r$, i.e., $u = 1, 2, \ldots, U$, where U denotes the total number of VNF types. Moreover, V_r, is the set of all required VNFs in request r. The available substrate resources here are bounded by a finite set of constraints. Namely, the overall resources capacities at the HCF possess higher available resources than the LCF in the cluster. In general, each node n has a specific computing capability, expressed by the total processing capacity bounded by $Q_{pr}(n)$, memory bounded by $Q_{me}(n)$, and the available link bandwidth on a substrate link bounded by $B(e)$. These boundaries present a practical network realization and implementation for different traffic volumes, where the requested resources are higher than the available substrate resources.

B. TERMINAL REQUEST MODEL

A model is developed here for the terminal request $r \in R$ of specific resources and delay requirements, where R is the set of total requests from terminals. Each request r is expressed by 7-tuple $r =< src, dst, V_r, Dep_r, B_r, \rho_r, L_r >$. The variables $src, dst \in N$, denote in order the source (often terminals), destination nodes hosting last VNF in the SFC. The variables V_r and Dep_r represent the set of desired VNF types, and their dependency conditions, ordered in the SFC in r, $V_r = [v_1, v_2, \ldots, v_u]$, $Dep_r = \{v_1 \rightarrow v_2 \rightarrow v_u\}$. Also, B_r and ρ_r specify the required link bandwidth that interconnects VNFs, and the maximum network delay boundary for the request, respectively. Now each hosted VNF in the nodes requires specific processing $Q_{pr}(v_u)$ and memory $Q_{me}(v_u)$ resources. Finally, the variable L_r represents the service lifetime for the request, after which the request is cleared from the network. Figure 2 shows the network model for multiple incoming requests, with different dependency sequence as specified by Dep_r, as well as different popularities.

V. DEEP LEARNING SFC PROVISIONING SCHEME

The proposed SFC provisioning scheme operates in three modes, i.e., learning, training and running modes, as per Figure 3. In the first mode, the network learns about the traffic and establishes popularity classes and lists. In the second mode, the network is trained, while achieving training objectives and enhancing success rates. In the third mode, the long short-term memory (LSTM) provisioning scheme is proposed, as detailed later.

Now key design factors and constraints are taken into consideration for the SFC provisioning scheme. Foremost, the node and link capacity requirements.

1) Node Capacity Requirements: Each VNF, v_u, in the request list is mapped to either a LCF node n_{lc} or a HCF node n_{hc} on the substrate network that has enough available computing $Q_{pr}(n)$ and memory resources $Q_{me}(n)$, $n \in N - \{n_m\}$. A key condition here is that the sum of processing and memory capacities required by VNF instances mapped to the nodes cannot exceed the amount of available physical resources. This in turn avoids node overloading and requests drops. In notations,

$$\sum_{r\in R}\sum_{v_u\in V_r}\sum_{s\in S_{v_u}} \lambda^r_{v_u}.Q_{pr}(v_u) \leqslant Q_{pr}(n), \forall n \in N - \{n_m\}, \quad (1)$$

$$\sum_{r\in R}\sum_{v_u\in V_r}\sum_{s\in S_{v_u}} \lambda^r_{v_u}.Q_{me}(v_u) \leqslant Q_{me}(n), \forall n \in N - \{n_m\}, \quad (2)$$

where $\lambda^r_{v_u}$ denotes the number of VNFs of a specific type u in request r mapped to a node, and $s \in S_{v_u}$ is a single instance in total of S_{v_u} instances for the VNF of type v_u.

2) Edge Capacity Requirements: Here each link between two consecutive VNF nodes in the SFC request must be mapped to a substrate link, $e, e \in E$ of enough available bandwidth, $B(e)$, i.e., larger than the request bandwidth, B_r. This is necessary in order to transverse sufficient amount of flow (required by SFC requests) at reduced overloading. This is modeled as,

$$\sum_{r\in R}\sum_{e\in E'} \Gamma^r_e.b_r \leqslant B(e), \quad (3)$$

where Γ^r_e denotes the number of virtual links in request r mapped to the network.

A. MODE I: LEARNING MODE

This is the mode (Figure 3) for learning for the VNFs popularity classes, during which the network is collecting and learning the traffic patterns. The network still lacks sufficient traffic patterns at the start of its operation, since the learning period,T_{learn}, is still less than the training period, T_{train}. Therefore, conventional provisioning is conducted for all the VNFs (network operating in normal settings). Here all the VNFs are mapped regardless of their frequencies and popularity classes. Hence each VNF occupies its own dedicated resources in the node.

Thereafter, once the first learning period has elapsed, then the network is trained, i.e., $T_{learn} \geq T_{train}$. Sufficient traffic patterns are now collected (dataset), which are used as in the training phase. Therefore, the learning mode replaces conventional mapping with the output of the training phase in the subsequent learning periods. Consider the details.

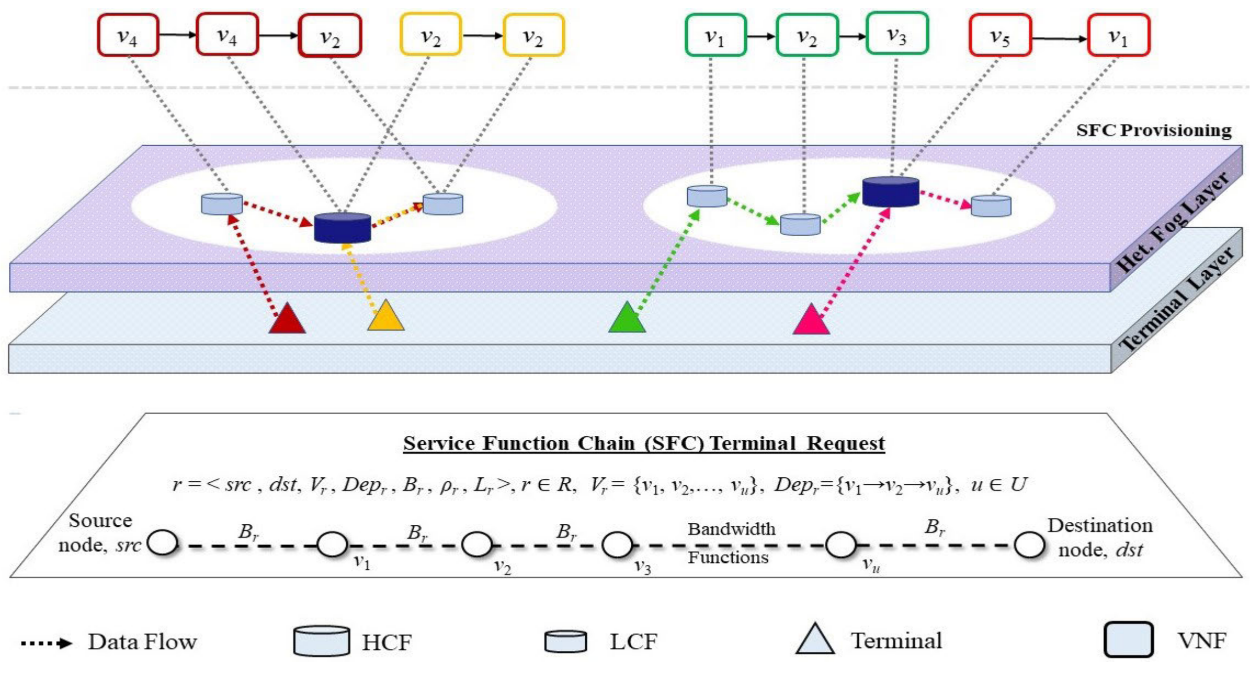
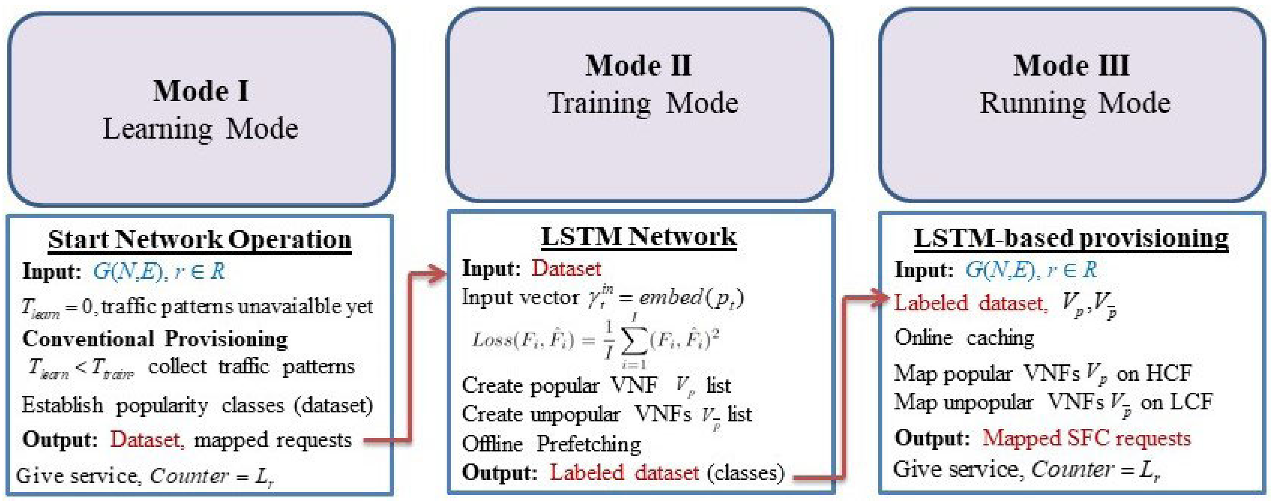


FIGURE 2. SFC provisioning concept with proposed demand model.



Mode I
Learning Mode
Mode II
Training Mode
Mode III
Running Mode
Start Network Operation
Input: $G(N,E), r \in R$
$T_{learn} = 0$, traffic patterns unavaialble yet
Conventional Provisioning
$T_{learn} < T_{train}$ collect traffic patterns
Establish popularity classes (dataset)
Output: Dataset, mapped requests
Give service, $Counter = L_r$
LSTM Network
Input: Dataset
Input vector $\gamma_t^{in} = embed(p_t)$
$Loss(F_i, \hat{F}_i) = \frac{1}{I}\sum_{i=1}^{I}(F_i, \hat{F}_i)^2$
Create popular VNF V_p list
Create unpopular VNFs $V_{\bar{p}}$ list
Offline Prefetching
Output: Labeled dataset (classes)
LSTM-based provisioning
Input: $G(N,E), r \in R$
Labeled dataset, $V_p, V_{\bar{p}}$
Online caching
Map popular VNFs V_p on HCF
Map unpopular VNFs $V_{\bar{p}}$ on LCF
Output: Mapped SFC requests
Give service, $Counter = L_r$

FIGURE 3. Different operating modes for the proposed LSTM-based SFC provisioning scheme.

Conventional SFC Provisioning: Since the scheme lacks traffic patterns at the start of the network operation. Therefore, the learning period is initialized ($T_{learn} = 0$). Hence the popular and unpopular lists are set to nulls, $V_{pop} = \{\}$, and $V_{\overline{p}} = \{\}$. Therefore, for each request, r, the scheme iterates and tries to map the VNFs in V_r and set up their interconnecting links.

Consider a terminal that transmits a request $r \in R$ to its closest fog node based on the highest signal power, Y_m^i, which can be either a HCF node of Y_{hc}^i captured power level or LCF node of Y_{lc}^i power level, i.e., $Y_m^i = max\{Y_{hc}^i, Y_{lc}^i\}$. For each request, a temporary variable is initialized to track total request delay, *Counter*, versus the request delay bound, ρ_r. Moreover, the closest fog node must possess sufficient resources to host the first VNF v_u in the request, $v_u \in V_r$. This is achieved by pruning the network to construct a temporary feasible graph *G'* composed of candidate nodes *N'* that possess sufficient processing and memory resources, i.e., $Q_{me}(n') > Q_{me}(v_u)$ and $Q_{pr}(n') > Q_{pr}(v_u)$, and candidate links E', with sufficient bandwidth, $B(e') > B_r$.

Once the network is pruned and the temporary feasible graph G' is constructed. Now the scheme selects a candidate node n' to map the VNF based upon two objectives. First, minimizing SFC delay. Second, improving load balancing across network nodes and links. Hence the objective function, φ, is defined as to achieve this goal, formulated as,

$$\varphi = min\big(D(src, n') + \Omega(n').(Q_{me}(n') + Q_{pr}(n')) + \Omega(e')(B(e'), \tag{4}$$

where $D(src, n')$ is the link delay between the source src and the candidate node n'. The variables $\Omega(n')$ and $\Omega(e')$ in order denote the node and link load weights, i.e., $\Omega(n') = |V_{map}(n')|/|V_r|$, where $|V_{map}(n')|$ is the total number of VNFs mapped onto the node and $|V_r|$ is the total number of VNFs in the request. Similarly, the link weight is computed $\Omega(e') = B(r)/B(e')$. The network here continuously monitors the load weights at each node and link, and dynamically updates them based upon the volume of the incoming traffic. For instant, when traffic is high at a particular node, then the weight is reduced to avoid node overloading. Meanwhile, when the node has low traffic density, then the weight is increased to accommodate more requests.

Now the scheme identifies a set of path (route) vectors, K between the src and the least-loaded candidate nodes. Then the scheme sorts these paths based on the shortest distance in an ascending order, and thereafter selects first node and link in the path, $k, k \in K$ that yields the shortest distance. Here if the $Counter \leq L_r$, then this node is selected as the hosting node $\hat{n}$ and link $\hat{e}$, at which the VNF is mapped and routed, respectively.

The hosting node $\hat{n}$ and link $\hat{e}$ are now added to the pruned network graph G'. This link is established from src to the node hosting the last VNF, $prev(prev \leftarrow n)$. The node and link resources are reserved now for this VNF, i.e., $Q_{me}(\hat{n} = Q_{me}(\hat{n}) - Q_{me}(v_u)$, $Q_{pr}(\hat{n}) = Q_{pr}(\hat{n}) + Q_{pr}(v_u)$ $B(\hat{e}) = B(\hat{e}) - B_r$.

For subsequent VNFs, the scheme again computes delay-bound candidate paths set, K, from $prev$ to the candidate node that maps the subsequent VNF, n', $n' \in N' - prev$, forming new set of vectors K. Each vector k satisfies minimum network delay for least-loaded nodes, i.e., $min\{D(n') + D(e')\} < \rho_r$. From these routes, a candidate node n' is selected based on minimum delay and least-load requirements, hence achieving minimum delay with previous node that hosts last mapped VNF, $prev$.

When all VNFs in request r are mapped, then r is flagged successful and data transmission phase is initiated for the terminal, while reserving resources for the entire lifetime period of the request, L_r. The $Counter$ records the elapsed time over which a request is in transmission phase. Once this time has reached the request lifetime, $Counter = L_r$, then data transmission phase is terminated and resources are released by updating hosting nodes and link available resources in $G' = (N', E')$. In notations, $Q_{me}(\hat{n}) = Q_{me}(\hat{n}) + Q_{me}(v_u)$, $Q_{pr}(\hat{n}) = Q_{pr}(\hat{n}) + Q_{pr}(v_u)$ and $B(\hat{e}) = B(\hat{e}) + B_r$. As a result, this setting relieves the limited resources in the nodes for use by other requests. In turn, network available resources is increased.

Note that if the candidate node in the cluster is resources deficient, then the VNF is relayed to the adjacent node in the cluster. Moreover, if no paths exist ($P = 0$) in the pruned network, then the request is dropped.

In the learning mode, nodes monitor the incoming requests and required VNFs, as well as creating dataset to model the traffic probability distribution over time. The fog nodes feed the current VNF at every time step to the deep learning model for use in the training mode, as detailed next.

B. MODE II: TRAINING MODE

The work here adopts a predictive framework that analyzes terminals demand patterns, based on a deep learning network, specifically, LSTM network, which is first proposed in [19]. This artificial recurrent neural network (RNN) is used to predict the VNF and its popularity class, derived from frequency and volume of incoming VNF types. This yields in a better utilization of resources in the substrate network, by alleviating traffic peaks when proactively mapping the popular VNFs on the HCF nodes in advance, and mapping the unpopular VNFs on the LCF nodes, prior to the requests arrival.

1) PROBLEM FORMULATION

The VNF prediction problem is formulated as a sequence labeling prediction of the requested VNF $v_u \in V_r$ at $(t+1)$th time step, $\hat{F}_i$, of Δ_{t+1} traffic volume, given the VNF status at the (t)th time step of Δ_t traffic volume, recorded over $i = 1, 2, \ldots, I$ observations. Hence the goal is maximize the probability of successfully predicting the incoming VNFs and their popularity classes, i.e., $\mathbb{P}[\hat{F}_i = F_i]$, while minimizing the least mean square error (MSE) and success rates, as presented later.

Here, the VNFs $v_u, \forall v_u \in V_r$ in the incoming requests are classified as either a popular VNF v_p, stored in the list V_p. Alternatively, they are classified as an unpopular VNF $v_{\overline{p}}$, stored in the list $V_{\overline{p}}$. These lists are determined based upon the observations of the popularity class $p, p = 1, 2, \ldots P$ over the I observations, where 1 and P indicate the least and highest popular VNF class, respectively. Namely, any VNF v_u in request r is deemed as popular, v_p if its frequency hits $\mathfrak{F}(v_u)$ exceeds a frequency threshold value, $\mathfrak{F}_{th}$, where this parameter highly impacts the aggressiveness configuration of the prefetching and caching procedures in the proposed scheme. Meanwhile, if the VNF frequency hits is less that $\mathfrak{F}_{th}$, then the VNF is classified as unpopular, $v_{\overline{p}}$. In notations,

$$v_u = \begin{cases} v_p, & \text{if } \mathfrak{F}(v_u) \geq \mathfrak{F}_{th}, \\ v_{\overline{p}}, & \text{otherwise.} \end{cases} \tag{5}$$

Now the key operation elements of the proposed reconfigurable deep learning model are composed of the input (initialization) and training phases. Consider the procedures of Mode II summarized in Figure 5.

```
1:  Input: Network $G = (N, E)$
    Request $r =< src, dst, V_r, Dep_r, B_r, \rho_r, L_r >$
2:  Output: $\gamma_t^{in}$, path vector $k$
3:  $T_{learn} = 0$ /* Learning Period */
4:  $V_{pop} = \{\}$, $V_{\overline{p}} = \{\}$ /* Null popular and unpopular lists */
5:  while $T_{learn} < T_{train}$ do
      Before network is trained
6:    Start conventional SFC provisioning $\forall r \in R$
7:    Iterate and process all incoming requests
8:    for (each $r \in R$) do
9:      $Counter = 0$ /* Initialize delay tracking variable */
10:     $K =$ /* Reset path vectors set */
11:     Iterate and map all $v_u \in V_r$ in request
12:     for (each $v_u \in V_r$) do
13:       Mapping Function()
14:       Prune network, build temporary graph $G'(N', E')$
15:       $N' \leftarrow$ Remove $n \in N$ with $Q_{me}(n) < Q_{me}(v_u)$ or
          $Q_{pr}(n) < Q_{pr}(v_u)$
16:       $E' \leftarrow$ Remove $e \in E$ with $B(e) < B_r$
17:       for (each $n' \in N'$) do
18:         Compute shortest paths between $src$ and least-
            loaded candidate nodes $N'$
19:         with $\varphi = min(D(src, n') + W(n')(Q_{me}(n') +$
            $Q_{pr}(n')) + W(e')(B(e'))$
20:         if $K = \{\emptyset\}$ then
21:           No feasible paths in $K$, drop request $r$
22:         end if
23:         Sort paths in $K$ by ascending order
24:         if $K \neq \{0\}$ then
25:           $k$=0 /* Choose first candidate path in list $K$ */
26:           Choose hosting node $\hat{n}$ and link $\hat{e}$
27:           $\hat{n} \leftarrow min(n' : \varphi(n'))$ /* choose first node in
              candidate path $k$ */
28:           $prev \leftarrow \hat{n}$ /* Set $prev \in$ route $k$ for request $r$ */
29:           Update tracking variable for request $r$
30:           $Counter = Counter$ + (proc. delay at $\hat{n}$) +
              (com. delay at $\hat{e}$)
31:           Reserve resources in $G'(N', E')$
32:           $Q_{me}(\hat{n}) = Q_{me}(\hat{n}) - Q_{me}(v_u)$
33:           $Q_{pr}(\hat{n}) = Q_{pr}(\hat{n}) + Q_{pr}(v_u)$
34:           $B(\hat{e}) = B(\hat{e}) - B_r$
35:         end if
36:       end for
37:     end for
38:   end for
39:   while $T_{learn} \geq T_{train}$ do
        Next learning round after network is trained Traffic re-
        ceived during Mode II $\rightarrow$ input vector $\gamma_t^{in}$
40:   end while
41: end while
42: Start Training Mode II
43: Offline Prefetching()
```

FIGURE 4. Mode I: Learning mode.

2) INPUT PHASE

This is the initialization phase that achieves labeled data, that finally yields the input vector γ_t^{in}. Namely, the input to the deep learning network is the current VNF v_u with p_t popularity index at the (t)th time step. Then the layers in the network start to map every VNF popularity index p_t to the input vector γ_t^{in}, based on the traffic patterns (dataset).

$$\gamma_t^{in} = embed(p_t), \tag{6}$$

where the function *embed* is a lockup table learnt during training.

```
1:  Input (Initialization) Phase:
2:  for each $T_{learn}$ do
3:    Collect traffic patterns
4:    Input vector $\gamma_t^{in} = embed(p_t)$
5:    if $T_{learn} < T_{train}$ then
6:      Learning Mode: Conventional SFC Provisioning
7:    else if $T_{learn} \geq T_{train}$ then
8:      Learning Mode: Traffic received during Mode II
9:      End of learning Mode
10:   end if
11: end for
12: Training Phase:
13: Output: Labeled dataset, $V_p$ and $V_{\overline{p}}$
14: Create $V_p$ and $V_{\overline{p}}$ lists
15: Call Offline Prefetching() Function
16: Running Mode: LSTM-based SFC Provisioning
```

FIGURE 5. Mode II: Training mode.

3) LSTM NETWORK STRUCTURE

LSTM is often insensitive to time gap length, as compared to conventional RNN, hidden Markov and other sequence learning models [20]. LSTM cell introduces a long-term memory, known as the cell memory state variable that is capable of learning long-term dependencies information of the network states. It focuses on the parameter information of previous periods and then labels the next, hence reducing the complexity of the prediction scheme. This presents a suitable prediction solution for time-series of variable sequences lengths, i.e., leveraged to predict future VNFs and their affiliated popularity classes. The LSTM network leverages parametric information of previous time steps and then labels the next to predict the VNFs over the $(t + 1)$th time step. Namely, the LSTM network recursively processes sequences (traffic volumes data) of variable lengths at every time step, t, of the input, and then maintains a hidden state which is a function of the previous state and the current input.

4) TRAINING PHASE

The training settings for the deep learning network includes four hidden layers and 50 LSTM units (cells) in each layer. The training process is now detailed. First the cell state at time t, ζ_t, which specifies information carried to the next sequence is modified by the forget gate, g_t^f, in the sigmoid layer placed underneath. In turn, it is then adjusted by the input modulation gate, g_t^{mod}, that delivers the new candidate cell state. The forget gate receives hidden state vector, h_{t-1}, (output vector of the LSTM cell) at the $(t-1)$th time step, and input vector at (t)th time step, γ_t^{in}, as its inputs. It produces an output number between 0 and 1 for each number in the previous cell state at the t$(t-1)$th time step, h_{t-1}. Namely, the output of the forget gate tells the cell state which information to forget or discard by multiplying 0 to a position in the matrix. Meanwhile, if the output of the forget gate is 1, then the information is kept in the cell state, where a sigmoid function, σ_g is applied to the weighted input/observation and previous hidden state. Eqs. (7)-(11) represent the cell state ζ_t, forget gate g_t^f, the input gate g_t^{in}, input modulation gate g_t^{mod}, and output gate (focus vector) g_t^{out}, all at the (t)th time step [19].

$$\zeta_t = g_t^f \zeta_{t-1} + g_t^{in} g_t^{mod}, \tag{7}$$
$$g_t^f = \sigma_g\big(\beta_f\big[h_{t-1}, \gamma_t^{in}\big] + \alpha_f\big), \tag{8}$$
$$g_t^{in} = \sigma_g\big(\beta_{in}\big[h_{t-1}, \gamma_t^{in}\big] + \alpha_{in}\big), \tag{9}$$
$$g_t^{mod} = tanh\big(\beta_\zeta\big[h_{t-1}, \gamma_t^{in}\big] + \alpha_\zeta\big), \tag{10}$$
$$g_t^{out} = \sigma_g\big(\beta_{out}\big[h_{t-1}, \gamma_t^{in}\big] + \alpha_{out}\big). \tag{11}$$

The parameters β_f, β_{in}, β_{out}, and β_ζ are the weight matrices and α_f, α_{in}, α_{out}, and α_ζ are the bias vectors for the forget, input, output gates, and the cell state, respectively (learnt during the training mode). Finally, the hidden state layer output h_t (working memory) is modeled as $h_t = g_t^{out}.tanh(\zeta_t)$. This hidden state yields the deep learning output during each time step t, encompasses the model learnt about the VNF popularity class unit this time step. Then the hidden state is used to predict the most likely VNF at the next $(t+1)$th time step.

Note that key parameters here for the model are the logistic *sigmoid* and the hyperbolic *tanh* nonlinear activation function for each gate to predict probability of the output. Foremost, the input gate is a *sigmoid* function with range $\in [0, 1]$ only adds memory, without the ability to forget memory, since the cell state equation is a summation between the previous cell state. Therefore, the input modulation gate is activated with a *tanh* activation function with a $[-1, 1]$ range that allows the cell state to forget memory.

After the deep learning network is well-trained, the fog nodes leverage it to predict the VNFs for new incoming requests, map recurring popular VNFs on the HCF nodes, and unpopular VNFs on the LCF nodes. At this stage, the network is well-trained and it has established the V_p and $V\overline{p}$ lists with minimum errors. Hence, it now performs offline pretching for these VNFs, as detailed next.

5) OFFLINE PREFETCHING

The network performs offline predictive prefetching (Figure 6) for the popular VNFs $v_p \in V_p$ on the LCF nodes, and unpopular VNFs $v_{\overline{p}} \in V_{\overline{p}}$ on the LCF nodes. Namely, these VNFs are migrated from the cloud domain to the fog nodes for faster retrieval by future terminals by providing a cached copy, provided by the VNF instance. Note that the VNF placement in the cache is based on the cache capacity at each node. Here the total cache capacity at each node must exceed the VNF instance s capacity requirements. This is expressed as,

$$\sum_{r\in R}\sum_{v_p\in V_p} \vartheta(n_{hc}).Q_{ch}(s) \leqslant Q_{ch}(n_{hc}), \tag{12}$$
$$\sum_{r\in R}\sum_{v_{\overline{p}}\in V_{\overline{p}}} \vartheta(n_{lc}).Q_{ch}(v_u) \leqslant Q_{ch}(n_{lc}), \tag{13}$$

where $\vartheta(n_{hc})$ and $\vartheta(n_{lc})$ denote the number of popular and unpolular VNFs cached in the HCF and LCF nodes, respectively.

```
1:  Input: $V_p$ and $V_{\overline{p}}$ lists, $N_{lc}$, $N_{hc}$
2:  Output: Offline prefetching of $V_p$ and $V_{\overline{p}}$ from cloud to fog
    nodes
3:  for each $\hat{n} \in N'$ do
4:    if $\hat{n} = n_{hc} \in N_{hc} \subseteq N'$ then
5:      Prefetch the unmapped popular VNFs on $n_{hc}$*
6:      $V_{pref}(n_{hc}) = V_p - \{V_{map} \cap V_p\}$
7:      $V_{pref}(n_{hc}) \rightarrow n_{hc}$
8:      Update resources on HCF node for each $v_p \in V_{pref}$
9:      for each $v_p \in V_{pref}$ do
10:       $Q_{me}(n_{hc}) = Q_{me}(n_{hc}) - Q_{me}(v_p)$
11:       $Q_{pr}(n_{hc}) = Q_{pr}(n_{hc}) - Q_{pr}(v_p)$
12:       Erase unpopular VNFs from HCF node
13:       $V_{er}(n_{hc}) = V_{map} - V_p$
14:       $n_{hc} \leftarrow$ Remove $v_{\overline{p}} \in V_{er}$
15:       Update resources for each removed VNF $\in$V$_{er}(n_{lc})$
16:       $Q_{me}(n_{hc}) = Q_{me}(n_{hc}) + Q_{me}(v_{\overline{p}})$ &$Q_{pr}(n_{hc}) =$
          $Q_{pr}(n_{hc}) + Q_{pr}(v_{\overline{p}})$
17:       Cache $v_{\overline{p}}$ on $n_{hc}$, update resources on $n_{hc}$
18:       $Q_{ch}(n_{hc}) = Q_{ch}(n_{hc}) - Q_{ch}(s)$
19:     else if $n_{lc} \in N_{lc} \subseteq N'$ then
20:       Prefetch the unmapped least popular VNFs on $n_{lc}$
21:       $V_{pref}(n_{lc}) = V_{\overline{p}} - \{V_{map} \cap V_{\overline{p}}\}$
22:       $V_{pref}(n_{lc}) \rightarrow n_{lc}$
23:       Update resources on LCF nodes for each VNF
          $\in$V$_{pref}(n_{lc})$
24:       Cache $v_{\overline{p}}$ on $n_{lc}$, and update resources on $n_{lc}$
25:       $Q_{ch}(n_{lc}) = Q_{ch}(n_{lc}) - Q_{ch}(s)$
26:       for each $v_{\overline{p}} \in V_{pref}(n_{lc})$ do
27:         $v_{\overline{p}} \rightarrow (n_{lc})$
28:         $Q_{me}(n_{lc}) = Q_{me}(n_{lc}) - Q_{me}(v_{\overline{p}})$
29:         $Q_{pr}(n_{lc}) = Q_{pr}(n_{lc}) - Q_{pr}(v_{\overline{p}})$
30:         $V_{er}(n_{lc}) = V_{map} - V_p$
31:         $n_{hc} \leftarrow$ Remove $v_u \in V_{er}(n_{lc})$
32:         Update resources on LCF for each removed VNF
            $v_p \in V_{er}(n_{lc})$
33:         $Q_{me}(n_{lc}) = Q_{me}(n_{lc}) + Q_{me}(v_p)$
34:         $Q_{pr}(n_{lc}) = Q_{pr}(n_{lc}) + Q_{pr}(v_p)$
35:       end for
36:     end if
37:   end for
38: end for
39: Start Running Mode: LSTM-based SFC Provisioning
```

FIGURE 6. Offline Prefetching() Function.

After the completion of Mode I, the hosting nodes now have accommodated various VNFs, regardless of their popularity class. Then the prefteching algorithm aims to gather the popular VNFs only on the HCF node, meanwhile collecting the unpopular VNFs on the LCF node. Namely, if the hosting node $\hat{n} \in N_{hc} \subseteq N'$ is a HCF node, $\hat{n} = n_{hc}$, then the goal is to prefetch the remaining popular VNFs that have not been mapped on this node during the learning mode. This is performed by creating a VNF preteching list, $V_{pref}(n_{hc})$, i.e., $V_{pref}(n_{hc}) = V_p - \{V_{map} \cap V_p\}$. Then the node processing and memory resources are updated for each $v_p \in V_{pref}$, $Q_{me}(n_{hc}) = Q_{me}(n_{hc}) - Q_{me}(v_p)$, $Q_{pr}(n_{hc}) = Q_{pr}(n_{hc}) - Q_{pr}(v_p)$.

After preteching these VNFs, they are now cached for use in the running mode. Namely, the cache resources at each HCF is subtracted by the VNF instance cache requirements, $Q_{ch}(n_{hc}) = Q_{ch}(n_{hc}) - Q_{ch}(s)$. Meanwhile, any previously mapped VNF that is not part of V_p, is erased from the HCF node. The erased VNFs are as $V_{er}(n_{hc}) = V_{map} - V_p$. Here for each erased VNF, the node resources are updated (i.e., releasing resources), $Q_{me}(n_{hc}) = Q_{me}(n_{hc}) + Q_{me}\left(v_{\overline{p}}\right)$ and $Q_{pr}(n_{hc}) = Q_{pr}(n_{hc}) + Q_{pr}\left(v_{\overline{p}}\right)$.

This process is similarly applied for prefetching the unmapped least popular VNFs $v_{\overline{p}}$ on the LCF node n_{lc}. Here the prefetching list is as $V_{pref}\ (n_{lc}) = V_{\overline{p}} - \{V_{map} \cap V_{\overline{p}}\}$. Then the processing and memory resources of the node are updated as, $Q_{me}(n_{lc}) = Q_{me}(n_{lc}) - Q_{me}\left(v_{\overline{p}}\right)$, and $Q_{pr}(n_{lc}) = Q_{pr}(n_{lc}) - Q_{pr}\left(v_{\overline{p}}\right)$, respectively, $\forall v_{\overline{p}} \in V_{pref}\ (n_{lc})$. The LCF node next erases any VNF $\notin V_{\overline{p}}$ that have been previously mapped in the learning mode. This is accomplished by creating the erasing list, $V_{er}(n_{lc})$, i.e., $V_{er}(n_{lc}) = V_{map} - V_p$. Thus releasing resources at the LCF node, $Q_{me}(n_{lc}) = Q_{me}(n_{lc}) + Q_{me}(v_p)$, $Q_{pr}(n_{lc}) = Q_{pr}(n_{lc}) + Q_{pr}(v_p)$.

C. MODE III: RUNNING MODE

This mode is initiated after traffic popularity modeling, and after which the deep learning model is now well-trained after Phase II. Namely, LSTM-based provisioning is conducted in this mode, presented and detailed in the pseudocode in Figure 7. Consider the details.

LSTM-Based SFC Provisioning: Following the training phase, the network now has established popularity classes and the V_p, $V_{\overline{p}}$ lists, derived from the training mode. These lists are periodically broadcasted by NC to all nodes in the cluster, where its exchanged. Nodes now establish cache tables, where each entry records a single VNF with its popularity class. Therefore, when a request is received by the node, its entire VNFs in V_r are examined from the cache table, in order to determine whether a VNF will be cached or mapped on that node.

The network now iterates and processes all incoming requests, $r \in R$. If the first VNF in the SFC, $v_u \in Dep_r \subseteq V_r$ exists in V_r, then this VNF is deemed as popular, i.e., $v_u = v_p$, where it has already been offline prefetched on the HCF nodes.

The network now prunes over all HCF nodes that possess sufficient cache capacity, i.e., removing $n_{hc} \in N$ with $Q_{ch}(n_{hc}) < Q_{ch}(s)$ and establishing N' candidate nodes, as well as removing $e \in E$ with $B(e) < B_r$ and building E'. Then, the scheme routes the VNF to the closest least-loaded HCF node, $min(n' : \varphi(n'))$, $n' = \forall n_{hc} \in N'$. Set this node as the hosting node, $(n_{hc}) = \hat{n}$, $prev \leftarrow \hat{n}$, $prev \in k$. Cache resources at the HCF hosting node $Q_{ch}(\hat{n})$ are now updated at $\hat{n}$, $Q_{ch}(\hat{n}) = Q_{ch}(n_{hc}) + Q_{ch}(v_p)$. These resources are reserved for the entire request lifetime, L_r. This process continues for the all popular VNFs in the request, $v_u \in V_p$, while taking into account shortest path with least-loaded HCF nodes.

Meanwhile, if any VNF in the request $v_u \in V_{\overline{p}}$, then it is deemed as unpopular $v_u = v_{\overline{p}}$. Then the network is pruned to identify the LCF node with sufficient cache resources. Thereafter, its routed to the the closest least-loaded LCF node with

```
1:  Output: SFC provisioning for $r \in R$
2:  Iterate and process all requests
3:  for (each $r \in R$) do
4:      Iterate and map all $v_u \in V_r$ in request
5:      for (each $v_u \in V_r$) do
6:          Check if the function is in function popularity list
7:          if $v_u \in V_p$ then
8:              $v_u = v_p$
9:              Prune over HCF nodes and build $G'(N', E')$
10:             $N' \leftarrow$ Remove $n_{hc} \in N$ with $Q_{ch}(n_{hc}) < Q_{ch}(s)$
11:             $E' \leftarrow$ Remove $e \in E$ with $B(e) < B_r$
12:             Route the VNF to the closest pruned HCF
13:             $min(n' : \varphi(n'))$, $n' = \forall n_{hc} \in N'$
14:             Set $(n_{hc}) = \hat{n}$ /* hosting HCF node */
15:             $prev \leftarrow \hat{n}$, $src, prev \in K$
16:             Reload $v_p$ from cache
17:             Update cache resources on the hosting HCF node
18:             $Q_{ch}(\hat{n}) = Q_{ch}(\hat{n}) + Q_{ch}(s)$
19:             Iterate over next VNF in the request $r$
20:         else if $v_u \in V_{\overline{p}}$ then
21:             $v_u = v_{\overline{p}}$
22:             Relay to LCF node
23:             Prune over LCF nodes and build $G'(N', E')$
24:             $N' \leftarrow$ Remove $n_{lc} \in N$ with $Q_{ch}(n) < Q_{me}(v_p)$
25:             $E' \leftarrow$ Remove $e \in E$ with $B(e) < B_r$
26:             Route the VNF to the closest pruned LCF node
27:             $min(n' : \varphi(n'))$,$n' = \forall n_{lc} \in N'$
28:             Set $(n_{lc}) = \hat{n}$ /* hosting LCF node */
29:             Update cache resources on the LCF hosting node
30:             $Q_{ch}(n\hat{n}) = Q_{cache}(\hat{n}) + Q_{ch}(v_{\overline{p}})$
31:         else if $v_u \notin V_p, V_{\overline{p}}$ /* $v_u \in V_r$ is a new function */ then
32:             Call Mapping Function()
33:         end if
34:     end for
35:     Start giving services for $Counter = L_r$
36:     Update $G'(N', E')$ node capacity and cache for mapped
        nodes
37:     Update $G'(N', E')$ link capacities along route $k$ for $r$
38: end for
```

FIGURE 7. Mode III: Running mode.

the *prev* node, i.e., $\varphi = min(D(prev, n') + W(n')(Q_{ch}(n') + W(e)(B(e'))$. This yields the hosting node for $v_{\overline{p}}$. This process continues for all $v_{\overline{p}}$ in request r.

Finally, if the requested VNF does not exist in the popular and unpopular lists, $v_u \notin V_p$, $V_{\overline{p}}$. Then this VNF is considered as entirely new (never mapped before). Therefore, it is then mapped using the conventional scheme, by calling the Mapping Function() used during the learning mode.

The hosting nodes here provide online caching for v_u. This results in significant advantages to the network. Foremost, reduced node loading, i.e., less processing and memory usage at the nodes. It also achieves less latency, since responses for cached VNFs are available immediately at close nodes to terminals. This accelerates services provided to the terminals. The caching process alleviates traffic bottleneck and congestion, saves resources and thereby enhancing network scalability and capacity without increased network cost.

VI. DEEP LEARNING SIMULATING RESULTS

The LSTM network is implemented on Python, i.e., attributed to the availability of libraries and open source tools. The

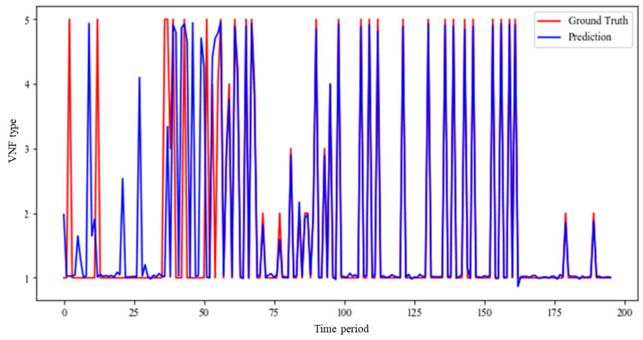


FIGURE 8. Prediction output of the LSTM network.

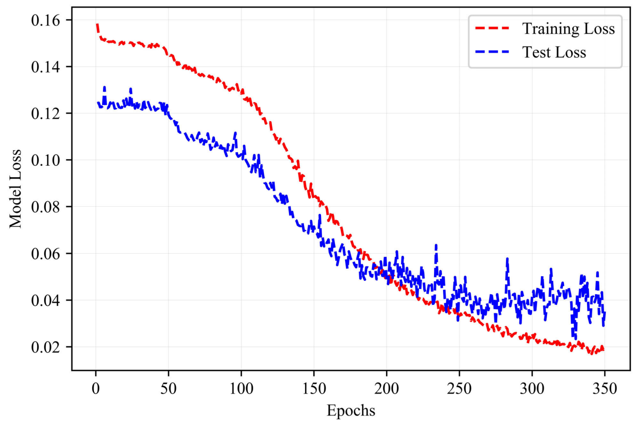


FIGURE 9. Model loss for the proposed LSTM network.

running platform leverages the Sequential, Dense, LSTM, and Dropout classes from the *keras.models* library, as well as the *MinMaxScaler* classes and the $feature_range$ parameter from the *sklear.preprocessing* library.

The proposed deep learning network model is now simulated using the dataset in [21] as part of the *BigDataChallenge* recorded over the period of two weeks. This dataset reveals terminals traffic volumes composed of 5 major activities (VNF types), i.e., SMS-in, SMS-out, call-in, call-out, and internet activities. Moreover, these VNFs are recorded over time steps inside a square ID of 200 meters, i.e., geographical grids that form a single cluster. The popularity class of each VNF here is determined by the activities of terminals in each cluster over a particular time step.

A. NETWORK TRAINING

As mentioned earliner, the LSTM network include four hidden layers, with 50 cells in each layer. The drop-out regularization rate in each layer used as a regulatizer is set at 0.2. The model is trained with 350 epoches over a duration that ranges from 200 minutes to 2 weeks. A data structure with 60 time steps, each of 10 minutes, and a single output is created, since LSTM cells store long term memory state. Hence in each training stage, there are 60 previous training set elements for each taken sample. Consequently, in the testing stage, the first 60 samples are needed a correct guess about the rest of the traffic distribution.

Overall, the training objective is to compute the *embd* function, weight matrices, and bias vectors that minimize the loss function for all training time instances. Figure 7 shows the prediction performance for the test set of the LSTM network. High approximation is achieved here between the prediction and the ground truth for the various VNF types over the training period (minutes). In particular after 75 minutes, where a high match is realized. See Table 1 for the overall model settings showing the hyper-parameters chosen for layers of the LSTM network.

B. LOSS FUNCTION

The objective is to achieve reduced loss function by applying the mean square error (MSE) between the prediction vector of the proposed model $\hat{F}_i$ and the actual ground truth F_i in the upcoming time step over I predictions generated from a sample of I data points on all variables. Note that both $\hat{F}_i$ and F_i features similar distribution over the same time period. This loss function model at every time step t, is modeled as,

$$Loss(F_i, \hat{F}_i) = \frac{1}{I}\sum_{i=1}^{I}(F_i, \hat{F}_i)^2. \tag{14}$$

Figure 9 shows the model loss (risk function) of the prediction network for one cluster, where low model losses (MSE) are achieved with the increased number of Epochs. The observations here can indicate a centralized data behavior (less skewed), i.e., high dispersion towards central moment. For instant, the model loss approaches 0.02 at 350 Epchoes for the cluster. Overall, the low MSE achieved here concludes that the proposed scheme/estimator can predict observations of the VNFs parameters with high accuracy.

C. VNF POPULARITY CLASS

A key characteristic of requests R is the high frequency demands for certain VNFs as modeled from traffic statistics in the dataset [21]. Thereby, VNFs popularity is present among different requests. The VNFs popularity class here accounts for the relative frequency of the VNFs and the relative obscurity of other VNFs as members of the requests (traffic population). It follows a power-law model distribution, $\Psi \sim Zipf(\mu, \Upsilon)$. Here Ψ is the popularity random variable, the variable μ denotes the skewness popularity parameter that models probability variation among VNFs types. Note that large μ represents a small amount of popularity (high diversity among distribution of terminals requests), i.e., highly right-skewed histogram distribution. Meanwhile, small μ (flat-skewed histogram) represents a high popularity. The variable Υ represents the total number of VNFs in the traffic, $\Upsilon = RU$.

The popularity of a VNF $v_u \in \Upsilon$ is defined as the number of traffic requests generated by N_m terminals requesting this VNF type, $N_m \to v_u$. Overall, the set of all VNFs that traffic generates is modeled as, $\Upsilon = v_u : \exists n_m \in N_m, N_m \to v_u$. Then, the popularity of a VNF v_u is defined as the number

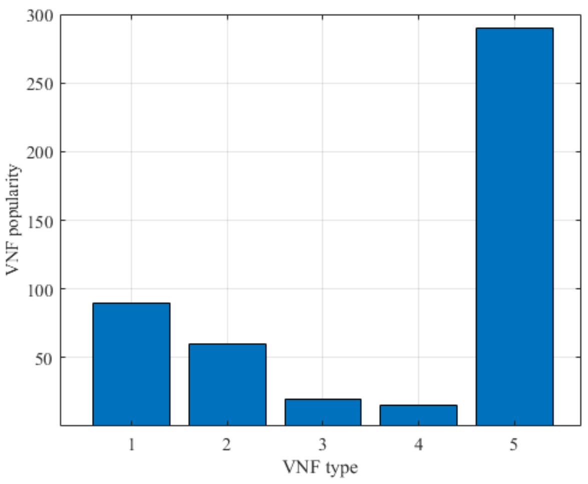


FIGURE 10. Popularity classes for the various VNF types.

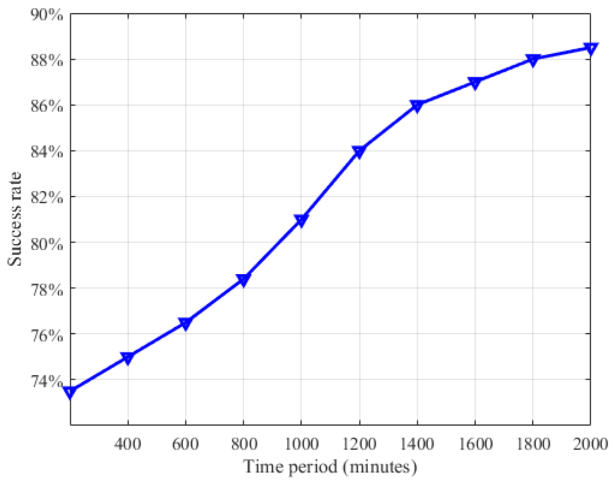


FIGURE 11. Success rates for the proposed LSTM network.

of terminals that generate trafunfic from, $P_{N_m}(v_u) = |O(v_u)|$, where $O(v_u) = n_m \in N_m, N_m \to v_u$. Moreover, the probability distribution over different VNFs and their associated popularities is defined by $p_{v_u}(\mu)$. For each VNF $v_u \in \Upsilon$, the probability that its popularity $P_{N_m}(v_u)$ is equal to μ is given by,

$$\mathbb{P}(\mu) = \frac{1}{\Upsilon} \sum_{v_u \in \Upsilon} \mu, \mu \in [0, \Upsilon]. \tag{15}$$

Figure 10 shows the popularity classes for the 5 VNF types, which corresponds to the VNFs with highest traffic densities. It is shown that VNF Type 5 yields the highest popularity class, i.e., 61.7% of the incoming traffic generates from terminals. This is followed by VNF Types 1 and 2 with 18.08% and 12.76 % of the overall generated traffic, respectively. Finally, VNF Types 3 and 4 feature low traffic density with 4.25% and 3.19%, respectively.

D. SUCCESS RATES

Finally, the prediction success rates are computed to evaluate the performance of the proposed LSTM scheme. Figure 11 plots the successful prediction probability versus extended time periods of increased dataset training sizes. It is obvious that the success probability of the proposed scheme is highly improved at increased time periods. This is attributed to the increase in the dataset sizes, which relate information on prior VNFs and their popularity rates. For example, the LSTM scheme achieves 72% success rates when dataset size is measured over 200 minutes. Then, the performance is gradually increased, until it reaches 93 % success rate at 2000 minutes. This in turn increases the robustness of the VNF prediction network.

VII. PERFORMANCE EVALUATION

The experiments are based on simulation models using Python programming language. Here a mesh network topology model is used, that takes into account realistic network settings and parameters. The network is composed on 25 HCF nodes and 58 LCF nodes, arranged in [5, 6] clusters, receiving SFC demands from [1,150] terminals. These details and other variable assignments are listed in Table 2.

The proposed heterogeneous fog (HF) architecture is now evaluated using the conventional and LSTM-based SFC provisioning schemes. Various key network performance metrics are considered. In particular, the number of successful (satisfied) requests over which service has been provided, the total network delay, and the energy consumption. Table 2 lists the various network parameters assignment.

A. NUMBER OF SUCCESSFUL REQUESTS

The SFC provisioning of an incoming request from a terminal is considered satisfied (successful), when all the request VNFs are mapped successfully on the network nodes. This includes satisfying the request delay bound B_r and resources requirements. When a request is successfully mapped, then the transmission plane is initiated for the entire lifetime period, L_r. This applies for all incoming requests R that are generated from terminals and received at the fog clusters by the HCF and LCF nodes. Note that incoming requests often yield in high traffic volumes of different delay and resources requirements, such as delay-sensitive, delay-tolerant, process-intensive. This in turn imposes high demands on the available resources on the fog nodes. Consequently, congested links and saturated nodes yield in denial of service to aggregated incoming requests after exceeding the network capacity. Therefore, these requests are dropped from the network, when the processing time exceeds the delay bound of the request, B_r. Therefore, SFC provisioning becomes more challenging for high traffic volumes of online incoming requests. This problem is further aggravated in conventional fog networks when mapping is solely conducted on the fog nodes without relaying to the cloud domain. Therefore, the proposed architecture here presents a potential for increased number of satisfied requests without the need to relay requests to the cloud domain. This is attributed to the

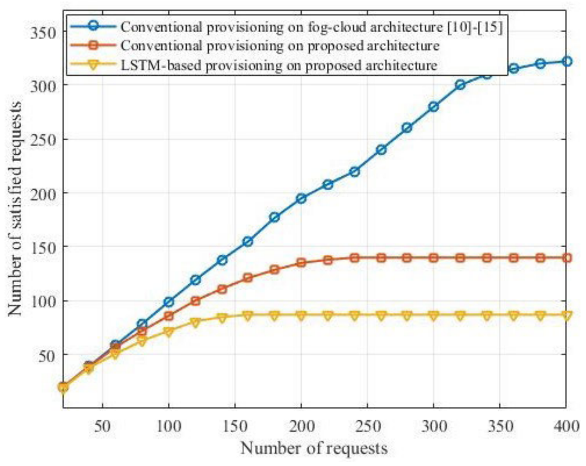


FIGURE 12. Number of satisfied requests for different architectures.

availability of the HCF nodes that possess abundant available resources in proximity to the terminals, at the edge network.

Figure 12 shows the number of satisfied requests per incoming requests mapped on the proposed heterogeneous fog (HF) architecture using conventional and LSTM-based provisioning schemes. The proposed LSTM-based scheme eliminates mapping redundant VNFs of the incoming requests over the network training period. This saves processing and memory resources at the fog nodes, which is now used more efficiently to accommodate new VNFs of new requests instead. This also increases the node-hit rate by serving requests directly from the cache on the fog node, while reducing the fetching from the cloud domain. Also, the abundant available resources at the HCF allows the network capacity to be further enhanced, by accommodating additional requests. This is opposed to the conventional scheme that continues to map the VNFs regardless of their popularity class, or whether they have already been mapped for a prior request.

For example, the LSTM-based provisioning scheme features approximately 90-97% success rates for the first 150 incoming requests, when implemented on the proposed HF architecture. Meanwhile, the conventional SFC provisioning scheme accommodates 85 and 95 requests out of the first 150 incoming requests, when mapping VNFs on the proposed heterogeneous fog and hybrid fog-cloud architectures in [10]-[15], respectively, i.e., approximately 56% and 65% success rates. It is observed here that the conventional scheme starts to saturate the network after 150 incoming success, which results in dropping subsequent incoming traffic from terminals. Note that the proposed architecture features reduced delay bounds B_r for the requests due to proximity of both HCF and LCF nodes to the terminals, as opposed to the upper level cloud nodes in the multi-tier hybrid architecture, which results in higher link propagation times, thus exceeding the delay bounds, thereby dropping the request.

Overall, the LSTM-based scheme features very low failure rate in the network for the first 300-330 requests. Then network congestion starts to have an impact on the admission ratio, i.e., since nodes are resources-constrained (realistic/practical setting). Saturation in the network for the proposed LSTM-based scheme occurs at approximately 400 requests, at which 350-360 requests are satisfied. Meanwhile, conventional provisioning schemes on hybrid fog-cloud architectures, as in [10]-[15] shows good admission ratio for the first 100 requests, Then the architecture starts to saturate at 200 requests, after which the network reaches its maximum nodes capacities, i.e., at approximately 300 incoming requests (160 requests are only satisfied).

Furthermore, the conventional scheme is also implemented on the proposed HF architecture, in order to demonstrate the efficiency of the proposed architecture. Here an increased admission ratio is achieved using the same conventional scheme as compared to mapping on hybrid fog-cloud architectures. This shows that the proposed HF architecture can accommodate more requests at reduced, delays, energy and cost (shown later). For example, for the proposed architecture can accommodate approx. 190 satisfied requests from 250 incoming requests, versus 150 requests for the hybrid architecture for the 250 incoming requests.

B. OVERALL NETWORK DELAY

For each request, packets encounter multiple types of network delay along their paths, as they transverse from the source node to the subsequent nodes along the path, unit reaching the destination node. In this work, the network delay is composed from the processing, transmission and propagation delays as presented next (queuing delay is not considered here).

Given a delay bound, ρ_r, the network delay along the designated path for request r, $D(k)$, must satisfy the delay constraint. This path delay is modeled as,

$$D(k) = \sum_{n' \in p} D_{pr}(n') + \sum_{e' \in p} D_{tran}(e') + \sum_{e' \in p} D_{prop}(n') \leqslant \delta_r, \tag{16}$$

$\forall r \in R$, where $D_{pr}(n')$, $D_{tran}(e')$ and $D_{prop}(e')$ represent the node processing delay, link transmission and propagation delays, respectively.

1) PROCESSING DELAY

This value measures the total time required by the node to process a mapped VNF, v_u, in the SFC of the request, r. The total processing time for R requests is formulated as,

$$D_{pr}(R) = \sum_{r \in R} \sum_{v_u \in V_r} \sum_{s \in S_{v_u}} A_r.1/\delta_{v_u}(n'), \forall n' \in N' - \{n_m\}, \tag{17}$$

where $\delta_{v_u}(n')$ denotes the processing rate of VNF type u in the request on a fog node, which is the request traffic unit per time (ms).

2) TRANSMISSION DELAY

It relates to the transmission rate of the link, i.e., amount of traffic units that are forwarded/transmitted from one node to another. Note that work here adopts first-come-first-serve transmission scheme. The transmission delay for all requests, $D_{tran}(\boldsymbol{R})$ is gauged as,

$$D_{tran}(R) = \sum_{r \in R} \sum_{e' \in E'} A_r.1/\delta(e'), \tag{18}$$

where A_r is the traffic load per request, and $\delta(e')$ is the link transmission rate, which is also the elapsed forwarding time per traffic unit for the request.

3) PROPAGATION DELAY

It represents the delay for data to propagate on link e between any two nodes, i.e., separation distance between the nodes divided by the propagation speed of the medium (e.g., wireless, fiber). The overall propagation delay for request r accounts for all interconnecting links between the source node, nodes hosting the VNFs to the destination node (propagation delay over all links joining the nodes at which VNFs are mapped). This delay is modeled as,

$$D_{prop}(R) = \sum_{r \in R} \sum_{e' \in E'} A_r.1/\delta(r). \tag{19}$$

The overall network delay here is defined as the time required by the VNFs to process incoming packets of various applications (requests under service), e.g., firewall, load balancer, and VPN function. This value is gauged by the processing time of the overall number of software-implemented VNFs at each fog node (listed in Table 2). Figure 13 shows the overall request delay using the proposed HF architecture, when mapping using the conventional and LSTM-based provisioning schemes, for the same number of satisfied requests.

The proposed LSTM-based scheme studies the VNFs popularity patterns, then it creates the popular and unpopular VNF lists. Then it performs offline pre-fetching for the popular VNFs on the HCF nodes and the unpopular VNFs on the LCF nodes. This pre-fetching enables the nodes to provide a cached copy of the VNF instance, instead of re-mapping it again for a different request. Consequently, this reduces

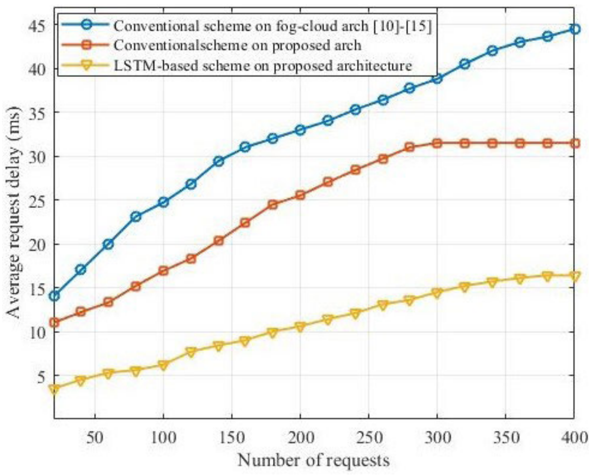


FIGURE 13. **Average request delay for different architectures.**

the number of used nodes, which in turn minimizes the processing times at the nodes. This is opposed to the conventional scheme that maps the same VNF repeatedly on different nodes, which yields in additional processing time at the hosting node. Along these times, the proposed LSTM-based scheme achieves reduced network delay at high number of requests versus conventional schemes.

For example, Figure 13 depicts that the average delay approaches 23 ms for 75 incoming requests when using the conventional scheme on hybrid fog-cloud architectures in [10]-[15]. This delay is further aggregated for higher incoming traffic, e.g., 31 ms for 150 requests. In contrast, the proposed HF architecture achieves reduced delays using the same SFC conventional scheme. Namely, the average request delay is reduced to 15 and 22 ms for the 75 and 150 incoming requests. This is due to the short propagation link delays between the terminals and nodes in the proposed HF architecture. Furthermore, when implementing the LSTM-based SFC provisioning scheme on the HF architecture, the average request delay is significantly reduced, i.e., 5 and 7 ms are achieved here for the 75 and 150 incoming requests. Therefore, the work here achieves approximately 77-80% reduction in average request delays as compared to mapping on multi-tier fog architecture. This is attributed to the short processing times at the HCF of popular VNFs, which have already been mapped in-advance. Also, the dense HCF can be used to support other requests that are in proximity to the terminals, at reduced propagation delays. Meanwhile, the conventional scheme saturates the close nodes much faster, hence suffering from relaying and handoffs to other clusters to host the VNFs. Consequently, packets traverse over long propagation distances over nodes separated by large geographical areas.

Note that network saturation occurs here as well, where satisfied requests yield the maximum delay when network is saturated. The network cannot admit any additional incoming requests and the delay at saturation is only for the same number of satisfied requests. For example, the conventional scheme satisfies 160 requests from 260 incoming requests, i.e., saturation point. Any additional incoming requests will yield in the same number of satisfied requests. Hence at 300 incoming requests, the same delay is encountered for the satisfied requests.

C. ENERGY CONSUMPTION

The energy consumption, Ξ, for the proposed architecture is defined as the total power consumption, W, during the entire network delay $D(r)$. It accounts for the power consumption levels at the candidate node $w(n')$, i.e., at the HCF, $w(n_{hc})$, the LCF, $w(n_{lc})$, and the NC, $w(n_{nc})$, in each cluster. In addition, it accounts for the power consumption for switch x, w_x, for X number of switches between the nodes. See Table 2 for parameters settings [22], [23]. In notations,

$$\Xi = D(r).W = D(r)(\sum_{n' \in N'} n'.w(n')).Xw_x, \tag{20}$$

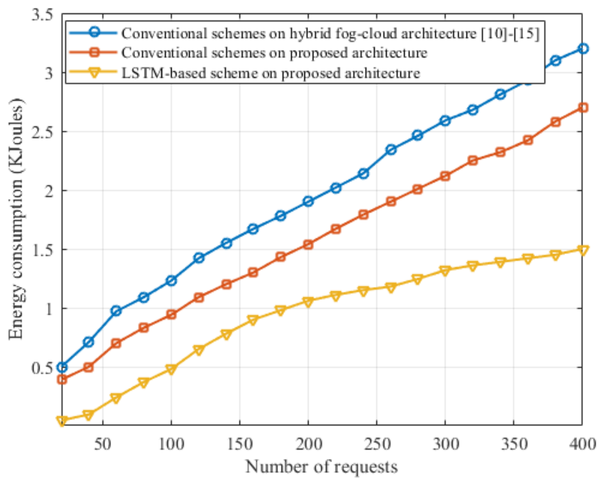


FIGURE 14. **Energy consumption levels for different architectures.**

where $w(n')$ is gauged by the power consumption rate in idle mode $\epsilon(n')$, the maximum power consumption for the candidate node $w(n')\big|_{max}$, and the saturation (utilization) factor at any node $\xi(n')\big|_{max}$, used during the SFC provisioning. This is expressed as,

$$w(n') = \epsilon(n').w(n')\big|_{max} + (1 - \epsilon(n')) + \zeta(n')\big|_{max}. \quad (21)$$

The energy consumption levels are depicted in Figure 14 for the conventional and LSTM-based provisioning schemes for the same number of satisfied requests. Results show that the proposed LSTM-based scheme consumes low energy levels compared to the conventional schemes at low and high number of satisfied requests. Here the hybrid fog-cloud architecture consumes 1.3 KJoules to map 150 requests. Meanwhile, the proposed architecture yields in 1 and 0.55 KJoules to map the same number of requests, using conventional and LSTM-based provisioning schemes, respectively. Note that energy consumption continues even after saturation. This is the energy requirements for providing service for the requests for their entire service lifetime. Figure 14 also shows that the LCF nodes reach saturation (100 utilization) after 350 requests, $\xi(n_{lc})\big|_{max} = 1$, whereas the HCF reaches it at 500 quests, i.e., $\xi(n_{hc})\big|_{max} = 1$. This demonstrates the advantages of the proposed HF architecture using the LSTM-based deep learning method, where a smaller number of nodes are used versus the conventional mode to accommodate the same number of satosfied requests. Thereby, less power and energy consumption levels are achieved.

D. OVERALL NETWORK COST

The overall realization cost of the SFC provisioning in different (i.e., mutlti-tier and heterogeneous) fog architectures comprises the deployment, processing, and communication costs. Consider the details.

1) DEPLOYMENT COST

The total license cost of deploying VNF software instances, modeled as,

$$C_{dep}(R) = \sum_{r\in R}\sum_{v_u\in V_r}\sum_{s\in S_{v_u}} \lambda^r_{v_u}.c(v_u), \forall n' \in N' - \{n_m\}, \quad (22)$$

where $c(v_u)$ is the license cost of VNF type u.

2) PROCESSING COST

The cost of resources assigned and reserved for the overall number of mapped VNFs in the SFC requests, expressed as,

$$C_{pr}(R) = \sum_{r\in R}\sum_{v_u\in V_r}\sum_{s\in S_{v_u}} c_{pr}(n)Q_{pr}(v_u) + c_{me}(n')Q_{me}(v_u), \quad (23)$$

$\forall n' \in N' - \{n_m\}$, where $c_{pr}(n')$ and $c_{me}(n')$ are the node processing and memory costs per resource unit, respectively.

3) COMMUNICATION COST

The total cost of edges assigned and used for all the mapped VNF edges in the requests. This includes communication cost between terminals and their affiliated VNFs on fog and/or cloud nodes. This is modeled as,

$$C_{com}(R) = \sum_{r\in R}\sum_{e'\in E'} \lambda^r_{v_u}.c(e').B_r, \quad (24)$$

where $c(e')$ accounts for the transmission cost per traffic unit between links in different layers. See Table 2 for different edge costs.

Figure 15 shows the overall network cost for different incoming requests onto the different architectures, while taking into consideration the resources saturation. Here the proposed LSTM-based provisioning scheme yields minimized cost for the proposed HF architecture, e.g., 38 units are requires to accommodate 150 requests. Also, the HF architecture still yields less cost (55 units) when using conventional scheme as opposed to the hybrid fog-cloud architecture (70 units). The LSTM-based scheme adopts caching policy based on the popularity class for the incoming VNFs. This results in less processing and memory resources requirements. Therefore, less number of nodes are used to accommodate the same number of satisfied requests. Consequently, reducing the overall network cost.

When the network starts to saturate, then the number of satisfied requests stops at a certain level. Then the architecture cannot accommodate any additional incoming requests. Therefore, it is obvious that the cost becomes stable at saturation. Note that the straight lines at saturation indicate the cost for the satisfied requests, regardless of the incoming requests. For example, the conventional scheme is already saturated at 300 incoming requests with 160 satisfied requests. These 160 requests have a service cost of 78. Similarly at 400 incoming requests, where again only 160 requests can be accommodated, which they have the same cost of 78.

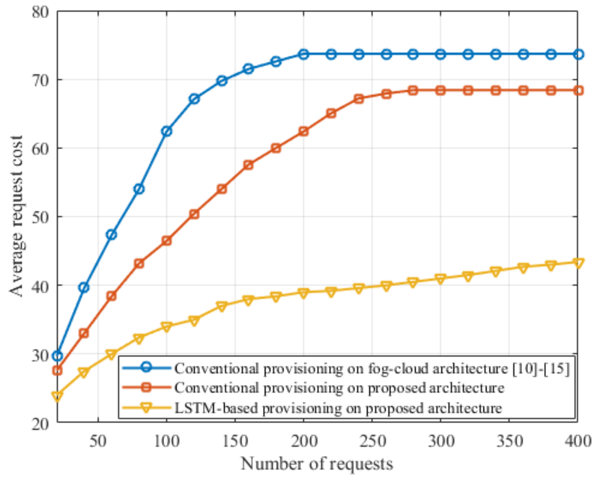


FIGURE 15. Average request cost for different architectures.

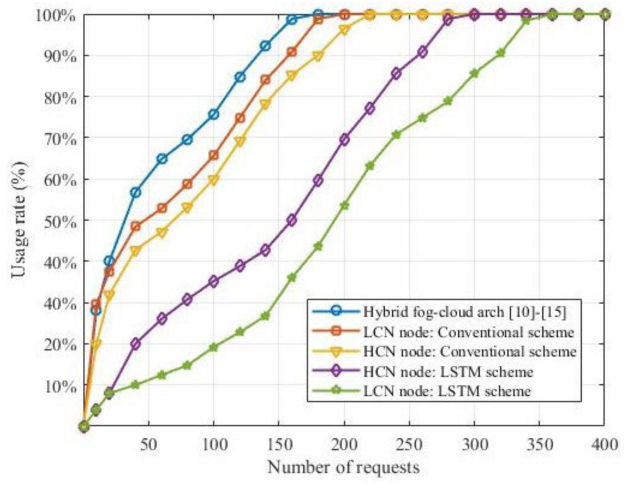


FIGURE 16. Average nodes usage rates in the proposed architecture.

The same analysis applies for the LSTM-based provisioning scheme. It can be noted that the proposed LSTM-based scheme significantly reduces the service cost as compared to conventional scheme, i.e., approximately 51-57% reduced cost at 200-400 requests.

E. NODE USAGE

Figure 16 shows the usage rate (%) of the HCF and LCF nodes under different provisioning schemes and architecture models. Note that work here adopts a resources-constrained network in order to demonstrate realistic operating setting, i. e., nodes and links have limited capacities. Therefore, saturation occurs when the nodes are full in-capacity. First, the hybrid fog-cloud architecture in [10]-[15] reaches 100% usage rate (all cloud and fog nodes become full) around 150-160 requests. Meanwhile, when performing the conventional provisioning scheme on the proposed architecture, the 100% usage rate is at 175 and 220 requests for the LCF and HCF nodes, respectively. Note that the LCF nodes here reach 100% usage earlier than the HCF nodes, as the number of the LCF nodes is higher than the number of the HCF node in each cluster. Therefore, the terminals can be at closer proximity to the LCF nodes, i.e., minimum path delays. Also, the mapping on the LCF and HCF nodes achieves load balancing between the nodes, which makes the LCF saturate earlier due to the low processing and storage capacities. Meanwhile, the usage rate for the proposed LSTM-based scheme is at 290 requests and 360 requests for the HCF and LCF nodes, respectively. Here the HCF node reaches 100% usage rate earlier than the LCF nodes, as approx. 85% of the incoming VNF types are popular, i.e., mapped on the HCF nodes, based on the traffic patterns (VNF Types 1 and 5). The LCF node reaches full usage rate later, as it is used to host the unpopular (least frequent) VNF types (e.g., VNF Types 2, 3 and 4). When the HCF reaches 100% usage rate, then the LCF node is used to map the new satisfied request. It is important to observe the impact of the training process and the incoming traffic highly impact the nodes saturation, i.e., based on the popularity classes and the number of the VNFs in the incoming requests.

It is obvious now that the aforementioned provisioning schemes in this work are implemented on the proposed HF architecture, which is composed of multiple LCF nodes and a single HCF node in each cluster. It is also interesting to study the proposed SFC provisioning scheme on the heterogeneous fog (HF) architecture when composed of a single LCF node and multiple HCF nodes (as opposed to the current model). When the number of HCF nodes is higher here, then the network capacity can significantly increase, more requests can be satisfied at reduced network delays. This also applies for different VNFs popularity/unpopularity distributions. However, the increased network capacity occurs at the detriment of increased cost and energy consumption, since a single HCF node consumes more energy and cost as compared to a single LCF node. Overall, the implementation of the nodes in the network can take different forms based on the operators strategies, taking into account also the CAPEX and OPEX.

Moreover, when the HCF node are unavailable (due to saturation, failure or extended delay), then the scheme switches to the LCF nodes in the cluster. It selects the LCF node that achieves the least network delay and possesses the highest available resources. The LCF nodes act as a back-up to host the incoming requests. The same procedure applies when the LCF nodes are unavailable due to cluster failure or saturation (based on the traffic pattern). Note that additional HCF nodes can be added to the architecture based on the scalability of the network demands and the infrastructure of the operators and coverage areas.

VIII. CONCLUSION

This paper presents a comprehensive study on deep learning for service function chain provisioning for network function virtualization in fog computing. The provisioning scheme is implemented on a novel heterogeneous fog architecture, composed of high- and low- capacity fog nodes of different resources capacities. The scheme performs conventional mapping, during which the network is well trained to predict the upcoming VNF in the next time step. Namely, a long short-term network is leveraged to predict incoming VNFs popularity classes and volumes. This creates dataset for prefetching and caching future requests, as an alternative to conventional provisioning. The proposed work achieves reduced network delays, reduced power and energy consumption and less costs, as well as increased capacity by accommodating additional new requests. Moreover, the deep learning network yields high success rates at reduced loss model. Future efforts will investigate the deep learning network for failure restoration, towards self-heeling networks, where the proposed architecture can be scaled to include back-up nodes and protection links.

REFERENCES

[1] M. Armbrust, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] N. Siasi and A. Jaesim, "Priority-aware SFC provisioning in fog computing," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2020, pp. 1–6.

[3] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, Jun. 2017.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[5] R. Chaudhary, N. Kumar, and S. Zeadally, "Network service chaining in fog and cloud computing for the 5G environment: Data management and security challenges," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 114–122, Nov. 2017.

[6] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, Ottawa, ON, Canada, May 2015, pp. 1202–1207.

[7] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1826–1857, 3rd Quart., 2018.

[8] Y. Qiu, H. Zhang, K. Long, H. Sun, X. Li, and V. C. M. Leung, "Improving handover of 5G networks by network function virtualization and fog computing," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Qingdao, China, Oct. 2017, pp. 1–5.

[9] J. Liu, S. Zhou, J. Gong, Z. Niu, and S. Xu, "Graph-based framework for flexible baseband function splitting and placement in C-RAN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 1958–1963.

[10] S. Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. Ben Hadj-Alouane, M. J. Morrow, and P. A. Polakos, "A platform as-a-service for hybrid cloud/fog environments," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Rome, Italy, Jun. 2016, pp. 1–7.

[11] O. Bibani, C. Mouradian, S. Yangui, R. H. Glitho, W. Gaaloul, N. B. Hadj-Alouane, M. Morrow, and P. Polakos, "A demo of IoT healthcare application provisioning in hybrid cloud/fog environment," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Luxembourg City, Luxembourg, Dec. 2016, pp. 472–475.

[12] R. Deng, R. Lu, C. Lai, and T. H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 3909–3914.

[13] D. Rosário, M. Schimuneck, J. Camargo, J. Nobre, C. Both, J. Rochol, and M. Gerla, "Service migration from cloud to multi-tier fog nodes for multimedia dissemination with QoE support," *Sensors*, vol. 18, no. 2, p. 329, Jan. 2018.

[14] C. Mouradian, S. Kianpisheh, and R. H. Glitho, "Application component placement in nfv-based hybrid cloud/fog systems," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Washington DC, USA, Jun. 2018, pp. 25–30.

[15] E. S. Gama, R. Immich, and L. F. Bittencourt, "Towards a multi-tier fog/cloud architecture for video streaming," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, Zurich, Switzerland, Dec. 2018, pp. 13–14.

[16] H. R. Khezri, P. A. Moghadam, M. K. Farshbafan, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, "Deep reinforcement learning for dynamic reliability aware NFV-based service provisioning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Puako, HI, USA, Dec. 2019, pp. 1–6.

[17] H. Rahmani Khezri, P. Azadi Moghadam, M. Karimzadeh Farshbafan, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, "Deep Q-Learning for dynamic reliability aware NFV-based service provisioning," 2018, *arXiv:1812.00737*. [Online]. Available: http://arxiv.org/abs/1812.00737

[18] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 263–278, Feb. 2020.

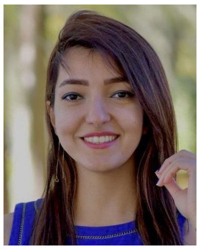
[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] S. Hochreiter and J. A. Schmidhuber, "Lstm can solve hard long time lag problems," in *Proc. 9th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Cambridge, MA, USA, Dec. 1996, pp. 4793–4796.

[21] G. Barlacchi "A multi-source dataset of urban life in the city of Milan and the province of Trentino," *Sci. Data*, vol. 2, no. 1, pp. 1–5, Feb. 2015.

[22] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Energy efficient algorithm for VNF placement and chaining," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, Madrid, Spain, May 2017, pp. 98–106.

[23] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, "Energy-efficient data replication in cloud computing datacenters," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Shenzhen, China, Dec. 2013, pp. 385–402.



NAZLI SIASI (Member, IEEE) received the B.S. degree in information technology and computer engineering from Tehran Polytechnic University, the M.S. degree in information technology and computer engineering from Tabriz State University, and the M.S. and Ph.D. degrees in electrical engineering from the University of South Florida, Tampa, FL, USA. She is currently an Assistant Professor of cybersecurity with Christopher Newport University, Newport News, VA, USA. Her research interests include networking, virtualization, and fog and cloud computing.

MOHAMMED JASIM (Member, IEEE) received the bachelor's degree in electrical engineering from Applied Science University, Jordan, the master's degree in electrical engineering from Brunel University London, U.K., and the Ph.D. degree in electrical engineering from the University of South Florida, USA. He was with Valparaiso University, Valparaiso, IN, USA. He is currently an Assistant Professor of electrical engineering with the University of Mount Union, Alliance, OH, USA. His research interests include millimeter wave communications, beamforming, and fog and cloud computing.

ADEL ALDALBAHI (Member, IEEE) received the B.S. degree in electrical engineering from Virginia Commonwealth University, Richmond, VA, USA, in 2011, and the M.S. and Ph.D. degrees in electrical engineering from the New Jersey Institute of Technology, Newark, NJ, USA, in 2013 and 2017, respectively. He joined with the Electrical Engineering Department, King Faisal University, as an Assistant Professor, in 2018. His research projects are funded by King Faisal University and Ministry of Higher Education (MOHE). His current research interests include visible light communication, wireless networks, modeling, analysis, and millimeter wave for 5G and beyond. He served as a TPC for ICWMC 2019 and 2020, as a Reviewer for IEEE Access, the IEEE Photonics, and the IEEE ISSPIT 2018. He served as a Session Chair for WIMOB 2018 and 2019, ICCAIS' 2019. He served as a Publicity Chair for WIMOB 2020.

NASIR GHANI (Senior Member, IEEE) received the bachelor's degree in computer engineering from the University of Waterloo, the master's degree in electrical engineering from McMaster University (supervisor Dr. Simon Haykin, Life Fellow, IEEE), and the Ph.D. degree in computer engineering from the University of Waterloo (supervisor Dr. Jon. W. Mark, Life Fellow, IEEE). He was an Associate Chair with the ECE Department, University of New Mexico. He was a Faculty Member with Tennessee Tech University. He is currently a Professor with the Electrical Engineering Department, University of South Florida. He is also a Research Liaison with Cyber Florida. He also spent several years working with large Blue Chip organizations (IBM, Motorola, Nokia) and several hi-tech startups. His research has been supported by the National Science Foundation, Defense Threat Reduction Agency, Department of Energy, Qatar Foundation, and Sprint-Nextel. He has published over 200 peer-reviewed publications. His research interests include cyberinfrastructure networks, cybersecurity, cloud computing, disaster recovery, and cyber-physical systems. He received the NSF CAREER Award, in 2005, and the Best Paper Awards at IEEE PIMRC 2017 and IEEE ANTS 2010. He has chaired symposia for the IEEE GLOBECOM, the IEEE ICC, the IEEE ICCCN, and workshops for the IEEE INFOCOM. From 2007 to 2010, he was also the Chair of the IEEE Technical Committee on High Speed Networking (TCHSN). He has served as an Associate Editor for the IEEE/OSA Journal of Optical and Communications and Networking, the IEEE Systems, and the IEEE Communications Letters. He has also edited special issues of the IEEE Network and the *IEEE Communications Magazine*.