Topology-Aware Prediction of Virtual Network Function Resource Requirements

Rashid Mijumbi, Sidhant Hasija, Steven Davy, Alan Davy, Brendan Jennings, *Member, IEEE*, and Raouf Boutaba, *Fellow, IEEE*

***Abstract*—Network functions virtualization (NFV) continues to gain attention as a paradigm shift in the way telecommunications services are deployed and managed. By separating network function from traditional middleboxes, NFV is expected to lead to reduced capital expenditure and operating expenditure, and to more agile services. However, one of the main challenges to achieving these objectives is how physical resources can be efficiently, autonomously, and dynamically allocated to virtualized network function (VNF) whose resource requirements ebb and flow. In this paper, we propose a graph neural network-based algorithm which exploits VNF forwarding graph topology information to predict future resource requirements for each VNF component (VNFC). The topology information of each VNFC is derived from combining its past resource utilization as well as the modeled effect on the same from VNFCs in its neighborhood. Our proposal has been evaluated using a deployment of a virtualized IP multimedia subsystem, and real VoIP traffic traces, with results showing an average prediction accuracy of 90%, compared to 85% obtained while using traditional feed-forward neural networks. Moreover, compared to a scenario where resources are allocated manually and/or statically, our technique reduces the average number of dropped calls by at least 27% and improves call setup latency by over 29%.**

***Index Terms*—Network functions virtualisation, dynamic resource allocation, topology-awareness, prediction, machine learning, graph neural networks, virtual network functions.**

I. Introduction

SERVICE provision in the telecommunications industry has traditionally been based on the use of specialised Network Appliances (NAs) for each NF. This tight coupling usually means that even slight changes in the operation of a given Network Function (NF) could necessitate replacement of the (NA) on which it runs. This short lifetime of the NAs leads to increased Capital Expenditure (CAPEX). In addition, the fact that NAs are specialised calls for specialised maintenance and limits flexibility, leading to increased Operating Expenditure (OPEX). These issues, combined with the need for strict adherence to regulations, stringent network stability and service quality targets, usually lead to extended product development cycles. Moreover, due to the fierce and ever increasing competition from services provided over-the-top, Telecommunications Service Providers (TSPs) have found themselves with consistently reducing average revenue per user, and therefore declining profitability. Therefore, TSPs are faced with an urgent need to find innovative and less expensive ways to increase and/or efficiently utilise network capacity and functionality, and achieve better service agility.

NFV [1], [2] has been proposed as a possible path towards this end. The main idea of NFV is to take advantage of recent advances in virtualisation technologies to decouple NFs (e.g., firewalls, load balancers) from dedicated NAs so as to run them in generic servers that may be located in datacenters or at centralised TSP points of presence. Thanks to NFV, different NFs can evolve independently of each other, and of hardware. Furthermore, by running VNFs in virtualised resources (e.g., Virtual Machines (VMs)), network resources can be efficiently allocated through dynamic scaling. Finally, NFV promises to lead to more efficient operations through automated and centralised management of networks and services. Because of these prospects, NFV has been identified as an essential enabler of the fifth generation (5G) of communication networks [3]. This means that in the near future, VNFs will be building blocks for critical application domains such as smart grid, smart cities, connected automotive, and eHealth.

However, NFV is still in infancy and making its anticipated gains a reality still requires overcoming a number of challenges. One of the most important of these challenges relates to efficiently and autonomously managing resources that are allocated to VNFs while also ensuring reliability of services that run on them [4]. Reliable and efficient resource management is especially important for the future 5G networks which will require a very high amount of resources, and whose expected applications (such as connected vehicles) will demand highly reliable networks. Specifically, there is need for algorithms to determine how resources from the Network Functions Virtualisation Infrastructure (NFVI) are shared among the VNFs. These algorithms should have capabilities of scaling VNF resources vertically and/or horizontally while meeting two conflicting objectives. On one hand, VNFs

Manuscript received November 30, 2016; accepted February 7, 2017. Date of publication February 9, 2017; date of current version March 9, 2017. This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) and is co-funded under the European Regional Development Fund under Grant Number 13/RC/2077. The associate editor coordinating the review of this paper and approving it for publication was H. Lutfiyya.

R. Mijumbi is with Bell Labs CTO, Nokia, D15 Y6NT Dublin, Ireland (e-mail: rashid.mijumbi@nokia.com).

S. Hasija, S. Davy, A. Davy, and B. Jennings are with the Telecommunications Software and Systems Group, Waterford Institute of Technology, X91 K0EK Waterford, Ireland (e-mail: rashid.mijumbi@nokia.com).

R. Boutaba is with the D. R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada.

Digital Object Identifier 10.1109/TNSM.2017.2666781

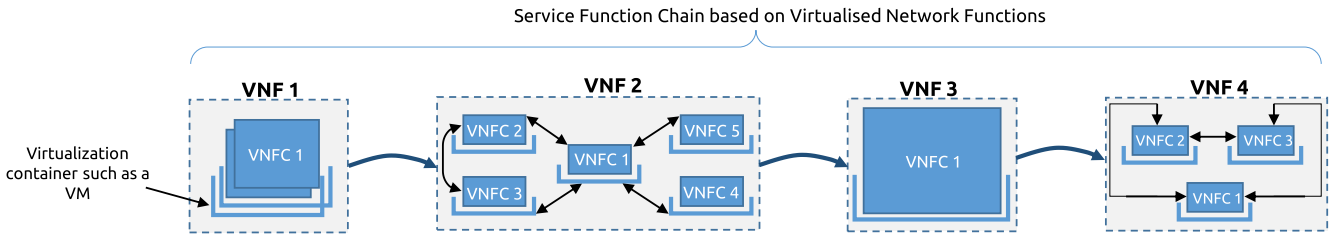

Fig. 1. NFV service function chain. VNF 1 has a single VNFC while VNF 2 has multiple VNFCs. The VNFCs may be horizontally or vertically scaled. While VNFs are connected to each other by directed links in a chain, the VNFCs may contain both directed and un-directed links, in a vendor-specific topology.

should be allocated enough resources at all times to meet service quality requirements (i.e., reliability). On the other hand, only the needed amount of resources should be allocated to the VNFs to ensure efficiency. Given that network traffic and hence the load of such VNFs vary over time, and since spinning-up new resources (horizontal scaling) may take some time (in case the VNFs run in VMs), there is need for an automated way of determining such resource needs ahead of time so that resources are available when needed, without causing system outages or inefficiently using them.

In this article, we propose a topology-aware, dynamic and autonomous system for managing resources in NFV based on the concept of GNNs [5]. Our proposal is motivated by the fact that in NFV, services are composed of one or more VNFs arranged in a specific order to create what is known as a Service Function Chain (SFC). Network traffic traverses the VNFs in a given SFC sequentially. This implies that resource requirements of a given VNF may be predicted by observing those of other VNFs in the chain. Therefore, the proposed GNN approach involves modelling each VNFC in a SFC as two parametric functions, each implemented by a Feedforward Neural Network (FNN). The task of each pair of FNNs representing a given VNFC is to learn (in a supervised way) the trend of resource requirements of the VNFC. This is achieved by combining historical local VNFC resource utilisation information with the information collected from its neighbours to forecast future resource requirements of the VNFC. In particular, the first FNN expresses the dependence of the resource requirements of each VNFC on the resource requirements of VNFCs in its neighbourhood. This is input into the second FNN which forecasts the resource requirements of the VNFC. The resource requirement forecast is in turn used to automatically spin-up and configure new VNFCs or turn them off as required. To the best of our knowledge, dynamic and automated management of resources in NFV is still an open research problem, and learning techniques based on artificial intelligence are particularly interesting possible solutions.

This article extends the results presented in our earlier paper [6] in several ways. First, we give a more detailed motivation of the proposed approach by use of recent results regarding the time needed to spin up VNFs in the most commonly used Virtual Infrastructure Managers (VIMs). We also give a more realistic example of a SFC early on in the article. We have also improved the description of the state computation step as well as the learning process. Most significantly, we have performed extra experiments to compare our proposal with a simple FNN approach, and to evaluate the performance with different changes in a number of parameters. Acronyms that have been frequently used in the paper are summarised in Table I.

TABLE I
SUMMARY OF ACRONYMS FREQUENTLY USED IN THE ARTICLE

Acronym	Description
5G	Fifth Generation
CAPEX	Capital Expense
ETSI	European Telecommunications Standards Institute
FNN	Feedforward Neural Network
GNN	Graph Neural Network
HOT	Heat Orchestration Template
IMS	IP Multimedia Subsystem
MANO	Management and Orchestration
NF	Network Function
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NN	Neural Network
OPEX	Operating Expense
SFC	Service Function Chain
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtualised Network Function
VNFC	Virtual Network Function Component
VNF-FG	Virtual Network Function Forwarding Graph

The article is organised as follows. We describe the problem in Section II and introduce GNNs in Section III. The proposed GNN-based resource allocation model and the corresponding learning algorithm are detailed in Sections IV and V respectively. Our proposal is evaluated in Section VI, related work discussed in Section VII, and the article concluded in Section VIII.

II. Problem Description and Motivation

The delivery of end-to-end services often requires packets, frames, and/or flows to traverse an ordered or partially ordered set of abstract NFs in what is known as an SFC. In NFV, such NFs are deployed in virtualised resources, and are hence known as VNFs. An example of such a SFC is shown in Fig. 1, in which the SFC is composed of 4 VNFs each connected to others by a directed link. Each VNF may be composed of one or more VNFCs, each hosted in a virtualisation container (virtual machines, linux containers, etc.).

A. Virtual Network Function Components

Big instances of VNFs – such as an IMS – may be composed of smaller instances of software components, called VNFCs. A VNFC is an internal component of a VNF which provides a defined sub-set of that VNF's functionality, with the main characteristic that a single instance of this component maps 1:1 against a single virtualisation container [7]. According to the European Telecommunications Standards Institute (ETSI), while VNF implementations must be standard and hence expose standard interfaces, VNFC implementations may be VNF provider specific. The ability to split VNFs into smaller VNFCs gives providers the flexibility with regard to a number of factors [7], e.g., the prioritization of performance, scalability, reliability, security and other non-functional goals; the integration of components from other VNF Providers; operational considerations; the existing code base, etc. This means that even for a VNF with the same functionality, different VNF providers might have a different number and topology of constituent VNFCs. The VNFCs in a VNF are linked to each other by a combination of directed and undirected links, and work together to provide the required functionality of the VNF. It is worth noting that it is possible to change the internal topology of VNFCs (as part of the continuous innovation process, for example, to improve performance) without necessarily changing the interfaces external to the overall VNF.

B. VNF Example

An example of a practical VNF which is composed from VNFCs is Clearwater's cloud IMS [8], an open source IMS core, developed by Metaswitch Networks. As shown in 2, it is composed of five core nodes named Bono, Sprout, Homestead, Homer, and Ralf. Bono is a Session Initiation Protocol (SIP) edge proxy which provides a Web Real-Time Communications (WebRTC) interface to UEs. It is the anchor point for UEs to the Clearwater system. Sprout is a SIP registrar and authoritative routing proxy which handles UE authentication. It includes a memcached cluster storing client registration data. Homestead provides a Web services interface to sprout for retrieving authentication credentials and user profile information. It runs as a cluster using Cassandra as the store for mastered/cached data. Homer is a standard XML Document Management Server (XDMS) used to store multimedia telephony (MMTEL) service settings documents for each user of the system using Cassandra as the data store. Ralf provides Rf Charging Trigger Function (CTF), which is used in IMS to provide offline billing. Bono and Sprout report P-CSCF and I/S-CSCF chargeable events respectively to Ralf, which then reports these over Rf to an external Charging Data Function (CDF). It uses a memcached to store and manage session state. As shown in the figure, communication is initiated by the UE attaching to Bono. Bono then requests for UE authentication from Sprout, which does this by querying the database in Homestead. During a call, Sprout uses user media settings stored in Homer, and the calls are charged by Ralf. From this flow of communications (for just one VNF), it is clear that if Bono is over-loaded caused by too many UEs trying to attach to the system, then Sprout would likely get more requests as well, and so would Homestead. Moreover, if each of these UE attachments is successful, both Homer and Ralf would also get overloaded. It is these interactions between VNFCs that motivate our approach. While this illustration considers mainly control plane signalling, a similar effect is expected by considering the data plane, for example, in a SFC where user packets have to be processed by one VNF before they move to the next.

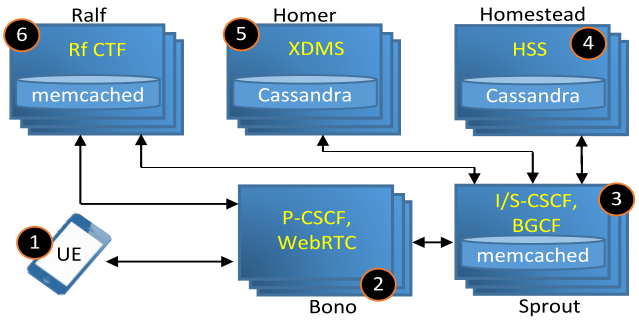

Fig. 2. VNF 2: Clearwater cloud IP multimedia subsystem.

Throughout this article, Fig. 1, and in particular VNF 2 and its internal structure (which also represents the topology of the Clearwater VNF in Fig. 2) will be used as a running example to illustrate various aspects of our proposal. However, we use such specific and simple illustrations only to enhance clarity for the reader. Our proposal can be applied with ease to any SFC whose topology can be represented in the form shown in Fig. 1.

C. Motivation

In order to have the SFC shown in Fig. 1, a number of problems should be solved. First, physical infrastructure must be deployed. Then, there must be algorithms to optimise the placement of virtual containers (or VNFs) onto the available physical servers. Finally, throughout the lifetime of the SFC, it is necessary to determine the actual amount of resources allocated to each virtualisation container and/or how many virtualisation containers are used for each VNFC. These three problems are refered to as server placement, function placement, and dynamic resource allocation respectively [4]. Server placement and function placement have already attracted a lot of attention, for example in [9]–[13] respectively, and are out of scope for this article.

In this article, we focus on dynamic resource allocation. We consider that the VNFs (and hence VNFCs) have already been placed/mapped in the respective virtual resources on which they run. This work is motivated by the fact that the resource requirements of each VNF change over time with changes in traffic, which calls for ways of increasing and reducing resources allocated to the VNFCs as needed. Even more, since there is a non-negligible delay in spinning-up new resources (such as VMs), waiting until the system is over-loaded so as to scale resource up could negatively impact user QoS.

As an example, in Fig. 3, we show results from a recent study that considers the boot up times (and their scalability) of three of the most commonly used VIMs OpenStack, Eucalyptus, and OpenNebula. As can be noted, even for a

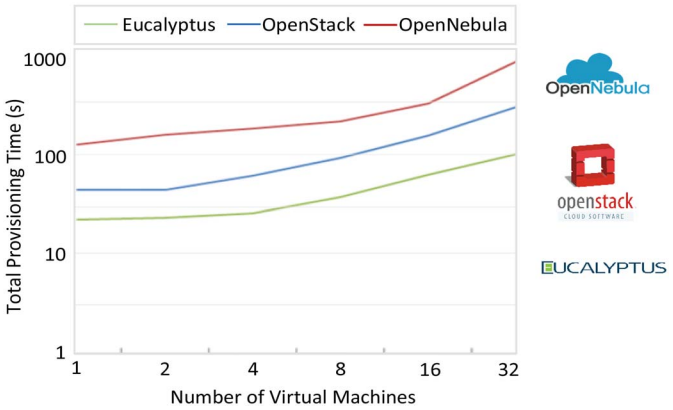

Fig. 3. VM provisioning time for 3 of the most common virtual infrastructure managers (VIMs). This is the total time taken to provision VMs in the most mature, widely deployed VIMs. This figure has been adapted from a recent study published in [15].

single VM, the time needed for boot and basic provisioning is in the order of tens of seconds. This time grows very quickly if a higher number of VMs is required. While Linux containers [14] can boot up faster than the times shown in the figure, they are not yet widely used, in part due to some of open issues (such as isolation) with them. Therefore, we believe that the VIMs shown in the figure will remain a very important part of NFV for the foreseeable future. Considering that NFV will be an essential building block of 5G systems, and given that most of the applications (such as automated driving, smart grid, etc.) which are expected to be supported by 5G require real-time response in case there is a need for more resources, it is clear that state-of-the-art solutions fall short. The objective of this article is to fill this gap. In particular, our aim is to take advantage of the topology of the VNFs so as to predict the resource requirements of each VNFC, and to use this prediction to scale in/out in time so as to have the resources available when they are needed. The next two sections introduce the concept of GNNs and how it has been used to develop a system that forecasts the resource requirements of each VNFC, in order to obtain advance information of the VNFC's upcoming resource needs, allowing an orchestration entity to satisfy such needs just in time.

III. Graph Neural Networks

GNNs [5] are a supervised learning model aimed at solving problems in the graphical domain. The main idea of GNNs is to define each node n in the graph based on its *features*, f_n, and to complement this by the information (features) observed in the *neighbourhood*, $n^\star$, of the node. While there may be different definitions of neighbourhood, what is used in this article is a set of nodes directly connected to node n. Using these two information sources, the GNN model determines a *state* s_n for each node n, which is then used to determine an *output* o_n for the same node. The determination of the state and output for each node is governed by equations (1) and (2) respectively.

$$s_n = \sum_{m \in n^\star} h_w(f_n, f_m, s_m), \quad \forall n \tag{1}$$

$$o_n = g_w(s_n, f_n), \quad \forall n \tag{2}$$

TABLE II
Summary of Key Notations

Symbols	Description
n	Node or Virtual Network Function Component
$m \in n^*$	Neighbour of n, i.e. node directly connected to n
l_m	Link connecting nodes m and n
s_n	State of node n
f_n	Features of node n
f_{mn}	Features of link between m and n
h_w	Transition parametric function
g_w	Output parametric function
m_n	Memory of node n
c_n	CPU of node n
d_n	Processing delay for node n
d_{mn}	Propagation delay of link mn
b_{mn}	Bandwidth of link mn
π	Number of past measurements used for predictions
τ	Number of time steps in future for which prediction is performed
i	Iteration for state computation
o_n	Predicted output/resource utilisation for node n
ξ_n	Actual resource utilisation of node n
t	Time instance

where f_m and s_m are the features and state of neighbour $m \in n^*$ respectively. It is possible to also include the features f_{mn} of the direct link between n and m in equation (1) only resulting in a problem with more dimensions. h_w and g_w are parametric functions which express, respectively, the dependence of the state at each node on the state of its neighbourhood, and the dependence of the node output on its state, respectively. h_w is known as the *transition function* while g_w the *output function*. Equations (1) and (2) represent the activity of a network consisting of units which compute h_w and g_w for each node. This is the main idea of GNNs, an information diffusion mechanism, in which a graph is processed by a set of units (h_w and g_w), each one corresponding to a node of the graph, which are linked according to the graph connectivity. These units update their states and exchange information until they reach a stable equilibrium. Interested readers are referred to [5] for more details about the model. It should suffice to say here that by directing the diffusion process, the model is expected to converge exponentially fast, and be stable while determining the node states, and hence the GNN output. In Table II, we summarise all the key notations used throughout the article.

IV. GNN-Based Dynamic Resource Management

Since neighbouring VNFCs will usually be part of the same SFC, resource fluctuations at one VNFC are expected to influence resource requirements at its neighbours as traffic flows from one VNFC to the other. This dependency of VNFCs on their neighbourhood makes the connectionist approach derived from the GNN model an interesting fit as an approach for managing resources in NFV. Therefore, the GNN-based dynamic resource management system proposed in this article is derived from equations (1) and (2), and is shown in Fig. 4 for a single VNFC. As can be seen, the system is comprised of four main

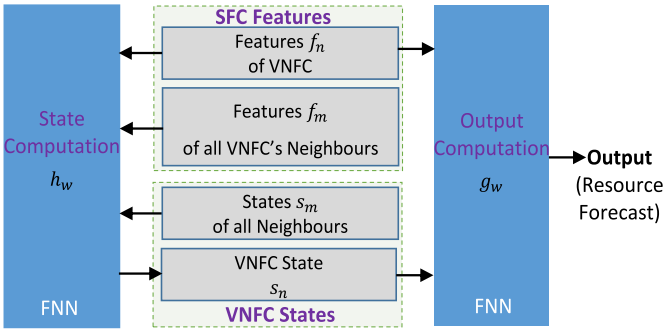

Fig. 4. GNN-based resource forecasting model for a single VNFC.

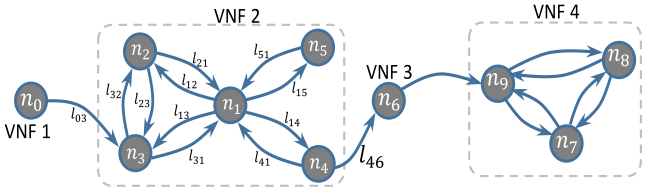

Fig. 5. SFC modelling: VNFC directed graph.

components: (1) SFC features, (2) VNFC states, (3) state computation, and (4) output computation. In what follows, these components are described.[1]

A. SFC Features

The SFC features are the observations or monitoring data from the VNFCs, and consitute the input to both h_w and g_w. In an NFV environment, these features represent the network parameters (such as CPU or RAM utilisation levels) that can be measured. As proposed by ETSI [16], the SFC in Fig. 1 may be represented as a VNF-FG. In this article, we consider the resulting VNF-FG at the granularity of VNFCs, i.e., the nodes represented in the VNF-FG are VNFCs rather than VNFs.

Specifically, we model a SFC as a directed graph $G(N, L)$, where N represents the set of VNFCs and L the set of links between these VNFCs. An example of such a representation is given in Fig. 5 which is based on the SFC in Fig. 1. As can be seen in the figure, subsets of VNFCs make up a VNF (e.g., n_1, n_2, n_3, n_4 and n_5 make up VNF 2 from Fig. 1). Each VNFC $n \in N$ has a set of features $f_n \in \mathbb{R}^{D_N}$ which represent a measurable resource for the VNFC, such as VNFC memory m_n, CPU c_n, processing delay d_n, etc. In the same way, each link $l_{nm} \in L$ which connects VNFC n to m is characterised by a set $f_{nm} \in \mathbb{R}^{D_L}$ of features, which could represent link delay d_{nm}, bandwidth b_{nm}, etc. D_N and D_L refer to the dimensions of the feature sets for VNFCs and links respectively. Equations (3) and (4) show example feature sets for VNFC n and link l_{nm} respectively, for which $D_N = 3$ and $D_L = 2$.

$$f_n = \begin{bmatrix} c_n \\ m_n \\ d_n \end{bmatrix} \tag{3}$$

[1]It is worth noting that Fig. 4 only shows the model for a single VNFC. Such a system would have to be duplicated for each VNFC in a given SFC, with the resulting topology being based on that of the Virtual Network Function Forwarding Graph (VNF-FG).

$$f_{nm} = \begin{bmatrix} b_{nm} \\ d_{nm} \end{bmatrix} \tag{4}$$

The objective is to monitor the features of each VNFC over time, and to use such historical observations, as well as the historical observations from the VNFC's neighbours to predict its subsequent features, which – in this case – represent future VNFC resource requirements. In order to define both historic and future resource utilisation, we refer to the VNFC and connected link features at (discrete) time step t by $f_n(t)$ and $f_{nm}(t)$ respectively. At any time t, we should be able to predict future resource utilisation using a finite horizon of past resource utilisation measurements. We denote the number of past measurements included in such a horizon as π. An example of current ($c_n(t)$, $m_n(t)$ and $d_n(t)$) and π previous measurements is shown by the vectors in equations (5) and (6). Using the observations represented by equations (5) and (6), the objective is to predict – say – the CPU requirement $c_n(t+\tau)$ of VNFC n at a time τ time steps after t. In the rest of this article, wherever f_n or f_{nm} is used, it should be interpreted to mean the set containing $f_n(t)$ or $f_{nm}(t)$ plus the full history of features over the period π.

It is important to note that modelling of links and their features is only included here for completeness of the model, as the link features will not be used as neighbourhood information for VNFCs. The reason is that we consider that the resource utilisation profile of a directed link is directly dependent on that of the VNFC at its source from which the traffic originates, and hence, the information obtained from a VNFC would be similar to that obtained from the link.

$$f_n(t) = \begin{bmatrix} c_n(t) \\ m_n(t) \\ d_n(t) \\ \cdot \\ \cdot \\ \cdot \\ c_n(t-\pi) \\ m_n(t-\pi) \\ d_n(t-\pi) \end{bmatrix} \tag{5}$$

$$f_{nm}(t) = \begin{bmatrix} b_{nm}(t) \\ d_{nm}(t) \\ b_{nm}(t-1) \\ d_{nm}(t-1) \\ \cdot \\ \cdot \\ \cdot \\ b_{nm}(t-\pi) \\ d_{nm}(t-\pi) \end{bmatrix}. \tag{6}$$

B. VNFC States

In line with the VNF design patterns proposed by ETSI [7], we consider that VNFCs can be *stateful*, with each VNFC $n \in N$ having a state $s_n \in \mathbb{R}^{S_D}$ of dimension S_D. The state s_n is derived from combining the features of a given VNFC with those from other VNFCs in its neighbourhood using the function h_w. This implies that the state of a given VNFC is dependent on the topology or connectivity of the VNF-FG.

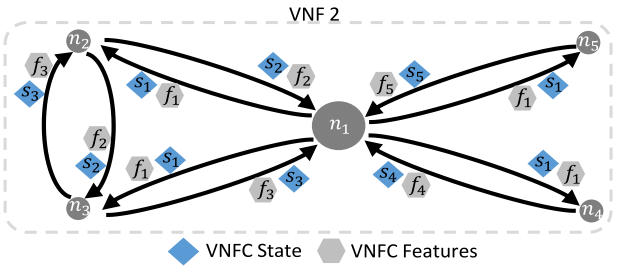

Fig. 6. States and features from VNFC neighbourhood. Each VNFC receives as input the state and features from all VNFCs that have a directed edge towards it. It is worth noting that the neighbourhood effects of VNFs 1 and 3 on 2 have been omitted from this figure and all the proceeding analyses only for brevity, and keeping the representations simple. However, the proposed solution takes into account the dependencies of all the VNFCs in the SFC.

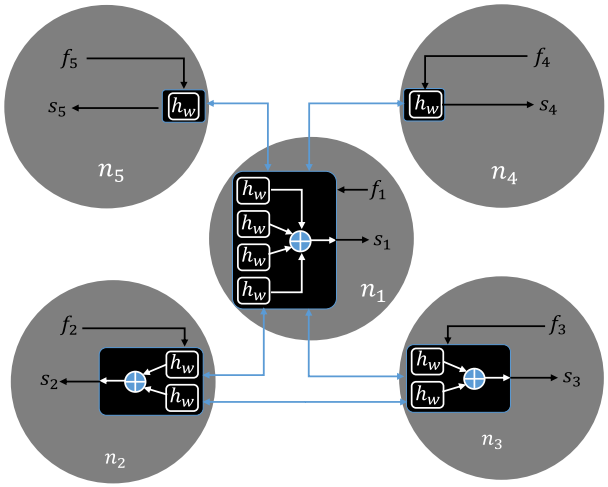

Fig. 7. GNN-based model for VNF 2: Each VNFC is replaced by a $h(w)$ function. The functions are connected to each other following the topology of the original VNF. For example, VNFC n_1 is connected to all other VNFCs while n_3 is only connected to n_1 and n_2. Observe that the number of h_w functions representing each VNFC is equal to the number of neighbours it has, such that the effect (state and features) of each neighbour is processed by the corresponding h_w function.

Such topology-awareness is represented in Fig. 6 which shows the dependencies of VNFCs in VNF 2 on each other. The Fig. depicts that, for example, the state s_1 of VNFC n_1 is dependent on the states s_2, s_3, s_4, and s_5 of all directly connected VNFCs, as well as the corresponding features f_2,f_3,f_4, and f_5. The state s_n is determined using equation (1). This means that, considering Fig. 6, the state s_3 of VNFC n_3 is given by (7).

$$s_3 = h_w(f_3, f_2, s_2) + h_w(f_3, f_1, s_1). \tag{7}$$

C. State Computation

State computation involves using equation (1) to determine the state for each VNFC. However, as can be observed from the equation, for any given pair of directly connected VNFCs, the state of each of them depends on that of the other. Therefore, the main task of state computation is to find a method to solve equation (1). The existence and uniqueness of a solution to (1) is guaranteed by Banach's fixed-point theorem [5], [17]. However, this requires that the global function h_w is a contraction map with respect to s, i.e., equation (8)

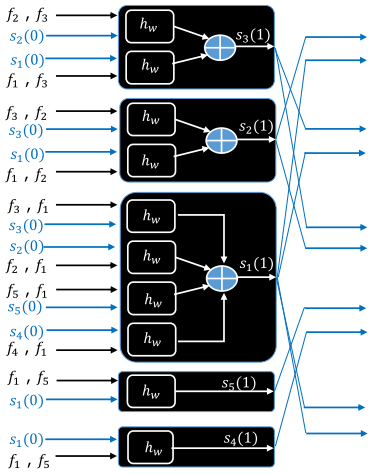

Fig. 8. First iteration. Fig. 7 has been transformed into a single vertical representation representing the first iteration.

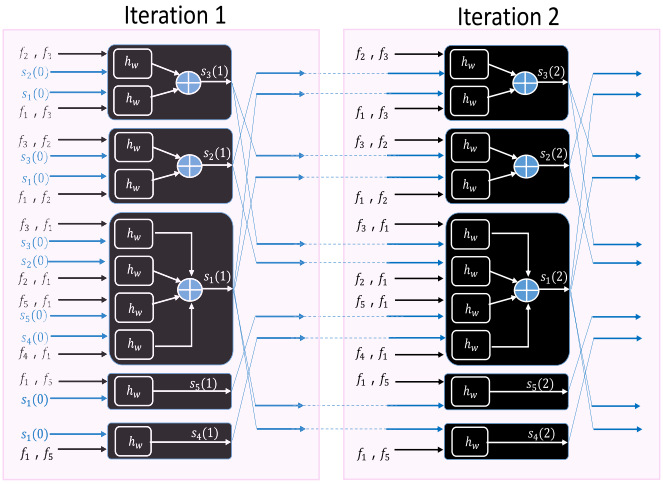

Fig. 9. First and second iterations. Fig. 8 has been extended to include the second iteration, with the VNFC states from the first iteration being input into the second iteration as the states of the VNFCs.

must hold for some constant $0 \leq \rho < 1$ and any two state vectors $s_a, s_b \in \mathbb{R}^{S_D}$, where $||.||$ represents a vector norm.

$$\|h_w(s_a) - h_w(s_b)\| \leq \rho \|s_a - s_b\| \tag{8}$$

When equation (8) is satisfied, state computation is achieved using a classic iterative scheme given in equation (9) where $s(i)$ is the i^{th} iteration of the computation. This way, the function h_w stores the current state $s(i)$, and when called, calculates the next state $s(i+1)$. It can be observed that this makes the current state $s_n(i)$ of a VNFC n dependent on the previous state $s_m(i-1)$ of its neighbour m.

$$s_n(i+1) = \sum_{m \in n^\star} h_w(f_n, f_m, s_m(i)), \ \ \forall n \tag{9}$$

To illustrate the iterative state computation process, consider Fig. 8 which is the same as Fig. 7, except that it has been transformed into a single iteration of the state computation process. This is the first layer of the GNN. It can be observed that the input to this layer is a set of initial states for

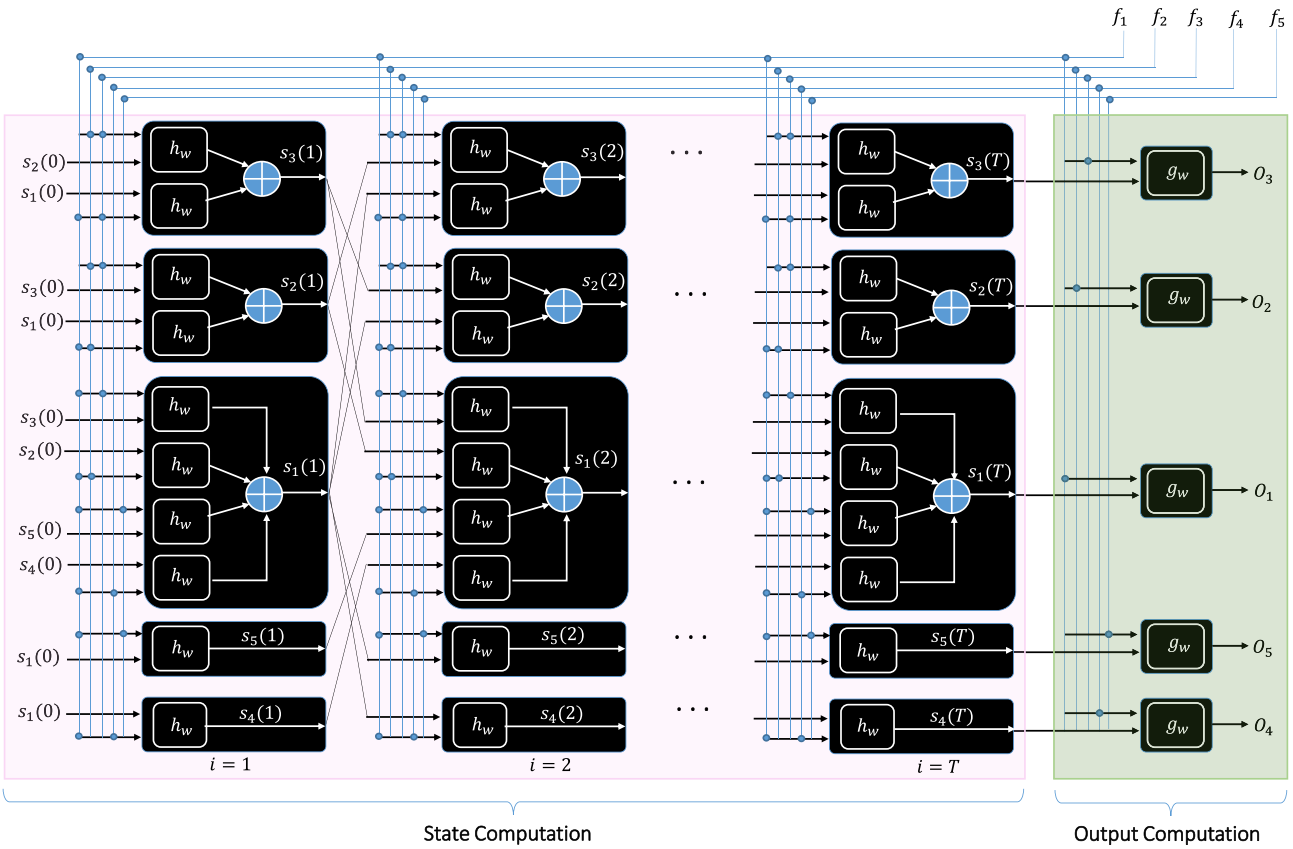

Fig. 10. Final GNN model including the encoding network to iteratively determine the states of VNFCs, and the g_w functions to determine the VNFC resource prediction.

each VNFC (which maybe initialised to zero), as well as the observed features. These are processed by the respective (for each neighbour) h_w functions in each VNFC and their outputs summed (in accordance with equation 9) to give the output of the first layer. As can be observed in Fig. 9, these outputs are fed into the second layer in the second iteration. The states are fed into the proceeding layer following the topology of the original VNF. For example, the first output state $s_3(1)$ from VNFC n_3 is only fed to its neighbouring VNFCs – n_1 and n_2. The iterative process is continued for an appropriate number, T of iterations to obtain what is known as the encoding network shown in Fig. 10. Each layer i in the encoding network corresponds to an iteration in which the state $s(i+1)$ is computed for each VNFC. The h_w units of any two consecutive layers are connected following VNF-FG connectivity.

It is worth remarking that the iterative process described above converges exponentially fast to the solution of equation (9), i.e., convergence to the fixed point [5]. The solution is equal to the convergence point of equation (9) for any initial value $s(0)$ of the VNFC states. In this article, a FNN is used as h_w. This way, we can ensure that h_w is a contraction map by limiting its parameters, i.e., the range of values that the weights w of the FNN can take on [18]. As will be discussed in the next section, this is achieved by using an *error function* designed with this requirement in mind.

Algorithm 1 GNN-Based Model for NFV $G(N, E)$

Initialise: w, iteration $i = 0$, state $s(i) = 0\ \forall n \in N$

procedure OBSERVATION
    Observe f for all VNFC's and their neighbourhoods
end procedure

procedure STATE COMPUTATION
    while $(i < T)$ **do**
        Compute $s(i+1)$ using equation (9)
        $i = i + 1$
    end while
end procedure

procedure OUTPUT COMPUTATION
    Compute $o(i)$ using equation (10)
end procedure

D. Output Computation

Output computation involves taking as input the states calculated by the h_w functions in the iterative process described in the previous subsection, and combining it with the feature set of the VNFC to forecast a future resource requirement. This is shown in Fig. 10. The final output (forecast resource requirement) of a given VNFC is produced by another unit,

which implements g_w for all VNFCs using equation (10).

$$o_n(i) = g_w(s_n(i), f_n), \ \forall n \tag{10}$$

The function g_w can be any general parametric function as long as it can be trained in a supervised manner, and the gradient of its output with respect to its input can be calculated. The original model proposed by [5], which is also adopted in this article, uses a FNN for g_w.

E. Summary

To summarise, the proposed model is defined by equations (9) and (10) and represented in Fig. 10 for a specific example of VNF 2. It takes as input the resource utilisation observations (features) of a SFC and outputs, for each VNFC, a forecast for the specified resource requirement. It can be observed that the model involves computing the state of each VNFC. This is achieved through an iterative process involving as many h_w functions for each VNFC as its neighbours in the SFC. The outputs of the state computation are used for the output computation state to determine the resource requirement prediction. As already discussed, each h_w or g_w is implemented by a FNN. This process is summarized in algorithm 1. As can be seen, the process consists of three main steps: (1) observing the resource utilisation of the VNFC as well as that in its neighbourhood, (2) using the observed resource utilisation to determine the state of the VNFC, and (3) using results from the first two stages to determine the forecasted resource utilisation.

V. Learning and Adaptation

In order to achieve forecasts that correctly approximate actual resource requirements, the two functions h_w and g_w must be trained. This involves using data that has both inputs f and target outputs ξ, to adapt the weights w of the FNNs to the task under consideration. In the case of the problem addressed in this article, we need to have sample data, that shows for a given resource utilisation profile (i.e., historic and current features $f_n(t-\pi), \ldots, f_n(t)$), the resource utilisation $o_n(t+\tau)$ at a given time in the future. This learning task can be posed as the minimisation of a penalised quadratic cost function (11).

$$e_w = \sum_{n \in N} \left(\frac{1}{2}(o_n - \xi_n)^2 + \beta L(o_n) \right) \tag{11}$$

The first term in equation (11) is the standard error term usually used for training FNNs [19]. The second term is a penalty function which is added to the error function to ensure that the function h_w is a contraction map. The relative importance of the second term can be adjusted using the constant β. The second term, which has been adapted from the one used in [18], is meant to limit the values that can be assumed by the weights w to low values. This is achieved by using the function L (defined below) to penalise the FNN whenever its output is above a given threshold μ, known as the contraction constant. In this article, since all inputs to the system are first scaled to the range (0, 1), the constants μ and β are both set to 1.

$$L(y) = \begin{cases} (y-\mu)^2 & \text{if } y > \mu \\ 0 & 0 \end{cases}$$

Algorithm 2 Learning and Adaptation

1: **procedure** Learning and Adaptation
2: Initialise: w, $k = 0$
3: **while** (stopping criterion not satisfied) **do**
4: Compute state s and output o using algorithm 1
5: $\frac{\partial e_w}{\partial w} \longleftarrow$ Back Propagation Through Time
6: Update w using equation (12)
7: $k = k + 1$
8: **end while**
9: **end procedure**

The learning objective is to find the weights w for each h_w and g_w such that the cost function (11) is minimised. The learning algorithm used in this article is based on gradient-descent, and involves four main steps:

1) STEP 1: At iteration $k = 0$, the weights w of h_w and g_w respectively are initialized randomly between -0.5 and $+0.5$.
2) STEP 2: At each iteration $k = k + 1$, state and output computation is done using equations (9), and (10) respectively,
3) STEP 3: The gradient $\frac{\partial e_w}{\partial w}$ of cost function (11) with respect to the parameters w for all h_w and g_w is computed,
4) STEP 4: The weights w for all h_w and g_w are updated using equation (12), where α is the learning rate.

$$w(k+1) = w(k) - \alpha \frac{\partial e_w}{\partial w} \tag{12}$$

Steps 1 and 4 are obvious, while step 2 has been discussed in Sections IV-C and IV-D. Step 3 is realised by using backpropagation-through-time (BPTT) [5], [20]. BPTT involves carrying out the traditional back propagation [20] on the encoding network (Fig. 10) to compute the gradient of the cost function for each h_w and g_w and summing all the gradients up. The learning and weight adaptation algorithm is summarised in Algorithm 2. In the algorithm, the learning steps 1−4 above are carried out on lines 2, 4, 5, and 6 respectively.

VI. Evaluations

A. Experimental Setup

The proposed system has been evaluated using the setup shown in Fig. 11. The deployment is comprised of 6 main components: The Clearwater cloud IMS described in Section II, OpenStack, User Equipments (UEs), Monitoring, Domain Name System (DNS), and the algorithms being tested. In our implementation, UEs are realised using SIPp [21]. SIPp is an open source test tool/traffic generator for the SIP

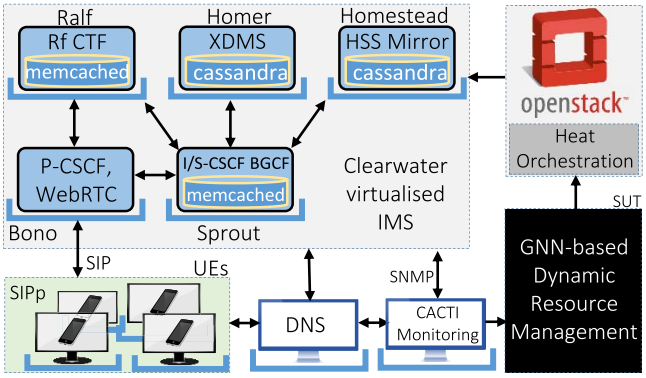

Fig. 11. NFV implementation used for evaluations.

protocol. Two SIPp instances were created each running in a VM. Each SIPp instance has 50,000 unique registered users. Calls originate from users on one SIP instance to users on the other. In order to monitor the resource utilisation of the VNFCs in the system, we used Cacti [22] an open-source, Web-based network monitoring and graphing tool which polls all system nodes using Simple Network Management Protocol (SNMP). Finally, we use BIND [23] an open source implementation of DNS to allow Clearwater nodes identify each other, and for load distribution when any of the nodes has more than one instance.

In the experiments, each of the Clearwater nodes represents a VNFC,[2] and is hosted in a VM running in OpenStack. Therefore, the basic evaluation system deployment included 10 VMs running in OpenStack (5 for Clearwater nodes, 2 for UEs, 1 for DNS, 1 for Cacti monitoring, and 1 hosting the system under test (the proposed algorithms). These VMs, and additional ones (for horizontal scaling of Clearwater) were automatically deployed in OpenStack using Heat Orchestration Templates (HOTs) [24].

B. Setup Parameters

Each VM used in the tests has 1vCPU, 2GB RAM and 8 GB Storage, each running Ubuntu 14.04. Calls were generated from one UE to the other following a Poisson distribution with an average arrival rate of 10 calls per second, and each call lasting an average of 180 seconds following a negative exponential distribution. To model a time of day effect on traffic arrivals, the above arrival pattern is repeated after every 50,000 calls, with the arrival rate and call duration parameters being halved and doubled alternately. During the duration of each call, real voice and video media are transmitted between the UEs. The voice/video content is derived from VoIP (Skype) traces [25] which contain network traffic captured on the main link of Politecnico di Torino involving Skype traffic from students, researchers, professors and administration staff. The original 3.75 GB of end-to-end voice only and voice+video calls traces with about 40 million packets was split into 40 .pcap files, each with about 1 million packets. For each established call, one of these media files (chosen at random) was played to simulate real voice or video media.

[2] It is important to note that while our experimental setup involves multiple VNFCs that make up a single VNF, it does not limit our proposal to a single VNF. This is because irrespective of the number of VNFs or constituent VNFCs, for as long as a topology of the functions in a SFC can be created, then the GNN-based model can be used. This also includes situations where the VNF may be a blackbox, in which case it would considered as having a single VNFC.

C. Experiments

Three sets of experiments were done. In each experiment, measurements of resource parameters (CPU, RAM, latency, Call drops) for all Clearwater nodes were taken every 15s. The first experiment was used to collect 10,000 data points which were used to train the FNNs. The history and forecasting periods used were $\pi = \tau = 20$, implying that for each VNFC, the last 20 observations were used to predict the resource requirement 20 time units in the future. In the second experiment, The trained system was tested to determine its prediction accuracy over 1,000 measurements, in which case, every 15s, the system was run to determine an output, and its output compared to the actual resource requirements 20 time units later. This was used as a prediction accuracy test without performing any resource allocations. Finally, in the third set of experiments, the system predictions were used to actually effect resource allocations in the Clearwater system. In this case, the system was programmed to effect a deployment of a new VNFC whenever it predicted that the % CPU utilisation of a given VNFC would exceed 40%, and where possible (if more than one are available), to reduce the number of deployed VNFC's when the predicted utilisation is 20%. The motivation behind using 40% and 20% respectively as the thresholds is VNF specific. In our monitoring of the normal operation of the Clearwater VNF, we observed that the VMs had a relatively low CPU utilisation most of the time, but that beyond 40% of CPU utilisation, performance (call drops) would degrade, while below 20% of CPU utilisation, the call drop rate remained almost unchanged. It is worth noting that these thresholds may be different for a different VNF.

D. Comparisons

The proposed system was compared with two alternatives: a **static** approach in which the resources were not changed at all, and a **manual** approach where VNFC deployments were programmed to be performed when the system crossed the (40%) resource utilisation threshold. The main difference between the manual programming and the proposal given in this article is that in the manual approach, the process of scaling resources is only started after a given threshold is reached, while in our proposal, the reaching of this threshold is predicted ahead of time, and the scaling process started before the threshold is actually reached. In addition, the prediction accuracy of the proposed system was compared with that resulting from a simple FNN. In order to compare the differences in prediction accuracies between the FNN and the GNN, the mean absolute percentage error (MAPE) defined using equation (13) was used.

$$MAPE = \left(\frac{1}{k} \sum_{t=1}^{k} \left| \frac{\xi_n(t) - o_n(t)}{\xi_n(t)} \right| \times 100 \right). \qquad (13)$$

TABLE III
NEURAL NETWORKS' ARCHITECTURE PARAMETERS

Neural Network	FNN	GNN	
	FNN for each VNFC	h_w	g_w
Number of Layers	3	3	3
Number of Neurons in Output Layer	3	3	3
Number of Neurons in Input Layer	63	129	66
Number of Neurons in Hidden Layer	33	66	35

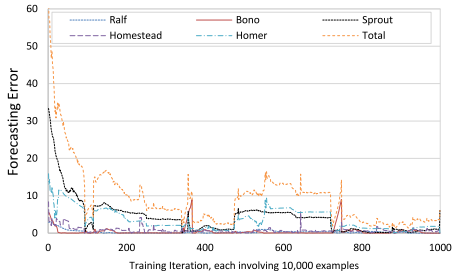

Fig. 12. Total forecasting error.

E. Neural Network Architectures

The evaluations include three neural network architectures. The first is the basic FNN whose performance is compared against that of the GNN. The other two are also FNNs, but with their architectures designed in line with the requirements of the functions h_w and g_w. The h_w, g_w, and FNN neural networks were implemented in architectures with (i) one hidden layer, and (ii) a number of neurons in the hidden layer equal to the average of the neurons in the input and output layers. Such an architecture has been shown to produce universal approximators [26], [27]. With these guiding rules, the resulting neural network parameters are given in Table III. It can be observed that each architecture has 3 layers. Since we consider 3 VNFC features (i.e., CPU, memory and processing delay - see Section IV-A for details), the dimension of the state is 3 (one for each feature). This is the reason why the output layer for each of the architectures is 3. The number of neurons in the input layer is determined based on the inputs that should be accepted. For example, for the FNN, since we have 3 SFC features and since we consider both the current value of each of the parameters plus the 20 previous values, the total number of inputs is $3 \times (20+1) = 63$. For h_w, the inputs contain two sets of the features (one from a neighbouring VNFC) as well as the state from the neighbouring VNFC. This gives $(3+3) \times (20+1) + 3 = 129$. For g_w, the inputs contain one set of features (from the VNFC under consideration) as well as the state of the VNFC. This gives $(3) \times (20+1) + 3 = 66$. Finally, the number of neurons in the hidden layers for each architecture is determined as the average of the number in the input and output layers.

F. Results

The evaluation results are shown in Figs. 12–26. Fig. 12 shows results from the first set of experiments (training), while

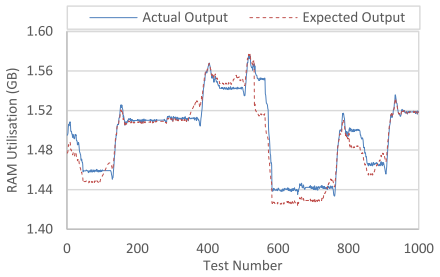

Fig. 13. Ralf RAM utilisation.

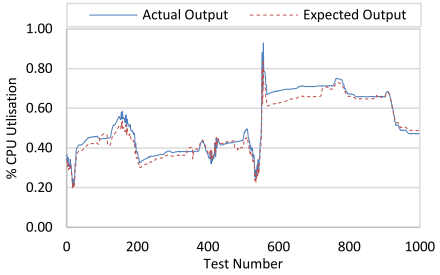

Fig. 14. Homer CPU utilisation.

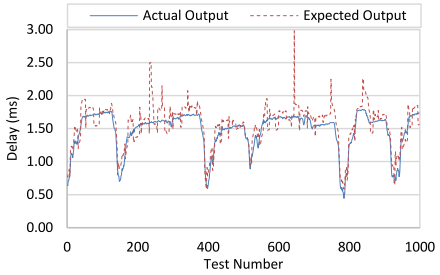

Fig. 15. Homestead processing delay.

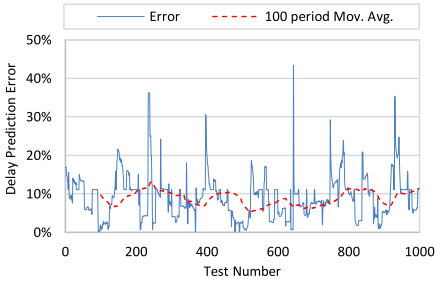

Fig. 16. Percentage error on delay prediction.

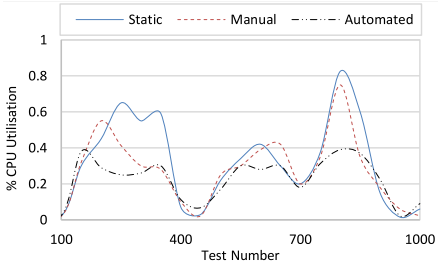

Fig. 17. Percentage CPU for homer.

Figs. 13–16 evaluate the prediction accuracy of the trained system. Figs. 17–20 are based on 100 period moving averages. Results from evaluating the effect of the resulting system are

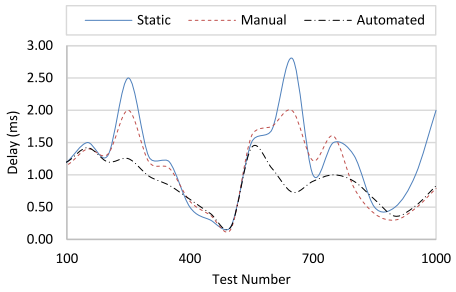

Fig. 18. Effect on processing latency.

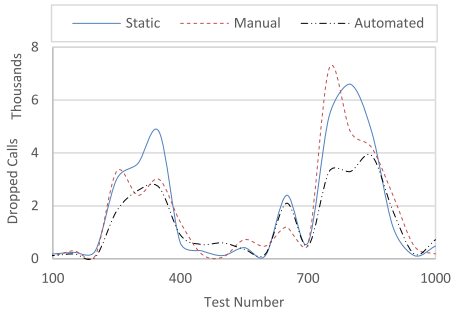

Fig. 19. Effect on calls dropped.

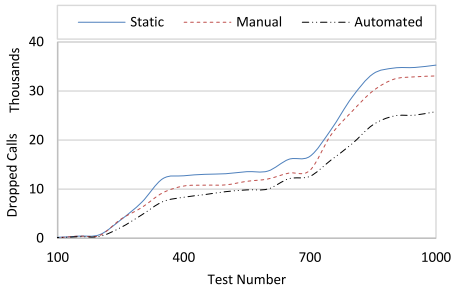

Fig. 20. Cumulative call drops.

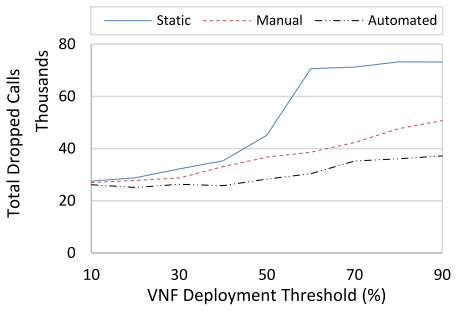

Fig. 21. Effect of deployment threshold.

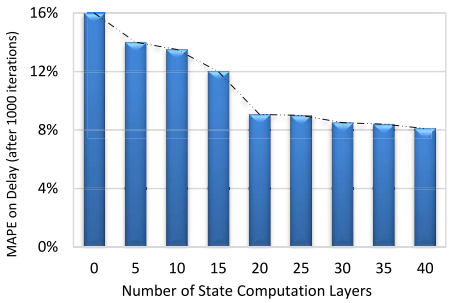

Fig. 22. Effect of number of state computation layers.

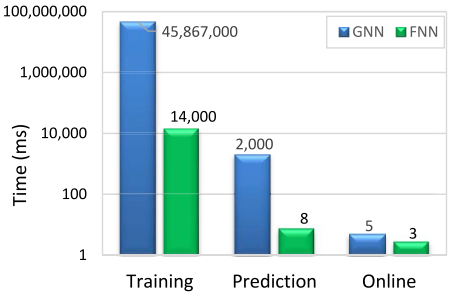

Fig. 23. Training and prediction times.

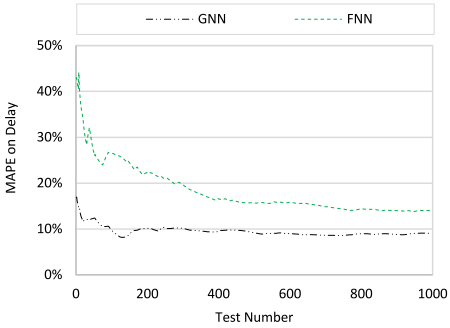

Fig. 24. Comparison with other approaches.

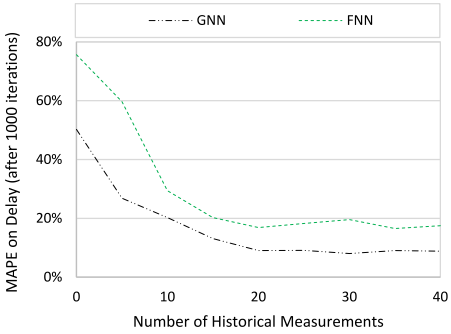

Fig. 25. Effect of number of past measurements.

shown in Figs. 17–20. Figs. 21–26 present a comparison of the proposed approach with a traditional neural network, as well as the effect of changing the different experimentation parameters.

From Fig. 12, it can be observed that the total prediction error (computed using equation (11)) is initially high, and falls almost exponentially until it becomes stable after about 700 iterations (each with 10,000 training examples) of the learning and adaptation algorithm.[3] With a final error of about 5 as shown in Fig. 12, and considering this is the total error for 10,000 examples and 5 VNFCs, it can be concluded that the

[3]It should be mentioned here that the stopping condition in algorithm II is 1000 iterations.

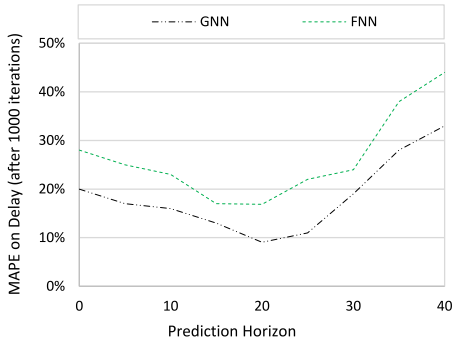

Fig. 26. Forecasting different horizons.

system achieves an approximate accuracy of about 99% percent on the training data set. However, this level of accuracy is not realised when the system is tested on a new data set as shown in Figs. 13–15, which show the prediction accuracy of the RAM utilisation for ralf, CPU utilisation for homer, and request processing latency for homestead respectively. First, it can be noted from Fig. 16 that the accuracy on latency prediction is about 90%. This loss in accuracy can be explained by the error term used during the weight learning phase which attempts to prevent the weights from assuming large values. By limiting the value of the weights, the FNNs may be prevented from generalisation with high accuracy, i.e., the capability to use acquired knowledge on new tasks.

In Fig. 17, we show the evolution of % CPU utilisation for homer for the three scenarios described above. It can be observed that the proposed automated approach correctly forecasts the trend in resource utilisation, and deploys an additional homer VNFC, leading to a reduction in the load of the current VNFC as the load is now shared. When the utilisation reaches 40%, the manual scenario also triggers the deployment of an additional homer VNFC, which takes some time to start taking up load, but when it eventually does, the resource utilisation of the original VNFC reduces compared to the static scenario in which the number of VNFCs is not altered. Similar profiles can be seen in the other two cases when the utilisation crosses the 40% mark. It is also worth noting that there is not a very big difference in the performance of the manual and automated scenarios when the resources have to be scaled down (when utilisation is below 20%). This can be explained by the fact that since it does not require any preparation to shutdown resources (VMs in this case), predicting the need to shutdown in the use case under consideration does not give any advantage since in any case both scenarios have to wait until a certain point is reached before scaling down. However, both approaches would still perform better than the static approach in which resources would be left allocated, even when unutilised.

The performance results discussed above can still be observed in Figs. 18–20 in which the automated approach outperforms the other two approaches. In fact, it can be seen that the total number of calls dropped due to the system being overloaded over the entire testing period is 29% lower for the proposed approach compared to the manual one. Moreover, it is important to state that our prediction is mainly based on system load, and does not take into consideration the effect of traffic arrivals. It is possible that by attempting to predict traffic arrivals, and incorporating this into the model may yield better results. However, since we used synthetic traffic arrivals (because we could not get more practical data), trying to predict this could have been trivial. This could be an interesting future consideration.

In Fig. 21, we evaluate the effect of the threshold used to effect resource deployments. It can be noted that the number of dropped calls for all the three approaches is initially low and comparable but increases as the deployment threshold is increased. This can be explained by the fact that when the threshold is low, all the approaches deploy new VNFs even before they are actually required, making all approaches have comparable performance. Such a configuration would however result into an inefficient utilisation of resources. However, this changes as the threshold is increased since at higher values, all VNFs are in urgent need for resources that a prediction or just-in-time deployment produces significant improvements. Fig. 22 shows the effect of varying the number of layers used in the state computation process. We observe that as the number of layers is increased, the mean average percentage error first reduces and then becomes almost constant. This implies that beyond a certain value, the increased layers become redundant without contributing to the prediction accuracy of the system.

Figs. 23–26 compare the performance of GNN with FNN. In Fig. 23, we show the time taken to train the neural networks and to make predictions. It can be observed that in each case, GNNs take significantly more time than FNNs. This is expected, since each GNN is made up of multiple FNNs. It should be noted that as shown in the figure, in the GNN experimental setup, each iteration takes about 45s to complete, giving a total training period (for 1,000 iterations) of about 45,000s. However, since the learning/training phase is an offline process, this does not affect the online performance of the system. After the training period, the weights of all the FNNs in the model are saved in a file, from where they can be loaded every time a prediction is needed. Therefore, we observed that each online prediction required about 2s, including the time required to read the weights from a file. Moreover, in systems that are time critical, the prediction system could be kept running (online results in Fig. 21), in which case the time needed to load the weights from file can be saved. This way, our evaluations showed that a prediction can be obtained in about 5ms. These prediction times are comparatively low, given that predictions are performed for resource requirements 300s ahead of time.

Fig. 24 shows the MAPE for GNN and FNN. It is evident that the GNN-based approach performs better than the FNN. This can be attributed to not only the topology-awareness of the GNN approach, but also the fact that GNN have a higher number of layers and neurons. Moreover, it can be observed from Fig. 25 that the prediction accuracy of both approaches improves as the number of historical measurements used is increased, before it becomes almost constant. Finally, for a given number of historical measurements, the prediction horizon (i.e., how far into the future resource requirements can be

predicted) leads to different prediction accuracies. For short prediction horizons, the MAPE is surprisingly high, and keeps reducing as the horizon is increased. This can be explained by the fact that the neural networks were trained to make predictions at different horizons and are therefore unable to effectively adapt to the new horizons without retraining. The errors reduce until when $\tau = 20$, the point for which the networks were trained, and then start rising again.

VII. Related Work

Resource Management in NFV involves a number of sub-problems [4]. However, until now most current approaches have concentrated on the placement of servers [10] and VNFs [28]. These approaches do not consider the need to autonomously and dynamically scale the resources allocated to VNFs whose load may vary over time. As stated in the first NFV white paper [2] the automation and efficiency of such processes is of paramount importance to the success of NFV. This requires efficient and timely deployment and tear-down of resource containers on which VNFs run to match changing traffic. OpenStack - one of the most widely used VIMs provides Heat [24]. Heat is an orchestration engine that can be used to automatically launch multiple SFCs based on templates (the Heat Orchestration Templates (HOT)) which define the features of each VNF and the overall SFC topology. However, unlike our proposal, Heat does not provide any capability to predict the resource requirements of the deployed VNFs so as to automatically (ahead of time) scale in or out. The automated scaling capabilities provided by Heat use Ceilometer to create alarms based on instance CPU usage and associate actions like spinning up or terminating instances based on CPU load. While this process is able to automatically deploy a given number of machines, the scaling is based on previously set thresholds similar to the manual case used in the comparisons. Therefore, our proposal may be used to compliment Heat by providing information on when the scaling threshold is about be reached, and in so doing, to trigger the scaling process ahead of time, such that the resources are available just when needed. As explained in the implementation setup, we used Openstacks Heat in the evaluation of our proposal. While Linux containers may be used to achieve significantly lower spin-up times, there are still a number of open questions on their applicability for VNFs which require isolation [1].

Dynamic resource management is usually aimed at finding a compromise between two competing objectives: efficiency and reliability. In order to increase reliability, redundant resources should be provided as backup, which reduces resources utilisation efficiency. With regard to efficiency, a number of approaches based on control theory [29], [30], performance dynamics modeling [31] and workload prediction [32], [33] may be followed. However, such generic resource management approaches cannot be trivially applied to NFV environments due to the additional challenges that result from the need to simultaneously consider multiple resource types (such as CPU, memory, latency). Moreover, these resource types are not only segmented into many VNFCs and their connecting links, but the VNFCs may also require different quality of service guarantees.

With regard to reliability, MLDO [34] is an adaptive live migration approach which uses machine learning techniques to predict VM failures and hence perform migrations over a wide area network so as improve reliability. Similarly, Autonomic Cloud Manager (ACM) Framework [35] is an autonomic framework which uses machine learning to predict failures of virtual machines and to pro-actively redirect the load to healthy virtual machines in similar or different cloud regions.

In [36], an approach for VM workload prediction based on deep learning was proposed. The authors designed a deep belief network (DBN) composed of multiple-layered restricted Boltzmann machines (RBMs) and a regression layer. The DBN was used to extract high level features regarding the loading of a VM and the regression layer was used to predict the workload of the VMs in the future. However, the model is applicable to resource prediction for just a single VM without temporal or spatial interactions with other VM, which is not practical for most real applications. In previous related work [37] we proposed machine learning techniques for dynamic allocation of resources in network virtualisation environments. The authors model the nodes and links in a physical network as agents which use reinforcement learning to allocate resources to virtual nodes and links as requirements change. However, the nature of SFCs in NFV present additional challenges since the graphs that represent the VNFs are directed, which makes the VNFCs dependent on each other, and hence the GNN approach proposed in this article more suitable in such a scenario. Moreover, since NFV is expected to be a building block for future communications systems where both resource utilisation efficiency and reliability are mandatory requirements, there is a need to provide automated solutions which are able to provide resources just-in time. Most previous approaches are either driven by efficiency or reliability but not both as is the case in this article.

In summary, our proposal enhances the state-of-the-art in that it complements VNF placement which is quite well studied, with a way to autonomously and dynamically scale up and down the initially allocated resources. This way, resources can be reserved for VNFs only when they are needed. Moreover, GNNs as used in our proposal are well suited for such a problem due to the ability to take advantage of topology dependencies which result when the load of VNFs is dependent on that of its neighbours. To the best of our knowledge, this is the first attempt to automate resource management in NFV through machine learning, and by taking advantage of the topology of the VNF-FG.

VIII. Conclusion

In this article, we have proposed an automated, dynamic and topology-aware resource management approach for NFV environments. The proposal models each VNFC in a SFC as a pair of parametric functions which combine the observed resource utilisation profile at a given VNFC with that observed at its neighbours so as to predict future resource requirements. The predicted resource requirements can then be used to spin

up new resources or plan global resource availability for the whole system. Through evaluations using a deployment of a virtualised IMS, and using real VoIP traces, we have evaluated our proposal, and showed that it can achieve a prediction accuracy of about 90%, and is able to enhance the processing delay and call drop rate by 27% and 29% respectively. In addition, comparisons with a FNN show that the proposed approach achieves a 5% higher prediction accuracy.

However, there might be some room to improve the current system so as to have even better generalisation accuracy by considering error functions with different penalty terms. Moreover, the backpropagation through time algorithm used for training the SFC encoding network requires to store the states of each parametric function. If the SFC is large, this might require a considerable amount of memory. Therefore, future work will attempt to find more efficient ways of training the encoding network.

ACKNOWLEDGMENT

The authors are indebted to the Editor-in-Chief for coordinating the review process, and to the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

[1] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[2] R. Guerzoni, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Introductory white paper," in *Proc. SDN OpenFlow World Congr.*, Jun. 2012, pp. 1–16.

[3] 5G PPP. (Jul. 2016). *5G PPP Architecture Working Group: View on 5G Architecture*. Accessed on Nov. 16, 2016. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-July-2016.pdf

[4] R. Mijumbi *et al.*, "Management and Orche stration challenges in network functions virtualization," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[6] R. Mijumbi *et al.*, "A connectionist approach to dynamic resource management for virtualised network functions," in *Proc. 12th IEEE/IFIP/ACM Int. Conf. Netw. Service Manag. (CNSM)*, Montreal, QC, Canada, Nov. 2016, pp. 1–9.

[7] ETSI Industry Specification Group (ISG) NFV. (Dec. 2014). *ETSI GS NFV-SWA 001: Network Functions Virtualisation (NFV); Virtual Network Functions Architecture*. Accessed on Nov. 16, 2016. [Online]. Available: http:// www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf

[8] Metaswitch Networks. (Jun. 2016). *Project Clearwater*. Accessed on Nov. 16, 2016. [Online]. Available: http://clearwater.readthedocs.io/en/latest/index.html

[9] D. Ta, S. Zhou, W. Cai, X. Tang, and R. Ayani, "Network-aware server placement for highly interactive distributed virtual environments," in *Proc. 12th IEEE/ACM Int. Symp. Distrib. Simulat. Real Time Appl. (DS RT)*, Vancouver, BC, Canada, Oct. 2008, pp. 95–102.

[10] R. Mijumbi, J. Serrat, J. L. Gorricho, J. Rubio-Loyola, and S. Davy, "Server placement and assignment in virtualized radio access networks," in *Proc. 11th Int. Conf. Netw. Service Manag. (CNSM)*, Barcelona, Spain, Nov. 2015, pp. 398–401.

[11] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," in *Proc. 11th Int. Conf. Netw. Service Manag. Mini (CNSM)*, Barcelona, Spain, 2015, pp. 50–56.

[12] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manag. Workshop (CNSM)*, Rio de Janeiro, Brazil, Nov. 2014, pp. 418–423.

[13] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[14] J. Claassen, R. Koning, and P. Grosso, "Linux containers networking: Performance and scalability of kernel modules," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, Istanbul, Turkey, Apr. 2016, pp. 713–717.

[15] M. Jones *et al.*, "Scalability of VM provisioning systems," in *Proc. 20th Annu. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Waltham, MA, USA, Sep. 2016, pp. 1–5.

[16] ETSI Industry Specification Group (ISG) NFV. (Oct. 2013). *ETSI GS NFV 001 V1.1.1: Network Function Virtualization. Use Cases*. Accessed on Nov. 16, 2016. [Online]. Available: www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[17] M. A. Khamsi and W. A. Kirk, *Banach Spaces: Introduction*. New Jersey, USA: Wiley, 2001.

[18] V. D. Massa *et al.*, "A comparison between recursive neural networks and graph neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Vancouver, BC, Canada, 2006, pp. 778–785.

[19] R. Rojas, *Neural Networks: A Systematic Introduction*. New York, NY, USA: Springer-Verlag, 1996.

[20] J. C. Principe, J.-M. Kuo, and B. de Vries, "Backpropagation through time with fixed memory size requirements," in *Proc. IEEE SP Workshop Neural Netw. Process. III*, Linthicum Heights, MD, USA, Sep. 1993, pp. 207–215.

[21] R. Day. (Jun. 2016). *SIPp*. Accessed on Nov. 16, 2016. [Online]. Available: http://sipp.sourceforge.net/

[22] The Cacti Group Inc. (Jun. 2016). *Cacti*. Accessed on Nov. 16, 2016. [Online]. Available: http://www.cacti.net/

[23] Internet Systems Consortium. (Jun. 2016). *BIND*. Accessed on Nov. 16, 2016. [Online]. Available: https://www.isc.org/downloads/bind/

[24] OpenStack. (Jun. 2016). *Heat Orchestration Templates*. Accessed on Nov. 16, 2016. [Online]. Available: https://wiki.openstack.org/wiki/Heat

[25] TSTAT. (Jun. 2016). *TCP STatistic and Analysis Tool: Skype Traces*. Accessed on Nov. 16, 2016. [Online]. Available: http://tstat.polito.it/traces-skype.shtml

[26] J. Heaton, *Introduction to Neural Networks for Java*, 2nd ed. St. Louis, MO, USA: Heaton Res., 2008.

[27] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results," *Neural Netw.*, vol. 11, no. 1, pp. 15–37, 1998.

[28] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, Kraków, Poland, May 2014, pp. 1–9.

[29] C. Barna, M. Fokaefs, M. Litoiu, M. Shtern, and J. Wigglesworth, "Cloud adaptation with control theory in industrial clouds," in *Proc. IEEE Int. Conf. Cloud Eng. Workshop (IC2EW)*, Berlin, Germany, Apr. 2016, pp. 231–238.

[30] Q. Zhang *et al.*, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proc. 9th ACM Int. Conf. Auton. Comput. (ICAC)*, San Jose, CA, USA, 2012, pp. 145–154.

[31] A. Pietrabissa *et al.*, "An approximate dynamic programming approach to resource management in multi-cloud scenarios," *Int. J. Control*, vol. 90, no. 3, pp. 492–503, Jul. 2016.

[32] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 14–28, Jan./Mar. 2014.

[33] Z. Chen, Y. Zhu, Y. Di, and S. Feng, "Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network," *Comput. Intell. Neurosci.*, vol. 2015, Jan. 2015, Art. no. 17.

[34] M. Arif, A. K. Kiani, and J. Qadir, "Machine learning based optimized live virtual machine migration over wan links," *Telecommun. Syst.*, vol. 64, no. 2, pp. 245–257, 2017.

[35] A. Pellegrini, P. D. Sanzo, and D. R. Avresky, "Proactive cloud management for highly heterogeneous multi-cloud infrastructures," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Chicago, IL, USA, May 2016, pp. 1311–1318.

[36] F. Qiu, B. Zhang, and J. Guo, "A deep learning approach for VM workload prediction in the cloud," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel Distrib. Comput. (SNPD)*, Shanghai, China, May 2016, pp. 319–324.

[37] R. Mijumbi, J.-L. Gorricho, and J. Serrat, "Contributions to efficient resource management in virtual networks," in *Proc. 8th IFIP WG 6.6 Monitor. Securing Virtualized Netw. Services (AIMS)*, Brno, Czech Republic, 2014, pp. 47–51.

Rashid Mijumbi received the B.Sc. degree in electrical engineering from Makerere University, Kampala, Uganda, in 2009, and the Ph.D. degree in telecommunications engineering from the Universitat Politecnica de Catalunya (UPC), Barcelona, Spain. He was a Post-Doctoral Researcher with the UPC and the Telecommunications Software and Systems Group, Waterford, Ireland, where he participated in several Spanish national, European, and Irish national research projects, including the Flamingo EU project and the SFI CONNECT Research Centre. He is currently a Software Systems Reliability Engineer with Nokia-Bell Labs, Dublin, Ireland. His research interests are in automated management of resources and services in 5G, NFV, and SDN. He is a regular Reviewer for various journals and serves on technical program committees of leading conferences in these areas. He was a recipient of the 2016 IEEE Transactions Outstanding Reviewer Award recognizing outstanding contributions to the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.

Sidhant Hasija received the degree in electronics and communication engineering from the LNM Institute of Information Technology, India, in 2013. He is currently pursuing the master's degree with Telecommunications Software and Systems Group, Waterford Institute of Technology. He participated in the Google Summer of Code 2012 on a project targeted toward GIS integration of wireless sensor networks. In 2013, he held a summer internship with the University of Southern Lazio, Cassino, Italy, in maintaining their stack of robotic applications to include 2-D localization and mapping. He worked in Sapient Nitro, Gurgaon, India, for two years in assisting clients with Web applications, development of MS SQL Server Data warehouse, and ETL from legacy databases.

Steven Davy received the B.A. degree from Trinity College Dublin in 2003 and the Ph.D. degree from the Waterford Institute of Technology in 2008, both in computer science. He is currently the Manager of the Programmable and Autonomous Systems Research Unit, Telecommunications Software and Systems Group, Waterford Institute of Technology. His research interests include network and service management, policy-based management, and more recently virtual network resource planning and deployment. More recently, his research has been on applying his expertise in telecommunications systems in areas that include intelligent transport systems and remote healthcare.

Alan Davy received the B.Sc. (with Hons.) degree in applied computing and the Ph.D. degree from the Waterford Institute of Technology, Waterford, Ireland, in 2002 and 2008, respectively. Since 2002, he has been with the Telecommunications Software and Systems Group, originally as a students and then, since 2008, as a Post-Doctoral Researcher. In 2010, he was with IIT Madras, India, as an Assistant Professor, lecturing in network management systems. He was a recipient of the Marie Curie International Mobility Fellowship in 2010, which brought him to work at the Universitat Politècnica de Catalunya for two years. He is currently a Senior Research Fellow and a Research Unit Manager of the Emerging Networks Laboratory, Telecommunications Software and Systems Group. He is the coordinator of the EU H2020 FETOpen Project CIRCLE: Coordinating European Research on Molecular Communications.

Brendan Jennings (M'05) received the B.Eng. and Ph.D. degrees from Dublin City University, Dublin, Ireland, in 1993 and 2001, respectively. He is the Head of Graduate Studies for the Waterford Institute of Technology, Waterford, Ireland, where he is also active as a Senior Researcher within the Emerging Networks Laboratory, Telecommunications Software, and Systems Group. He has spent periods as a Visiting Researcher with the KTH Royal Institute of Technology, Stockholm, Sweden, and in EMC^2 Research Europe, Cork, Ireland. He regularly serves on the organization and technical program committees of a number of network and service management related conferences. His research interests include network management, cloud computing, and nanoscale communications.

Raouf Boutaba (F'12) received the M.Sc. and Ph.D. degrees in computer science from the Université Pierre et Marie Curie, Paris, France, in 1990 and 1994, respectively. He is currently the Associate Dean of Research with the Faculty of Mathematics and a Full Professor of Computer Science with the University of Waterloo, Canada. He was a recipient of several best paper awards and other recognitions such as the Premier's Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, the Joe LociCero and the Dan Stokesbury Awards in 2009, the Salah Aidarous Award in 2012, and the 2014 McNaughton Gold Medal, which is IEEE Canada's highest honour awarded to outstanding Canadian engineers recognized for their exemplary contributions to the engineering profession. He is a fellow of the Engineering Institute of Canada and the Canadian Academy of Engineering.