Enabling Multicast Slices in Edge Networks

Yugen Qin, Qiufen Xia*, *Member, IEEE,* Zichuan Xu, *Member, IEEE,* Pan Zhou, *Member, IEEE,* Alex Galis, *Member, IEEE,* Omer F. Rana, *Senior Member, IEEE,* Jiankang Ren, *Member, IEEE,* Guowei Wu.

***Abstract*—Telecommunication networks are undergoing a disruptive transition towards distributed mobile edge networks with virtualized network functions (VNFs) (e.g., firewalls, Intrusion Detection Systems (IDSs), and transcoders) within the proximity of users. This transition will enable network services, especially IoT applications, to be provisioned as *network slices* with sequences of VNFs, in order to guarantee the performance and security of their continuous data and control flows. In this paper we study the problems of delay-aware network slicing for multicasting traffic of IoT applications in edge networks. We first propose exact solutions by formulating the problems into Integer Linear Programs (ILPs). We further devise an approximation algorithm with an approximation ratio for the problem of delay-aware network slicing for a single multicast slice, with the objective to minimize the implementation cost of the network slice subject to its delay requirement constraint. Given multiple multicast slicing requests, we also propose an efficient heuristic that admits as many user requests as possible, through exploring the impact of a non-trivial interplay of the total computing resource demand and delay requirements. We then investigate the problem of delay-oriented network slicing with given levels of delay guarantees, considering that different types of IoT applications have different levels of delay requirements, for which we propose an efficient heuristic based on Reinforcement Learning (RL). We finally evaluate the performance of the proposed algorithms through both simulations and implementations in a real test-bed. Experimental results demonstrate that the proposed algorithms is promising.**

***Index Terms*—Network slicing; multicasting; Internet of Things; network function virtualization; throughput maximization; cost minimization; approximation algorithms.**

I. Introduction

With the development of the Internet of Things (IoT) technique, IoT applications (eg., automatic driving applications, smart home applications, and mobile phones applications) are emerging as the major services of mobile users. One fundamental functionality of IoT applications is multicast that transfers data from a source node to a given set of destinations [20], [36]. For example, a power distribution company in Australia, Energy Queensland, has a system that reduces peak demand for power by remotely turning off consumers' hot water systems via a small device installed in their meter box and controlled over their network [20]. On one hand, the data collected by such meter boxes need to be multicasted to different control stations for processing and decision. On the other hand, the control commands needs to be multicasted to many meter boxes. In addition, in virtual reality(VR)

Y. Qin, Z. Xu, and G. Wu are with the School of Software, Q. Xia is with the International School of Information Science and Engineering, Dalian University of Technology, and the Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, China, 116621. E-mails: qyg@mail.dlut.edu.cn, z.xu@dlut.edu.cn, wgwdut@dlut.edu.cn, qiufenxia@dlut.edu.cn

P. Zhou is with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science & Technology, Wuhan, 430074, China. Email: panzhou@hust.edu.cn.

A. Galis is with Department of Electronic and Electrical Engineering, Torrington Place, London WC1E 7JE, United Kingdom. Email: a.galis@ucl.ac.uk.

J. Ren is with the School of Computer Science and Technology, Dalian University of Technology, Dalian, Liaoning, China, 116024. Email: rjk@dlut.edu.cn.

O. F. Rana is with the Cardiff University, United Kingdom. Email: RanaOF@cardiff.ac.uk.

* Corresponding author: Qiufen Xia. Email: qiufenxia@dlut.edu.cn.

games, multiple VR headsets may need to transfer their data to a nearby 5G base station for processing and the processed data (or gaming data) need to be multicasted to multiple players.

To guarantee the security and performance of multicasting for IoT applications, a variety of intermediary network functions, e.g., firewalls, Intrusion Detection Systems (IDSs), proxies, and WAN optimizers, are deployed in the network. For example, video processing applications usually need various network functions, e.g., video decoding, motion detection, video frame enhancement, object detection shadow network, and object recognition deep network, to process the videos before multicasting them to users [37], [40]. Such a sequence of network functions can be considered as a *network service chain*. Conventional network functions are usually implemented in dedicated hardware, making it very expensive and inflexibly to achieve the benefits of network functions. Network Function Virtualization (NFV) [3], [4], [13], [29], [48], [49] is emerging as a promising paradigm that provides inexpensive and flexible network services, by implementing network functions as software running in Virtual Machines (VMs) or containers. In this paper, we consider the implementation of network services for multicast applications in an edge network, where each multicast request requires to process its traffic by a network slice consisting of a sequence of Virtualized Network Functions (VNFs) before reaching its set of destinations.

There are many challenges of slicing the edge network for multicast requests in IoT applications, which are referred to as *multicast slices* in edge networks [2], [8]. First, users of IoT applications have end-to-end delay requirements to guarantee that their traffic reaches their destinations in almost real-time. The experienced delay of multicast requests depends on the locations that host network slices. Naive placement of multicast slices into edge locations that are far away from its multicast group members may incur a prohibitive long delay. Second, each multicast slice has multiple VNFs to process its traffic, and such VNFs can be placed into multiple cloudlets for a better delay or lower implementation cost. This brings the difficulties of multicast slicing into a new dimension, as different combinations of the VNFs in a multicast slice can increase the solution space dramatically. Specific challenges include (1) how to jointly find one or multiple cloudlets to implement the VNFs of a network slice and a multicast tree for each incoming multicast request, subject to the computing demands and delay requirements of requests, (2) how to smartly determine the combinations of VNFs of a multicast slice that can be placed together into a single cloudlet, such that the cost of implementing the request is minimized while its delay requirement is met, (3) given a set of multicast groups without the knowledge of the multicast requests in each group, how to smartly determine the number of slices of different delay-guarantees is a key problem in the edge network, and (4) how to devise an approximation algorithm with a provable approximation ratio to minimize the cost of implementing each admitted multicast request, such that the distance of the approximate solution to the optimal one is bounded.

Most studies on multicasting in conventional networks or software-defined networks do not consider the service chain requirement of each user request [18], [19], [56], [57]. The solutions of these studies thus cannot be directly applied to NFV-enabled multicasting, due to the lack of efficient methods of jointly finding locations for VNFs and multicast trees. There are a few recent studies on NFV-enabled multicasting problem. For example, Zhang *et al.* [56], [57] investigated the NFV-enabled multicast problem by assuming that there are sufficient computing and bandwidth resources in an SDN to accommodate a multicast request. Xu *et al.* [50] investigated the

problem of NFV-enabled multicasting, by devising an approximation algorithm with a provable approximation ratio for realizing a single NFV-enabled multicast request and an online algorithm with a guaranteed competitive ratio for the online NFV-enabled multicasting problem. They however do not consider the delay requirements of multicast requests. Although Ren, Xu, and Yu *et al.* considered the delay-aware NFV-enabled multicasting [54], [43], [53], dynamic provisions of multicast slices with different delay guarantees for different multicast groups is ignored.

To the best of our knowledge, we are the first to consider the problems of delay-aware network slicing for multicast requests in edge networks with the aim to either minimize its implementation cost or maximize the network throughput. The major contributions of this paper are summarized as follows.

- We give optimal solutions to the delay-aware network slicing problems by formulating them into Integer Linear Programs (ILPs).
- We then devise the very first approximation algorithm with an approximation ratio of $1 + \epsilon$ for minimizing the implementation cost of the request, where ϵ is an accuracy in the approximation algorithm that finds the delay-constraint shortest path in a graph [25]. We also propose an efficient heuristic for the delay-aware network slicing for multicast in an edge network, if the cloudlets have limited computing resource to implement the VNFs of a given set of multicast requests arrived in the system.
- Given a set of multicast groups without the knowledge of their future requests, we consider the delay-oriented network slicing problem with a set of given levels of delay guarantees. We propose a dynamic framework and a learning-based algorithm to dynamically adjust the number of different multicast slices with different delay-guarantees in the system.

The rest of the paper is organized as follows. Section II reviews the related work. Section III introduces the system model, notations, and problem definitions. Section IV proposes exact solutions for the delay-aware network slicing problems. Section V devises an approximation algorithm for the delay-aware network slicing for a single multicast request without resource capacity constraints. Section VI develops an efficient heuristic algorithm for the delay-aware network slicing problem for multiple multicast requests with resource constraints of cloudlets in an IoT edge network. Section VII proposes a learning-based heuristic for the delay-oriented network slicing problem with levels of delay requirements in an IoT edge network without the knowledge of future arrivals of requests. Section VIII and Section IX evaluate the performance of the proposed algorithms by both experimental simulations and implementations in a real test-bed, respectively. Section X concludes the paper and future work.

II. Related Work

Service chaining has gained much attention in the past few years, it however still remains the most challenging problems in the deployment and management of NFV-enabled Software-Defined Networks (SDNs). In service chaining, one fundamental question is how to chain various instances of VNFs together to offer services for users and how to route traffic among the VNFs. Therefore, NFV-enabled routing and traffic steering have attracted much attention from the literature [3], [4], [16], [17], [19], [24], [27], [33], [49], [50], [55]. These studies can be classified into two categories: (1) unicasting, and (2) multicasting. For the investigations on unicasting, most of them focus on hybrid networks with both hardware and software network functions [33], online algorithm

design for dynamic networks [12], [24], [27], and delay-awareness [22], by proposing exact solutions [24], approximation solutions [4], [52], heuristics [33], [49], [52], online algorithms [19], or game theory based solutions [6].

Most studies on QoS-aware multicasting focus on the traffic steering in conventional wired or wireless networks, and there exist many excellent solutions [1], [18], [19], [32]. Recently, with the emerging of new networking technologies such as SDN and NFV, multicasting has re-gained the attention of many researchers, as the application of traditional multicasting solutions is not a straightforward process. Specifically, there are several studies that focused on multicasting in SDNs [18], [19]. Huang *et al.* [19] studied the online multicasting in software-defined networks with both node and link capacity constraints, by devising the very first online algorithms with provable competitive ratios. Huang *et al.* [18] studied the scalability problem of multicasting in SDNs, by proposing an efficient algorithm to find a branch-aware Steiner Tree (BST) for each multicast request. These solutions however cannot be directly applied to the problem of NFV-enabled multicasting in cloud networks, because they ignore the service chain requirements of multicast requests. Simple application of these solutions may cause the traffic of multicast requests being forwarded to destinations without being processed by their service chains.

Studies that investigated network slicing and NFV-enabled multicasting include the ones due to Leconte *et al.* [23], Zhang *et al.* [56], [57], Xu *et al* [50], Soni *et al.* [45], Ren *et al.* [41], [42], and Yu *et al.* [54]. Specifically, Leconte *et al.* proposed a resource allocation framework for network slicing. Multicasting is not considered in the paper. Zhang *et al.* [56], [57] investigated the NFV-enabled multicasting problem in an SDN without considering resource capacities in the SDN. They assumed that data traffic of each multicast request can only be processed by one server. Xu *et al.* [50] considered the NFV multicasting problem by assuming the traffic of each request can be processed by multiple servers , as long as the implementation cost can be improved. Approximation and online algorithms are proposed. They however do not consider the chaining of VNFs by assuming the VNFs in each service chain is consolidated into a single cloudlet. Later, Xu *et al.* [53] studied the problem of NFV-enabled multicasting by considering the resource sharing among requests. Ren *et al.* [41], [42] investigated the problem of embedding a service graph that consisting instances of VNFs into the substrate network. Soni *et al.* [45] proposed a scalable multicast group management scheme and a load balancing method for the routing of best-effort traffic and bandwidth-guaranteed traffic. These studies however do not consider the delay requirements of multicast requests.

III. Preliminaries

In this section, we first introduce the system model, notations and notions. We then define the problems precisely.

A. System model

We consider an edge network $G = (V, E)$ with a set V of switches and cloudlets that are deployed within the proximity of IoT service users. There is a set of cloudlets in G that can implement various VNFs running on its commodity servers, and a set E of links between switches and the cloudlets. Let V_{CL} ($\subseteq V$) be the subset of switches attached with cloudlets. Due to space limitation of the places that deploy cloudlets, each cloudlet usually has a computing capacity. Denote by C_v the computing capacity of the cloudlet attached to switch $v \in V_{CL}$. There is a transmission delay in each link $e \in E$ when user traffic is transmitted via it. Let d_e be the delay of

transmitting a unit data traffic via link $e \in E$. Fig. 1 is an example of the edge network for IoT applications, where two multicast slices with delay guarantees are deployed in G.

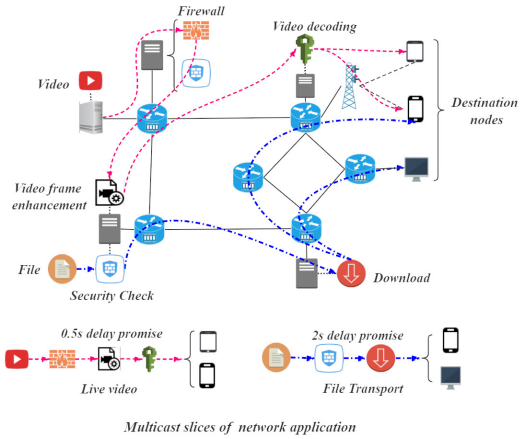


Fig. 1. An IoT edge network G and a multicast slice.

B. Multicast slices, multicast groups, and multicast requests

We consider multicast requests that require to transfer their traffic from a source node to a given set of destinations. Each multicast request requires a network slice to process its data traffic. Denote by r_k a NFV-enabled multicast request r_k that can be represented by a quadruple $r_k = (s_k, D_k; b_k, NS_k)$, where $s_k \in V$ is the source, D_k is the set of destinations $D_k \subseteq V$, b_k is the size of data that needs to be transferred to its destinations, and NS_k is the *multicast slice* of r_k that consists of a sequence of VNFs. We also consider the set of nodes in $s_k \cup \{D_k\}$ as a *multicast group*, denoted by $\mathcal{G}$. Each multicast group may have multiple multicast requests with each having a member multicasting its traffic to the rest members.

Assume that there are L_k VNFs in multicast slice NS_k of request r_k, where $1 \leq l \leq L_k$ for each NS_k. To implement r_k, its multicast slice NS_k enforces every message from source s_k of r_k to go through each VNF $f_l \in NS_k$ in the specified order prior to reaching destinations in D_k, as illustrated in Fig. 1. To this end, the VNFs of NS_k must be assigned to cloudlets and chained together. We assume that the VNFs of NS_k may be placed into multiple cloudlets, because a single cloudlet may not have enough computing resource to implement all VNFs of NS_k. Denote by $C_v(f_l)$ the amount of computing resource demanded by VNF f_l to process unit data traffic in cloudlet $v \in V_{CL}$. The computing resource demand of $f_l \in NS_k$ thus is $b_k \cdot C_v(f_l)$, and the total computing resource demand of request r_k is the accumulative resource demand of all the network functions in its network slice NS_k. To implement each r_k with NS_k, its traffic needs to be transferred from source s_k to the placed VNFs of multicast slice NS_k and then multicasted to its destinations in D_k. Following the study by Xu *et al.* [50], we adopt the concept of *a pseudo-multicast tree* to refer to such a tree for each multicast request r_k. The reason is that the pre-processing traffic and post-processing traffic of r_k may share the same links or switches. Such a tree is actually not a traditional multicast tree. The pseudo-multicast tree is used to describe the multicast tree that first transfers the traffic from its source to the VNFs for processing and then transfers the processed traffic to its destinations. In the rest of the paper, we call a tree by either multicast or pseudo-multicast tree, if no confusions arise.

C. Delay requirements of multicast requests

Multicast request r_k requires to transfer an amount b_k of data to its destinations in D_k within a given delay requirement. We here consider an end-to-end delay of r_k that is defined as the delay experienced by it from its source s_k to its destinations D_k, consisting of the processing delay in each VNF $f_l \in NS_k$ and the transmission delays along the paths that transfer the traffic from its source to the destinations. Let T_k be the pseudo-multicast tree that transfers the data traffic.

For the processing delay, considering that the VNFs in NS_k may be placed into multiple cloudlets, the traffic of r_k will be forwarded to its destinations in D_k after being processed by the final VNF in NS_k, i.e., f_{L_k}. Let $y_{k,l,v}$ be a binary decision variable that shows whether VNF $f_l \in NS_k$ of r_k is assigned to cloudlet $v \in V_{CL}$ for processing. The processing delay d_k^p experienced by request r_k is

$$d_k^p = \sum_{f_l \in NS_k} \sum_{v \in V_{CL}} y_{k,l,v} \cdot d_p(v, f_l) \cdot b_k, \tag{1}$$

where $d_p(v, f_l)$ is the delay of processing a unit amount of data by VNF f_l in cloudlet $v \in V_{CL}$.

The transmission delay in T_k is the delay from the start of transmission until all destinations in D_k finish receiving the data. Let $d_k^{t,m}$ be the transmission delay of request r_k from s_k to one of its destinations t_m.

The delay experienced by r_k thus is

$$d_k = d_k^p + \arg \max_{t_m \in D_k} d_k^{t,m}, \tag{2}$$

which needs to be no greater than its specified delay requirement d_k^{req}, i.e.,

$$d_k \leq d_k^{req}. \tag{3}$$

D. Cost models

As the network operator of G charges each admitted multicast request based on its resource usage, the major concern of the operator is its *operational cost* that is defined as the sum of the costs of its computing and bandwidth resource consumptions for the multicast slices used to implement multicast requests. Let $c(e)$ and $c(v)$ be the costs of using one unit of bandwidth and computing resources at link $e \in E$ and cloudlet $v \in V_{CL}$, respectively. Denote by $q_{e,k}^{pre}$ by an indicator variable that indicates whether link $e \in E$ transfers the pre-processed traffic by VNF $f_1 \in NS_k$ of r_k. Recall that the traffic may be forwarded to multiple locations for processing if VNFs in NS_k is placed into multiple locations. The traffic between two consecutive VNFs in NS_k may need to be transferred along the edges in G. Thus, denote by $q_{e,k,l}^{pro}$ an indicator variable that shows whether edge e is used to transfer the processed traffic by $f_l \in NS_k$. Denote by $\mathcal{P}$ a set of all possible paths from cloudlets in V_{CL} to the destinations in D_k, which could be pre-computed in the network planning stage. The operational cost due to implementing r_k thus is

$$c_k = \sum_{f_l \in NS_k} \sum_{v \in V_{CL}} y_{k,l,v} \cdot c(v) \cdot b_k + \sum_{e \in E} q_{e,k}^{pre} \cdot c(e) \cdot b_k + \sum_{l=1}^{L_k - 1} \sum_{e \in E} q_{e,k,l}^{pro} \cdot c(e) \cdot b_k + \sum_{p \in \mathcal{P}} z_{v,m}^{post} \sum_{e \in p} c_e \cdot b_k \tag{4}$$

E. Problem definitions

Given an edge network $G = (V, E)$ for IoT applications with a set V_{CL} of cloudlets and a multicast request r_k $(= (s_k, T_k; b_k, NS_k))$, we consider the following three *delay-aware network slicing problems*.

Problem 1: Assuming that the computing resource in each cloudlet is abundant to implement a multicast request r_k, the *delay-aware network slicing problem for a single multicast request* without computing resource capacity in IoT edge network G for a single NFV-enabled multicast request r_k is to create a network slice for r_k by jointly placing the VNFs of service chain NS_k of r_k to cloudlets in V_{CL} and finding routing paths for r_k, such that the implementation cost of multicast request r_k in the created network slice is minimized, if the VNFs in its network slice NS_k can be assigned to multiple cloudlets, subject to its delay requirement d_k^{req}.

Problem 2: Assume that the computing resource in each cloudlet $v \in V_{CL}$ of G is capacitated for a given set R of multicast requests. For each multicast request in R, the IoT edge network may or may not have enough resources at that moment to create a network slice for it. We here define the *delay-aware network slicing problem*

for multiple multicast requests in an IoT edge network $G = (V, E)$ for a given set R of requests, which is to create a number of network slices with the aim to admit as many requests as possible while minimizing the operational cost due to network slice creation, by jointly placing the VNFs of each network slice NS_k and finding a multicast tree in G for each admitted multicast request r_k, subject to computing capacity constraints on cloudlets of G and delay requirement constraints of multicast requests.

Problem 3: So far we assumed that user requests have their specified delay requirements, some users however may not know how to determine a specific delay requirement. In most cases, network service providers provide a set of network slices with different levels of delay guarantees, such that each user can select a slice with its preferred delay guarantee. For example, network slices for vehicular applications may share the same level of delay requirements, e.g., response within 50 milliseconds. On the other hand, VR services are extremely sensitive to network latency. Users may experience dizziness if their viewing experience is repeatedly hindered by excessive latency. Therefore, it is essential to keep the motion-to-photon latency to less than 20 milliseconds. The network slices with 20 ms delay guarantee can be considered as the first level of delay guarantees, while the network slices within 50 ms delay guarantees are the second level of delay guarantee. We may also have another level of delay guarantee of 100 to 500 ms. Therefore, given a set of multicast slices that are already serving user requests in the network G, the network operator needs to decide the number of slices to be created in the next time slot. Assuming that time is divided into equal slots, the current time slot is denoted by t. Let O be the number of levels of delay guarantees, and $d^{req,o}$ be the oth delay guarantees with $1 \leq o \leq O$. Specifically, we assume that each of such delay guarantee is for a unit amount of data traffic. Users could select their preferred multicast slice according to their data traffic. *The delay-oriented network slicing problem with levels of delay requirements* is to dynamically adjust the number of multicast slices for each level of delay guarantees, such that as many user requests are admitted while meeting the capacity constraints of cloudlets, by allowing users to select their preferred network slice with a level of guaranteed delay.

All the defined problems are NP-hard, as even their special case – the traditional multicast problem without the network slicing requirement is NP-hard [9]. Since the problems are NP-Hard, we aim to devise approximation algorithms with a guarantee of the distance from the optimal solution and efficient heuristics that smartly implement the multicast requests. Given a value $\gamma \geq 1$, a *γ-approximation algorithm* for a minimization problem P_1 is a polynomial time algorithm $\mathcal{A}$ that outputs a solution whose value is no more than γ times the value of an optimal solution for any instance I of P_1, where γ is the approximation ratio of algorithm $\mathcal{A}$.

For the sake of clarity, we summarize the symbols used in this paper in the Table II.

IV. Integer Linear Programs for the Delay-Aware Network Slicing Problems

We here propose optimal solutions for the delay-aware network slicing problem for a single multicast request without computing resource capacity and the delay-aware network slicing problem via integer linear programs.

A. ILP for the delay-aware network slicing problem for a single multicast request

The delay-aware network slicing problem without computing resource capacity deals with a single multicast request r_k and aims to minimize the implementation cost of the multicast request r_k. Recall that we use a binary variable $y_{k,l,v}$ to show whether VNF $f_l \in NS_k$

TABLE I
SYMBOLS

Symbols	Meaning
$G = (V, E)$	a software-defined network (SDN) with a set V of SDN-enabled switches and a set E of link that interconnect the switches
V_{CL}	a set of switches, each of which has a cloudlet being attached
v	$v \in V_{CL}$ or V
e and d_e	a link $e \in E$ and the delay of transmitting a unit data traffic via link $e \in E$
$r_k = (s_k, D_k; b_k, NS_k)$	a NFV-enabled multicast request, with a source node $s_k \in V$, a set D_k of destinations, an amount b_k of data that needs to be transferred to its destinations in D_k, and network slice NS_k
L_k and f_l	the number of VNFs in network slice NS_k and its lth network function
$C_v(f_l)$	the amount of computing resource demanded by network function f_l to process unit data traffic in cloudlet $v \in V_{CL}$
T_k	the multicast tree that transfers the data traffic of request r_k
d_k^p	the processing delay experienced by request r_k
$d_p(v, f_l)$	the delay of processing a unit amount of data by VNF f_l in cloudlet $v \in V_{CL}$
$z_{e,k,m}^{pre}$ and $z_{e,k,m}^{post}$	binary indicator variables that shows whether link $e \in E$ is used to transfer r_k's pre- and post- processed traffic by the final VNF $f_{L_k} \in NS_k$
$d_k^{t,m}$	the transmission delay experienced by request r_k from s_k to t_m ($\in D_k$) of request r_k
d_k	the delay experienced by multicast request r_k
d_k^{req}	the specified delay requirement of multicast request r_k
$c(e)$ and $c(v)$	the usage costs of one unit of bandwidth and computing resources at link $e \in E$ and server $v \in V_S$, respectively
$q_{e,k}^{pre}$ and $q_{e,k}^{post}$	binary indicator variables that indicate whether link $e \in E$ transfers the pre-processed and post-processed traffic by VNF $f_{L_k} \in NS_k$ of multicast request r_k
$y_{k,l,v}$	a binary indicator variable that shows whether VNF $f_l \in NS_k$ of multicast request r_k is assigned to the cloudlet that is attached to $v \in V_{CL}$ for processing
c_k	the implementation cost of multicast request r_k in the created network slice
$p_{v,k}^{pre}$ and $p_{v,k}^{post}$	binary indicator variables that show whether switch $v \in V$ is used to forward the pre- and post-processed traffic of r_k. Let $\delta(v)$ denote the incident edges of switch node $v \in V$
$G' = (V', E')$	an auxiliary graph constructed based on the original network G.
$v'_{k,l}$ and $v''_{k,l}$	a pair of virtual cloudlets in the auxiliary graph for each cloudlet $v \in V_{CL}$
OPT'	the optimal solution to the delay-constraint shortest path in auxiliary graph G'
OPT	the optimal solution to the delay-aware NFV-enabled multicasting problem without computing capacity
$Pri(r_k)$	the priority of admitting a multicast request r_k
t	a time slot
R and R^t	a set of multicast requests and a set of multicast requests in time slot t
$r_k^t = (s_k^t, D_k^t; b_k^t, NS_k^t)$	a NFV-enabled multicast request in time slot t, with a source node $s_k^t \in V$, a set D_k^t of destinations, an amount b_k^t of data that needs to be transferred to its destinations in D_k^t, and network slice NS_k^t in time slot t
d_k^t	the processing delay experienced by request r_k^t
ϑ and θ	the request admit rate and a threshold of the acceptable request admit rate
s_t and a_t	the state of reinforcement learning (RL) algorithm in time slot t and the action of RL algorithm in time slot t
$Q(s_t, a_t)$ and $rwd(s_t)$	the Q-value of reinforcement learning algorithm with state s_t and action a_t, and the reward of the reinforcement learning algorithm with state s_t.

of multicast request r_k is assigned to cloudlet $v \in V_{CL}$. Let $q_{e,k}^{pre}$ denote whether link $e \in E$ transfers the pre-processed traffic of r_k. We further let $z_{v,k}^{pre}$ be binary indicator variable that shows whether switch $v \in V$ is used to forward the pre-processed traffic of r_k. Similarly, we use $z_{v,k,l}^{pro}$ to show whether switch v is used to forward the traffic processed by $f_l \in NS_k$, where $1 \leq l \leq L_k - 1$. Let $z_{p,m}^{post}$ be binary indicator variables that show whether path $p \in \mathcal{P}$ is used to forward the post-processed traffic of r_k from cloudlet $v \in V_{CL}$ to t_m. Let $\delta(v)$ denote the incident edges of switch node $v \in V$, respectively. The objective of the ILP thus is

$$\textbf{ILP1}: \quad \min \quad c_k \tag{5}$$

subject to the following constraints.

$$\sum\nolimits_{v \in V_{CL}} y_{k,l,v} = 1, \qquad \text{for each of } f_l \in NS_k \tag{6}$$

$$\sum\nolimits_{e \in \delta(s_k)} q_{e,k}^{pre} = 1 \tag{7}$$

$$\sum_{e \in \delta(v)} q_{e,k}^{pre} \geq z_{v,k}^{pre}, \qquad \text{for each switch } v \in V \tag{8}$$

$$\sum_{e\in\delta(v)} q^{pre}_{e,k} \le 2\cdot z^{pre}_{v,k}, \qquad \text{for each switch } v\in V \tag{9}$$

$$\sum_{v\in V_{CL}} y_{k,1,v} \sum_{e\in\delta(v)} q^{pre}_{e,k} = 1 \tag{10}$$

$$\sum_{v\in V_{CL}} y_{k,l,v} \sum_{e\in\delta(v)} q^{pro}_{e,k,l} = 1 \tag{11}$$

$$\sum_{e\in\delta(v)} q^{pro}_{e,k,l} \ge z^{pro}_{v,k,l}, \qquad v\in V \text{ and } 1\le l\le L_k-1 \tag{12}$$

$$\sum_{e\in\delta(v)} q^{pro}_{e,k,l} \le 2\cdot z^{pro}_{v,k,l}, \quad v\in V \text{ and } 1\le l\le L_k-1 \tag{13}$$

$$\sum_{v\in V_{CL}} y_{k,l+1,v} \sum_{e\in\delta(v)} q^{pro}_{e,k,l+1} = 1, \qquad 1\le l\le L_k-1 \tag{14}$$

$$\sum_{e\in\delta(v)} q^{pro}_{e,k,l} \ge y_{k,l,v}, \quad v\in V_{CL} \text{ and } 1\le l\le L_k-1 \tag{15}$$

$$\sum_{p\in\mathcal{P}} z^{post}_{p,m} = 1, \quad \text{for each } v\in V_{CL} \text{ and each } t_m\in D_k \tag{16}$$

$$\sum_{f_l\in NS_k}\sum_{v\in V_{CL}} y_{k,l,v}\cdot d_p(v)\cdot b_k + \sum_{e\in E} z^{pre}_{e,k,m}\cdot d_e\cdot b_k + \sum_{p\in\mathcal{P}} z^{post}_{v,m}\sum_{e\in p} d_e\cdot b_k \le d^{req}_k, \text{ for each } t_m\in D_k \tag{17}$$

$$y_{k,l,v},\ q^{pre}_{e,k},\ q^{pro}_{e,k,l},\ z^{pre}_{v,k},\ z^{pro}_{v,k,l},\ z^{post}_{p,m}\in\{0,1\}. \tag{18}$$

Constraint (6) indicates that each of the VNF in NS_k can only be assigned to a cloudlet to process the traffic of r_k. Constraint (7) shows that there has to be one link that routes the traffic of r_k out of its source s_k. Constraints (8) and (9) jointly show that if a switch is used to forward the pre-processed traffic by $f_1\in NS_k$ of r_k, there has to be at least one and at most two of the incident edges that are used to route the traffic in and out of switch $v\in V$. Constraint (10) says that the pre-processed traffic by f_1 has to go to a cloudlet $v\in V_{CL}$, if f_1 is placed into v (i.e., $y_{k,1,v}=1$ for $v\in V_{CL}$). Similarly, constraint (11) guarantees that the traffic processed by f_l has to start with the assigned cloudlet of f_l; that is, there has to be an edge of $v\in V_{CL}$ routing the processed traffic of f_l to the next network function if $y_{k,l,v}=1$. Constraints (12), (13), and (14) have the same meanings as those of constraints (8) (9), and (10). The only difference is that constraints (12), (13), and (14) are enforced on the traffic processed by function $f_l\in NS_k$. Constraint (15) makes sure that if VNF f_l of NS_k is placed to cloudlet $v\in V_{CL}$, there will be at least one of its incident edges that are used to route the traffic to/from the cloudlet. Constraint (16) shows that one of the paths from cloudlets to destinations in D_k have to be selected to route the post-processed traffic of r_k. Constraint (17) enforces the delay requirement of multicast request r_k. Constraint (18) makes sure that each of the decision variables is an indicator variable with its value being either 1 or 0.

B. ILP for the delay-aware network slicing problem

The objective of the delay-aware network slicing problem is to maximize the number of multicast requests that can be admitted, given the capacity constraints of cloudlets. We thus introduce a binary indicator variable x_k to indicate whether request r_k is admitted or not. The objective of the problem thus is

$$\textbf{ILP2}: \qquad \max_{r_k\in R} \quad x_k, \tag{19}$$

subject to constraints (6), (7), (8), (9), (10), (11), (12), (13), (14), (15), (16), (17), (18), and the following additional constraints.

$$\sum_{v\in V_{CL}} y_{k,l,v} = x_k \tag{20}$$

$$\sum_{r_k \in R} \Big(\sum_{f_l \in NS_k} \sum_{v \in V_{CL}} y_{k,l,v} \cdot c(v) \cdot b_k + \sum_{e \in E} q^{pre}_{e,k} \cdot c(e) \cdot b_k + \sum_{l=1}^{L_k - 1} \sum_{e \in E} q^{pro}_{e,k,l} \cdot c(e) \cdot b_k + \sum_{p \in \mathcal{P}} z^{post}_{v,m} \sum_{e \in p} c_e \cdot b_k \Big) \leq B \quad (21)$$

$$\sum_{r_k \in R} \sum_{f_l \in NS_k} y_{k,l,v} \cdot b_k \cdot C_v(f_l) \leq C_v \quad (22)$$

$$x_k \in \{0, 1\}, \quad (23)$$

where constraint (20) says that each of the VNF in NS_k can only be assigned to a cloudlet to process the traffic of r_k if r_k is admitted. Since we aim to maximize the number of admitted requests while minimizing the total implementation cost of all admitted requests, we use constraint (21) to make sure that the total implementation cost of admitted multicast requests is no greater than a given budget. As long as the budget B is small enough, the cost of implementing admitted multicast requests can be minimized. Constraint (22) guarantees that the computing capacity of each cloudlet $v \in V_{CL}$ is no greater than the accumulative allocated computing resources to its its assigned VNFs of the admitted requests.

V. An Approximation Algorithm for the Delay-Aware Network Slicing Problem for a Single Multicast Request

In this section we deal with the delay-aware network slicing problem for a single multicast request without computing resource capacity constraints, by devising an efficient approximation algorithm with an approximation ratio.

A. Overview

The most challenging part of devising an approximation algorithm for the problem is how to jointly place the VNFs in each network slice NS_k into several cloudlets if necessary and find the routing paths for the request such as its delay requirement is met. We address this challenge by proposing a smart construction of an auxiliary graph $G' = (V', E')$ based on the original network $G = (V, E)$, and the original problem is transferred to a problem of finding a delay-constraint shortest path in the auxiliary graph G'.

B. Approximation algorithm

We now describe the approximation algorithm by first constructing the auxiliary graph G' and then elaborate on the algorithm.

Minimizing the implementation cost of each multicast request is to jointly minimize both the processing and transmission costs. Also, VNFs in NS_k can be placed into several cloudlets to make sure they are close to both the source and destinations, thereby increasing the probability of meeting the delay requirement of multicast request r_k. On the other hand, several of them can be placed together to save the transmission cost incurred on edges. To reflect such properties, the basic motivation of the construction of auxiliary graph is to jointly consider the processing, transmission costs of a NFV-enabled multicast request, and the service chaining requirement of the request. To jointly consider the processing and transmission costs, we create a pair of virtual nodes in the auxiliary graph for each cloudlet of the original network. We then move the processing costs to the edges of the auxiliary graph, and uniformly consider processing and transmission costs as "edge costs" in the edges of the auxiliary graph. For the service chaining requirement, we duplicate cloudlets in the original network for each VNF in a network slice, and connect those cloudlets according to connections in the original network.

We construct the auxiliary graph as follows.

We first add auxiliary nodes into the auxiliary graph G'. Specifically, we create L_k pairs of *virtual cloudlets* for each cloudlet $v \in V_{CL}$, each pair representing the

lth VNF in NS_k is placed in cloudlet v. Let $v'_{k,l}$ and $v''_{k,l}$ be such a pair of virtual cloudlets for the lth VNF and cloudlet v, and we add them into node set V' of the auxiliary graph G', i.e., $V' \leftarrow V' \cup \{v'_{k,l}, v''_{k,l}\}$. The source node s_k of r_k is also added into the node set V' of auxiliary graph G'. The set of destination nodes in D_k is considered as a single *virtual sink* and added into set V', i.e., $V' \leftarrow V' \cup \{s_k, D_k\}$.

We then connect the nodes in G' as follows.

- First, to make sure the processing and transmission costs are considered jointly. We move the the processing costs into edge weights in auxiliary graph G'. Specifically, for each VNF f_l in NS_k, we add an edge from $v'_{k,l}$ to $v''_{k,l}$ and set its weight as the processing cost of a unit data by VNF f_l in cloudlet $v \in V_{CL}$, i.e., $c(v)$. Therefore, if the data of r_k is processed by VNF f_l in cloudlet v, its traffic will traverse edge $\langle v'_{k,l}, v''_{k,l}\rangle$ in auxiliary graph G';
- Second, VNFs in NS_k can be assigned to different cloudlets. To reflect this case in the auxiliary graph, we here connect the nodes in V'. Specifically, for each l with $1 \leq l \leq L_k - 1$ and each pair of cloudlets v and u, there is an edge in E' from $v''_{k,l}$ to $u'_{k',l+1}$, i.e., $\langle v''_{k,l}, u'_{k',l+1}\rangle$. Its cost and delay are set to the transmission cost and delay of the amount b_k of data from cloudlet v to u in network G, i.e., $\sum_{e \in p_{v,u}} c(e) \cdot b_k$ and $\sum_{e \in p_{v,u}} d_e \cdot b_k$;
- Third, to allow some of the VNFs of each network slice NS_k being consolidated into a single cloudlet to save transmission cost, we connect the nodes that represent the same cloudlet. Specifically, we connect the virtual cloudlets of each cloudlet $v \in V_{CL}$. There is an edge $\langle v''_{k,l}, v'_{k,l+1}\rangle$ from node $v''_{k,l}$ to node $v'_{k,l+1}$ for each l with $1 \leq l \leq L_k - 1$. This means that VNFs f_l and f_{l+1} will both be placed to cloudlet $v \in V$, if the traffic of r_k traverses edge $\langle v''_{k,l}, v'_{k,l+1}\rangle$. Since VNFs f_l and f_{l+1} are placed into the same cloudlet, there is no transmission cost and delay incurred in links of the network G, we set the cost and delay of edge $\langle v''_{k,l}, v'_{k,l+1}\rangle$ to zero;
- We finally connect the source node, virtual cloudlet nodes, and the virtual sink. There is an edge from the source s_k of multicast request r_k to the set of virtual cloudlets that represent the first VNF $f_1 \in NS_k$, i.e., $\{v'_{k,1} \mid 1 \leq k \leq |V_{CL}|\}$. That is, there is an edge $\langle s_k, v'_{k,1}\rangle$ for each k with $1 \leq k \leq |V_{CL}|$. This edge denotes the shortest path from source s_k to cloudlet v in the original network G. Its cost is set to the accumulative cost of all the edges in the shortest path, and the delay is the total transfer delay of amount b_k of data along the path. In addition, the processed traffic only will be forwarded to the destinations in D_k after being processed by the final VNF f_{L_k} in NS_k. We thus add an edge from each v''_{k,L_k} to node D_k. The cost of edge $\langle v''_{k,L_k}, D_k\rangle$ is set to the total weight of all edges in the Steiner tree that spans the nodes in $\{v\} \cup D_k$ of the original network G, and the delay along this edge is the maximum delay of a branch of the Steiner tree that transfers the data of r_k to one of its destinations.

An example of the constructed auxiliary graph for the problem of finding a multicast tree for a multicast request is shown in Fig. 2. The delay-aware network slicing problem for a single multicast request without computing capacity constraint thus is transferred to the problem of finding a delay-constraint shortest path from node s_k to node D_k in the auxiliary graph G'. The feasible solution to the later will return a feasible solution to the original problem.

Let p' be the delay-constraint shortest path from s_k to D_k in G'. We now construct the multicast tree T_k in G for multicast request r_k. Specifically, we replace each

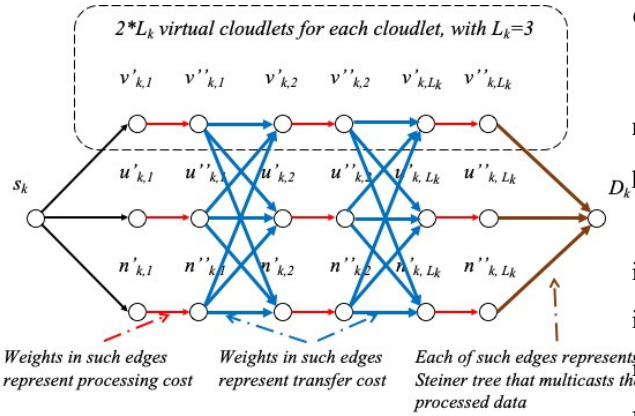


Fig. 2. An example of the auxiliary graph $G' = (V', E')$. Note that an edge $\langle v'_{k,l}, v''_{k,l} \rangle$ represents the processing of request r_k's data by the lth VNF of the request in cloudlet v, and its weight is set to the processing cost. Similarly, an edge $\langle v''_{k,l}, u'_{k,l+1} \rangle$ denotes that the lth and $l+1$th VNF of request r_k are placed to cloudlets v and u, respectively.

edge in p' with its corresponding shortest path in the original network G. For example, edge $\langle s_k, v'_{k,1} \rangle$ for each cloudlet v is replaced by the shortest path from source s_k to cloudlet v in G. Edge $\langle v''_{k,l}, u'_{k,l+1} \rangle$ is replaced by the shortest path from cloudlet v to cloudlet u. In addition, edge $\langle v''_{k,L_k}, D_k \rangle$ is replaced by the Steiner tree in G that spans the nodes in $\{v\} \cup D_k$. The detailed steps of the proposed approximation algorithm are illustrated in **Algorithm** 1.

Algorithm 1 `Appro_Multicast`

Input: $G = (V, E)$, V_{CL}, C_e for each $e \in E$, C_v for each $v \in V_{CL}$, and a multicast request $r_k = (s_k, D_k; b_k, NS_k)$.

Output: The locations for the VNFs in network slice NS_k of multicast request r_k and the multicast tree T_k to transfer the data of r_k.

1: For each cloudlet $v \in V_{CL}$, find a minimum-cost Steiner tree in network G that spans nodes in $\{v\} \cup D_k$, and let T'_k be the found Steiner tree;
2: Construct an auxiliary graph $G' = (V, E)$, by creating L_k pairs of virtual cloudlets for each cloudlet $v \in V_{CL}$, adding the source node s_k and the destination node D_k into V', connecting the nodes in V', and setting the edge costs and delays, as illustrated in Fig. 2.
3: Find a delay constraint shortest path p' from node s_k to node D_k in the auxiliary graph G', by invoking the algorithm in [25].
4: Replace each edge $\langle s_k, v'_{k,1} \rangle$ for each cloudlet v by the shortest path from source s_k to cloudlet v in network G;
5: Replace each edge $\langle v''_{k,l}, u'_{k,l+1} \rangle$ by the shortest path from cloudlet v to cloudlet u;
6: Replace edge $\langle v''_{k,L_k}, D_k \rangle$ by the Steiner tree in G that spans the nodes in $\{v\} \cup D_k$;
7: Merge each pair of nodes $v'_{k,l}$ and $v''_{k,l}$ and delete edge $\langle v'_{k,l}, v''_{k,l} \rangle$;
8: Merge all virtual cloudlets of each cloudlet $v \in V_{CL}$;
9: Return the final multicast tree T_k;

C. Algorithm analysis

We first show the feasibility of the solution by **Algorithm** 1 and then derive the approximation ratio of the proposed approximation algorithm as follows.

Lemma 1: The solution obtained by **Algorithm** 1 is a feasible solution to the delay-aware network slicing problem for a single multicast request r_k, assuming that the delay requirement d_k^{req} is larger than $\max\{\arg\max_{e \in E} \xi_e, \arg\max_{v \in V_{CL}} \xi_v\} \cdot c(T)$, where T is a multicast tree that implements request r_k.

Proof To show the feasibility of the solution, we need to show that (1) a shortest path from node s_k to node D_k in auxiliary graph G' corresponds to a multicast tree in the original network G, and within the multicast tree the traffic of r_k will be processed by all the VNFs in its network slice NS_k before being forwarded to its destinations in D_k; and (2) the delay requirement d_k^{req} is met.

We first show there always exists a delay constraint shortest path in G'. This is due to the fact we consider the scenarios that d_k^{req} is larger than $\max\{\arg\max_{e \in E} \xi_e, \arg\max_{v \in V_{CL}} \xi_v\} \cdot c(T)$. This means that for each edge $e \in E$ and each cloudlet $v \in V_{CL}$, we adopt the lowest tolerance level of delay requirement violation by setting $\xi_e = \arg\max_{e \in E} \xi_e$ and $\xi_v = \arg\max_{v \in V_{CL}}$. This is realistic in real scenarios, because network service providers can specify their targeted delay requirements for users in the network deployment stage. Users usually select a level of end-to-end delay requirements from the ones offered by the network service providers.

We then show the shortest path p' from s_k to D_k in G' corresponds to a multicast tree in G that forwards its data to VNFs in NS_k for processing before transferring the data to its destinations. It is clear that the source node s_k of r_k is connected to the virtual cloudlets $v'_{k,1}$ for each

$v \in V_{CL}$ and the first VNF $f_1 \in NS_k$. Also, starting from such virtual cloudlets the traffic can be forwarded to virtual cloudlets that represent other cloudlets. However, the sequence of traversed VNFs strictly follows the sequence in NS_k, because there is an edge between $v''_{k,l}$ to $v'_{k,l+1}$ for all l with $1 \leq l \leq L_k$. Finally, after being processed by the final VNF f_{L_k} in NS_k, i.e., the path includes some nodes v''_{k,L_k}, the processed data will be transferred to the destinations in D_k through the Steiner tree that is represented by edge $\langle v''_{k,L_k}, D_k \rangle$ in G'.

Since the found delay-constraint shortest path p' has a delay that is no greater than d_k^{req} and path p' represents the NFV-enabled multicasting of data of r_k to its destinations, it is clear that the delay requirement of request r_k is met by the multicast tree T_k derived from p'.

Theorem 1: Given a network $G = (V, E)$, a set V_{CL} of switches that are attached with cloudlets, a multicast request $r_k = (s_k, D_k; b_k, NS_k)$ that needs to transfer an amount b_k of data from its source s_k to its destinations in D_k within delay requirement of d_k^{req}, its network slice NS_k that guarantees the traffic being processed by the sequence of VNFs in NS_k before being forwarded to its destinations, there is an approximation algorithm for the delay-aware NFV-enabled multicasting problem without computing resource capacity, i.e. **Algorithm** 1, which delivers an approximate solution with an approximation ratio of $1+\epsilon$ in $O((L_k)^3 \cdot (V_{CL})^2 \cdot (\log\log L_k \cdot V_{CL} + 1/\epsilon))$ time, where ϵ is an accuracy parameter in the algorithm for delay-constraint shortest path problem [25].

Proof According to Lemma 1, the solution obtained by **Algorithm** 1 is a feasible solution. In the following, we only need to show the approximation ratio and the running time of the algorithm.

We first show the approximation ratio of the proposed approximation algorithm, which is to show that the accumulative cost of the derived multicast tree T_k is no more than $1+\epsilon$ times of the optimal cost OPT. Let OPT' be the optimal solution to the delay-constraint shortest path in auxiliary graph G'. Denote by c the approximate solution obtained by **Algorithm** 1. Clearly, we have

$$c \leq (1+\epsilon)OPT', \tag{24}$$

due to the result in [25]. To show the approximation ratio of the proposed algorithm, we need the relation between OPT and OPT'. To this end, we show that the optimal solution to the delay constraint shortest path problem in G' cannot be improved to a better solution to the delay-aware network slicing problem for a single multicast request in the original network G. We divide the implementation cost of request r_k due to **Algorithm** 1 into two parts: (1) the cost incurred by transferring the data of r_k from its source s_k to the final VNF f_{L_k} in NS_k and the cost due to the processing in VNFs, let p_{s_k,L_k} be such a path; and (2) the cost due to multicasting the processed data from the location for the final VNF f_{L_k} in NS_k to its destinations in D_k. For (1), it can be seen that the replacement of any edge in path p_{s_k,L_k} by an alternative edge in E will increase the total cost of p_{s_k,L_k}, since each edge in the auxiliary graph either represents a shortest path in the original network or data transfer among VNFs in the same cloudlet. For (2), since we do not consider the delay requirement in the finding of the Steiner tree to transfer the processed data to the destinations of multicast request r_k, the replacement of any edge in the Steiner tree will also increase the cost of the tree. Therefore, we have $OPT' = OPT$.

As we use the algorithm due to the algorithm in [25] to find a delay-constraint shortest path in G', the approximation solution has the following approximation ratio shown in inequality 24. Also, since $OPT = OPT'$, we

have

$$c \leq (1+\epsilon)OPT' = (1+\epsilon) \cdot OPT, \qquad (25)$$

which means that the approximation ratio (i.e., $\frac{c}{OPT}$) of **Algorithm** 1 is $1+\epsilon$.

We finally show the running time of the proposed approximation algorithm. It can be seen that the most time consuming part of the algorithm is the finding of delay constraint shortest path in auxiliary graph, which takes $O(m \cdot n(\log\log n + 1/\epsilon))$ time, where m and n denote the number of edges and nodes in G'. From the construction of the auxiliary graph, we can see that there are $O(L_k \cdot V_{CL})$ nodes and $O((L_k)^2 \cdot |V_{CL}|)$ edges. This means the running time of **Algorithm** 1 is $O((L_k)^3 \cdot (V_{CL})^2 \cdot (\log\log L_k \cdot V_{CL} + 1/\epsilon))$.

VI. An Efficient Heuristic for the Delay-Aware Network Slicing Problem

We here consider the delay-aware network slicing problem, by admitting as many as requests in a given set R of multicast requests while minimizing the implementation cost of the admitted requests, subject to the capacity constraints of the cloudlets and the delay requirements of admitted multicast requests.

A. Algorithm

The basic idea of the proposed solution is to propose a flexible model to characterize the priority of admitting a request, such that the system throughput is maximized. Intuitively, to maximize throughput, we usually favor the requests with small resource demands. That is, the multicast requests that transfer less data and require less VNFs in its service functions. In addition, the requests with larger delay requirements usually can be admitted more easily, as there are more choices to select cloudlets with enough computing resources. Therefore, we use the following priority model to capture the priority $Pri(r_k)$ of admitting a multicast request,

$$Pri(r_k) = \frac{1}{b_k \cdot \sum_{f_l \in NS_k} C_v(f_l)} + \lambda \cdot d_k^{req}, \qquad (26)$$

where λ is a tuning parameter that denotes the importance of the impact of delay requirements on the priority of the requests. This model means that the multicast with a less computing resource demand and a higher delay will have a higher priority to be considered for admitting.

Given the priorities of all multicast requests in R, we rank the requests into a decreasing order in terms of their priorities, and then admit the multicast requests one-by-one by an algorithm that is a slightly modified version of **Algorithm** 1. Specifically, some cloudlets in V_{CL} may not have enough computing resource to implement the VNFs in NS_k of the current considered multicast request r_k. We thus prune the network G by excluding such cloudlets and their incident links in E. Notice that the VNFs of NS_k may be placed to multiple cloudlets. We remove the cloudlets that do not have enough computing resource to implement the VNF that has the minimum resource demand, i.e., $\arg\min_{f_l \in NS_k} C_v(f_l)$. Based on the pruned network, we then invoke **Algorithm** 1 to find the multicast tree of multicast request r_k. The procedure continues until no more multicast requests can be admitted. The detailed steps of the proposed algorithm are illustrated in **Algorithm** 2, which is referred to as algorithm `Heu_Multicast`.

B. Discussion on considering other slicing criteria

Network slicing is proposed to allow the network being sliced according to multiple criteria. In this paper we consider the slicing of networks according to the delay requirements of users. In particular, in the delay-oriented network slicing problem with levels of delay requirements, we slice the network according to different levels of delay requirements. Other criteria of slicing networks may be

Algorithm 2 `Heu_Multicast`

Input: $G=(V,E)$, V_{CL}, C_e for each $e \in E$, C_v for each $v \in V_{CL}$, and a set of multicast requests with each multicast request being denoted by $r_k=(s_k, D_k; b_k, NS_k)$.
Output: The number of admitted multicast requests in R.

1: Rank the multicast requests in R according to their total computing resource demand and delay requirements, i.e., into a decreasing order of Eq. (26);
2: $Num\_Admitted \leftarrow 0$;
3: **for** each multicast request r_k in the ranked sequence **do**
4: Prune network G, by removing the cloudlets that do not have enough computing resource to implement the VNF that has the minimum resource demand, i.e., $\arg\min_{f_l \in NS_k} C_v(f_l)$, and their incident links;
5: Invoke **Algorithm** 1 to find a multicast tree for T_k;
6: **if** $T_k = \emptyset$ **then**
7: Reject multicast request r_k;
8: Continue;
9: $Num\_Admitted \leftarrow Num\_Admitted + 1$;
10: **return** $Num\_Admitted \leftarrow Num\_Admitted + 1$;

the types of services, security levels, quality of services, and etc. It must be mentioned that any slicing criteria for multicasting is basically the implementing of the VNFs of each multicast request in cloudlets that meet the criteria. Our solution can be easily extended to consider the security and service type criteria of network slicing. Specifically, assuming that each cloudlet has a level of security guarantee, we can extend the proposed solutions by adding a new constraint of when to select a cloudlet for each VNF (in building the auxiliary graph of algorithm `Appro_Multicast`).

C. Algorithm analysis

We now show the feasibility and performance of the proposed **Algorithm** 2 in the following theorem.

Theorem 2: Given a network $G=(V,E)$, a set V_{CL} of switches that are attached with cloudlets, the computing resource capacities of the cloudlets in V_{CL}, a set of multicast requests with each multicast request $r_k=(s_k, D_k; b_k, NS_k)$ requiring to transfer an amount b_k of data from its source s_k to its destinations in D_k within delay requirement of d_k^{req}, its network slice requirement NS_k that guarantees the traffic being processed by the sequence of VNFs in NS_k before being forwarded to its destinations, there is an approximation algorithm for the delay-aware network slicing problem for a single multicast request, i.e. **Algorithm** 1, which delivers a feasible solution to the problem in time $O(|R| \cdot (L_k)^3 \cdot (V_{CL})^2 \cdot (\log\log L_k \cdot V_{CL} + 1/\epsilon))$, where ϵ is an accuracy parameter in the algorithm for the delay-constraint shortest path problem [25].

Proof To show the feasibility of the solution obtained by **Algorithm** 2, we need to show that the computing capacity is not violated and the delay requirement of each admitted request is met. Obviously, no computing resource capacity is violated since we have pruned the network G before invoking **Algorithm** 1 for each request, by deleting the cloudlets that cannot meet the minimum computing resource demand of the VNFs in the network slice of a request. In addition, the delay requirement is guaranteed by **Algorithm** 1, as shown in Theorem 1.

For the running time of the heuristic algorithm, the ranking takes $O(|R|)$ time. Since the admission of each request invokes **Algorithm** 1, the admission of all requests in R takes $O(|R| \cdot (L_k)^3 \cdot (V_{CL})^2 \cdot (\log\log L_k \cdot V_{CL} + 1/\epsilon))$ time. The theorem holds.

VII. A Learning-Based Algorithm for the Delay-Oriented Network Slicing Problem

We now investigate the problem of delay-oriented network slicing problem with levels of delay requirements. Given a set of multicast slices with different levels of delay guarantees, we answer the question of how many each type of multicast slices should be provided in the future to meet user demands.

A. An optimization framework

The IoT service provider of the edge network G needs to strategically create network slices to admit a maximal number of user requests in future. Notice

that user requests are allowed to select their preferable multicast slices created by the IoT service provider. Their decisions have a vital role in deciding the number of to-be-created multicast slices in the network G. The IoT service provider of G may not know how the users make such decisions. How to jointly predict user decisions and optimize the placement of network slices thus is the primary focus of the IoT service provider.

To tackle the afore-mentioned challenge of an IoT service provider, we design an optimization framework that combines Reinforcement Learning (RL) and combinatorial optimization methods. Namely, we assume that there is an agent serving as a coordinator between user requests and the IoT service provider. The agent learns the interaction between user requests and the IoT service provider, by suggesting how many instances of multicast slices of each level of delay guarantees to create in the next time slot. The IoT service provider then adopts the suggestion of the agent and invokes algorithm `Heu_Multicast` to create the multicast slices. Fig. 3 shows an example of the proposed optimization framework.

As there is no enough data to train a useful deep learning model in IoT edge computing, Reinforcement Learning becomes a widely used online learning category for VNF allocation in edge computing. Sarsa algorithm is a representative learning method based on Reinforcement Learning [14], [44]. By using the Sarsa algorithm, we can find an acceptable prediction of the number of multicast slices that should be created in a short time, while minimizing the computing cost $c(v)$ on the cloudlet meanwhile meeting the delay guarantee d_k^{req} of the users requests.

B. The Reinforcement Learning procedure

We now describe the details of proposed algorithm based on a Reinforcement Learning (RL) process.

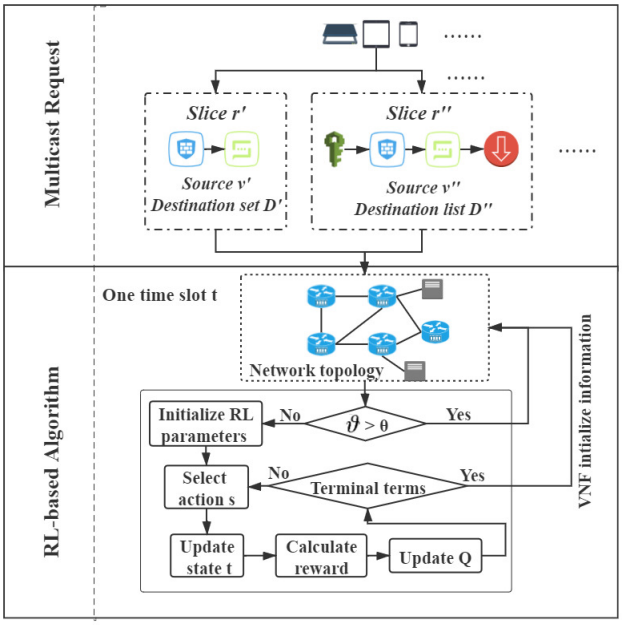


Fig. 3. The proposed RL-based optimization framework.

In each time slot t, the total number of requests is R^t. Before elaborating on the algorithm, we first define the request admit rate ϑ as

$$\vartheta = \frac{\sum_{d_k^t \le d_k^{req}, r_k^t \in R^t} r_k^t}{\sum_{r_k^t \in R^t} r_k^t}, \tag{27}$$

where r_k^t is a request of R^t in time slot t and $d_k^t \le d_k^{req}$ means that the delay requirement of multicast request r_k^t is met. Also we define a constant θ as the acceptable request admit rate of the system. In time slot t, if the request admit rate $\vartheta \ge \theta$, we do nothing but proceed to the time slot $t+1$; otherwise, we run RL procedure to decide whether to initialize new multicast slices in network, or shutdown all the multicast slices and restart the initialize process on which condition there are not enough resources for new slices. Specifically, at the beginning of each time slot t, the agent of the network service provider observes the state s_t of the system, and it is asked to choose an action a_t according to the Q-table. Following the action, the state of the environment transitions its state from s_t to s_{t+1} and the agent receives

a reward $rwd(s_t)$. According to reward $rwd(s_t)$, we update Q-table. Here we define the details.

- **State space:** The state of the system consists of currently admitted users' requests and the computing cost of the multicast slices.
- **Action space:** The agent needs to decide whether to increase or decrease the number of multicast slices of each level of delay guarantees. Thus, the action taken for the agent can be modeled as $\{-1, 0, 1\}_o$, where -1 means that the agent wishes to increase the number of multicast slices with oth level delay guarantee, 0 indicates that the agent wants to maintain the current number, and 1 implies that the agent wants to decrease the number. We assume that the number is increased or decreased by a fixed percentage.
- **Reward:** The reward is defined in Table II. As shown in the table, we divide the reward into four levels: (1) the decrease of computing cost and the increase of admit request rate, which is the best case we expect to see. We set the reward of this case as 2; (2) When the computing cost increases while the request admit rate increases, we set this reward to 1; (3) If the computing cost decreases while the admit rate drops, we set the reward to -1; and (4) the case that computing cost increases and request admit rate drops is the least case we expect, we thus set the reward of this case to -2.

TABLE II
REWARD

ϑ / Computing cost	Increase	Decrease
Decrease	-2	-1
Increase	1	2

And we update the Q-value by

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[rwd(s_{t+1}) + \gamma Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (28)$$

where γ is attenuation value, α is the learning rate, $Q(s_t, a_t)$ is the Q-value of the RL algorithm with state s_t and action a_t, and $rwd(s_t)$ is the reward of RL algorithm under state s_t of the system.

- **Objective:** Recall that the objective of the delay-oriented network slicing problem with levels of delay guarantees is to maximize the accumulated number of user requests that can be admitted by the IoT edge network G. The objective of the RL procedure thus adopts the same objective.

The detailed steps of the proposed algorithm is shown in **Algorithm** 3, which is referred to as `Learning_Multicast` for simplicity.

Algorithm 3 `Learning_Multicast`

Input: A set of multicast requests $r_k^{t-1} = (s_k^{t-1}, D_k^{t-1}; b_k^{t-1}, NS_k^{t-1})$ and its experienced delay d_k^{t-1} in time slot $t-1$.
Output: The new initialized multicast slices in each time slot.

1: **for** $t \leftarrow 1 \cdots t$ **do**
2: Calculate request admit rate ϑ. If $\vartheta \geq \theta$, start next time slot $t+1$, otherwise turn to next step;
3: Run the algorithm 1 to get new multicast slices and run the algorithm 2 with the requests in the previous time slot.;
4: Update $Q(s_t, a_t)$;
5: If the $Q(s_t, a_t)$ never changes in the previous m iterators or the Q(s_t,a_t) comes to zero, shutdown all multicast slices and restart; otherwise turn to next step;
6: Calculate the computing cost and delay experienced by the new multicast slices with the requests of previous time slot. If the new request admit rate $\vartheta' \geq \theta$, start next time slot $t+1$ to run the algorithm 2, otherwise turn to step 3;

VIII. SIMULATIONS

In this section we evaluate the performance of the proposed algorithms through experimental simulation.

A. Environment settings

We consider an edge network consisting of from 50 to 250 nodes, where each network is generated using GT-ITM [11]. The number of servers in each network is set to 10% of the network size, and they are randomly co-located with switches in the network. We also use real network topologies, i.e., GÉANT [10] and an ISP

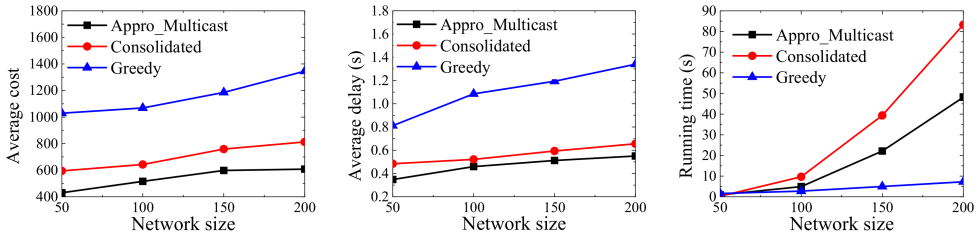
network from [46]. There are nine cloudlets for the GÉANT topology as set in [13] and the number of cloudlets in the ISP networks are provided by [38]. The computing capacity of each cloudlet varies from 40,000 to 120,000 MHz [15] (cloudlets with around tens of servers). Five types of network functions, i.e., Firewall, Proxy, NAT, IDS, and Load Balancing, are considered, and their computing demands are adopted from [13], [28]. The source and destination nodes of each multicast request is randomly generated, *the ratio* of the maximum number D_{max} of destinations of a multicast request to the network size $|V|$ is randomly drawn in the range of $[0.05, 0.2]$. The data of each request is randomly drawn from $[10, 200]$ Megabyte, and the delay requirement of transferring such data is randomly generated from $[0.05, 5]$ $seconds$. Notice that the transfer of larger amount of data can be divided into smaller amounts and transferred by multiple multicast requests. The running time of each algorithm is obtained based on a machine with a 3.70GHz Intel i7 Hexa-core CPU and 16 GiB RAM. Unless otherwise specified, these parameters will be adopted in the default setting.

Benchmark algorithm: Since this study is the very first to study the delay-aware NFV-enabled multicasting problem in a cloud network by assuming that the VNFs in each network slice can be placed into multiple cloudlets, there is no existing algorithms that deal with the exact same problem. We however use the following benchmark algorithms to investigate the performance of the proposed algorithms.

- We first compare the performance of the proposed approximation and heuristic algorithms with the algorithm in [50], [51] that consolidates all VNFs in each network slice into a single cloudlet. For simplicity, the algorithm is referred to as algorithm **Consolidated**
- We also compare the performance of the proposed approximation and heuristic algorithms with a greedy approach. The algorithm greedily selects the locations for each VNF in network slice NS_k of each multicast request r_k. Specifically, the algorithm finds the cloudlet that is closest to source node s_k, and then packs as many VNFs in NS_k to the cloudlet until no computing resource available. If there are still VNFs in NS_k that are not assigned, we find the next cloudlet that is the closest to the found cloudlets. After all VNFs in NS_k have been placed, the greedy approach finds the Steiner tree that connects the location for the final VNF in NS_k and the destinations in D_k. For the sake of simplicity, we denote this greedy algorithm as algorithm `Greedy`.

B. Performance evaluation

We first investigate the performance of algorithms `Appro_Multicast`, `Consolidated`, and `Greedy` in terms of the average cost of implementing a multicast request, the average delay experienced by a multicast request, and the running time in different networks, by varying the network size from 50 to 200. The results are shown in Fig. 4. From Fig. 4 (a) and (b), it can be seen that algorithm `Appro_Multicast` admits each multicast request at the lowest cost and delay among the three algorithms. The rationale behind is that algorithm `Appro_Multicast` jointly finds the paths from source nodes to cloudlets and the Steiner tree from the cloudlet to destination nodes, via the construction of the auxiliary graph G'. Furthermore, algorithm `Appro_Multicast` allows the VNFs in each network slice to be assigned to multiple cloudlets, thereby realizing a fine-grained trade-off between VNF implementation cost and the transmission cost. In addition, algorithm `Appro_Multicast` takes a bit more time

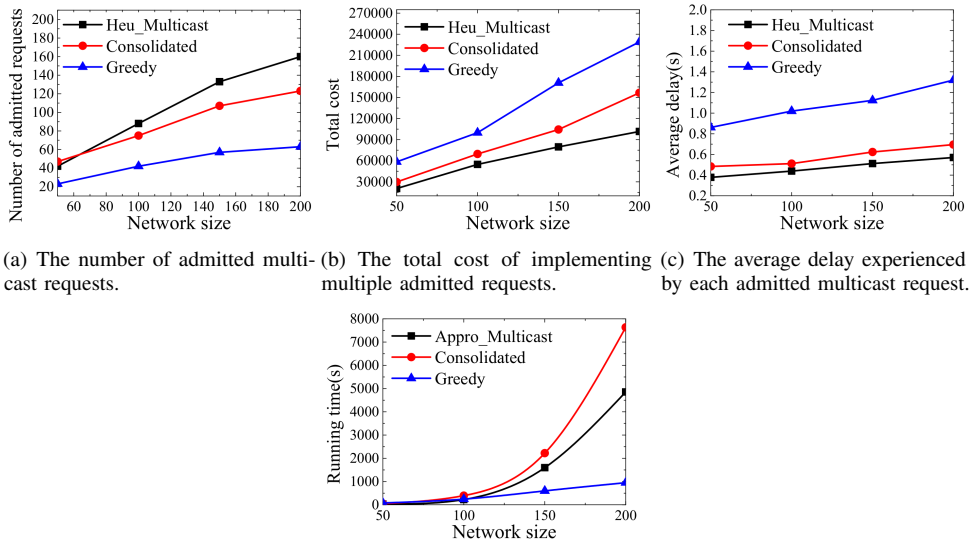


(a) Average cost of implementing a single multicast request.

(b) Average delay experienced by a single multicast request.

(c) The running time of each algorithm for processing a single multicast request.

Fig. 4. The performance of algorithms `Appro_Multicast`, `Consolidated` and `Greedy` in different synthetic networks with sizes varying from 50 to 200.



(a) The number of admitted multicast requests.

(b) The total cost of implementing multiple admitted requests.

(c) The average delay experienced by each admitted multicast request.

(d) The running time of each algorithm for processing multiple multicast requests.

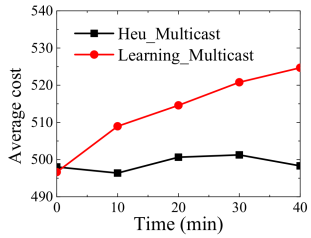
Fig. 5. The performance of algorithms `Heu_Multicast`, `Consolidated` and `Greedy` in different synthetic networks with sizes varying from 50 to 200.

to deliver a solution than that by algorithm `Greedy`. Notice that the exact solution due to **ILP1** is not scalable for large problem sizes, because the number of variables increases exponentially with the increase of problem sizes. We implemented **ILP1** by LP Solve [26], and it takes very long time to deliver an optimal solution for a network with 10 nodes. This makes the result meaningless, and we do not present the results of exact solutions in the rest of this paper.

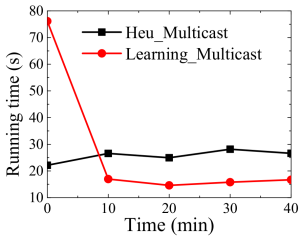
We then study the performance of the proposed heuristic algorithm `Heu_Multicast` in terms of the number of admitted requests, the total cost of implementing admitted requests, the average cost of implementing each admitted multicast request, and the running time, by varying the network size from 50 to 200. Fig. 5 shows the results. From Fig. 5 (a), we can see that the proposed heuristic outperforms algorithms `Consolidated` and `Greedy`. The reason is that algorithm `Heu_Multicast` explores a fine-grained trade-off between the delay requirement and the computing resource demand of each request. However, algorithm `Consolidated` assigns the VNFs in each network slice into a single cloudlet. This prevents some network slices being admitted by any cloudlet in the network, since there is no cloudlet with enough computing

resource for them. Similarly, algorithm Greedy may allow requests with high resource demands to occupy the computing resource in cloudlets, so other requests may not be able to be admitted due to lack of resource. In addition, algorithm Greedy greedily selects cloudlets that are close to the VNFs of each multicast request, without considering the impact of delay in the selection of cloudlet. This may lead to requests cannot be admitted due to the violation of their delay requirements. From Fig. 5 (b), we can also see algorithm Heu_Multicast achieves a higher implementation cost than that of algorithm Greedy, because algorithm Heu_Multicast admits more requests than algorithm Greedy. It can be seen in Fig. 5 (c), algorithm Heu_Multicast has a lower average cost of implementing each request than that by algorithm Greedy. Algorithm Heu_Multicast takes more time to deliver a solution to the problem than algorithm Greedy.

We finally evaluate the performance of algorithm Learning_Multicast against that of algorithm Heu_Multicast in network GÉANT in a finite time horizon of 100 minutes, by assuming that there are 200 multicast requests in each time slot and the percentage of new requests of each time slot is 20%. The admit rate ϑ is set to 95%. Note that algorithms Learning_Multicast and Heu_Multicast deal with different delay settings. To compare their performance, we assume that algorithm Heu_Multicast knows all delay requirement levels of the requests. Fig. 6 shows the evaluation results, from which we can see that algorithm Learning_Multicast has a slightly higher average cost than that of algorithm Heu_Multicast. The reason is that algorithm Learning_Multicast assumes the delay requirement of requests are not known, this may lead to the creation of some multicast slices that are not matched by any requests.



(a) Average cost of implementing each admitted multicast request.



(b) The running time of each algorithm for processing a multicast request.

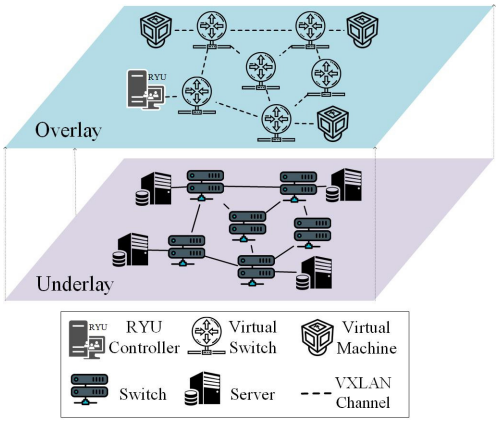
Fig. 6. The performance of algorithms Heu_Multicast and Learning_Multicast.

IX. Implementations in a Test-bed

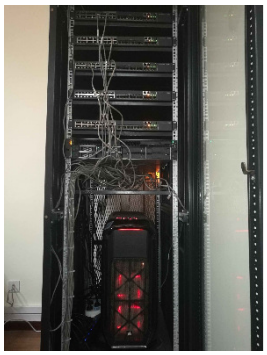
In this section, we evaluate the performance of the proposed algorithms on a real test-bed.

A. Testbed settings

We build a test-bed consisting of both an underlay with hardware switches and an overlay with virtual switches, as shown in Fig. 7. The physical underlay consists of five switches, i.e., Huawei S5720-32C-HI-24S-AC, H3C S5560-30S-EI, Ruijie RG-5750C-28Gt4XS-H, CISCO 3750X-24T, and Centec aSW1100-48T4X. It also has five servers with i7-8700 CPU and 16G RAM. We also use the Raspberry PI with 1.2GHz CPU and 1GB RAM [39] to represent the IoT devices that serve as the source node of each multicast slice. Netconf [7] and SNMP [5] protocols are used to manage the switches and the links that interconnect them. VXLAN [30] is used to virtualize an overlay network with a number of containers. We then virtualize hundreds of Open vSwitch (OVS) [35] nodes in the overlay network with real network functions by using Mininet. The Mininet is a network virtualization



(a) The underlay and overlay of the test-bed



(b) The physical deployment of the hardware switches

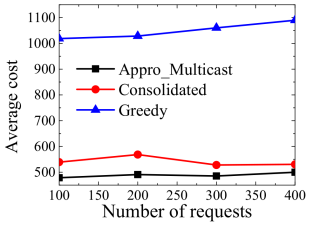
(c) The raspberry pi

Fig. 7. A test-bed with both hardware switches and virtual resources.

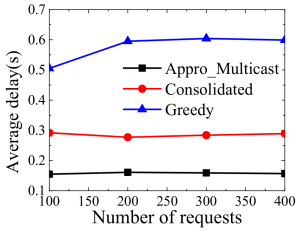
tool which creates a network of virtual hosts, switches, controllers, and links [31], [34].

The overlay network is built following the real topology AS1755 and GÉANT [10]. Its OVS nodes and VMs are controlled by a Ryu controller [47]. The proposed algorithms are implemented as Ryu applications. All the rest settings are the same as the simulations in the previous subsection.

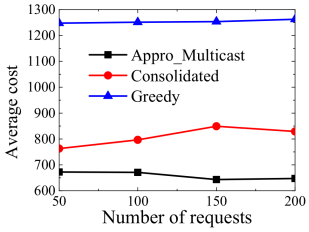
In the afore-mentioned test-bed, we now study the performance of algorithms `Appro_Multicast`, `Consolidated` and `Greedy` in real networks i.e., GÉANT and AS1755 [10], by varying the number of multicast requests from 50 to 1000. Fig. 8 shows the result, from which we can see that in both networks the average cost of implementing a multicast request by algorithm `Appro_Multicast` is much lower than that of algorithms `Consolidated` and `Greedy`. Also, the average delay experienced by each admitted multicast request obtained by algorithm `Appro_Multicast` is much lower than that by algorithms `Consolidated` and `Greedy`, in both of the real networks GÉANT[10] and AS1755 [21].



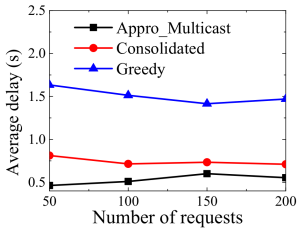
(a) Average cost of implementing each admitted multicast request in network GÉANT.



(b) Average delay experienced by multicast requests in network GÉANT



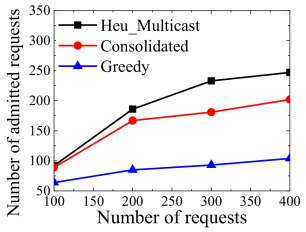
(c) Average cost of implementing each admitted multicast request in network AS1755.



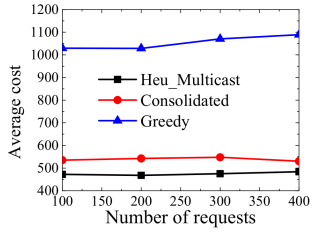
(d) Average delay experienced by multicast requests in network AS1755

Fig. 8. The performance of algorithms `Appro_Multicast`, `Consolidated` and `Greedy` in real networks GÉANT and AS1755.

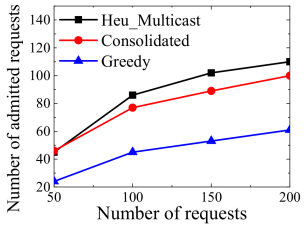
We then investigate the performance of algorithms `Heu_Multicast`, `Consolidated` and `Greedy` in networks GÉANT and AS1755, by varying the number



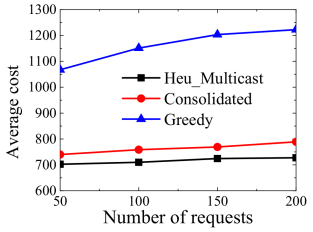
(a) The number of admitted multicast requests in network GÉANT.



(b) The average cost of implementing each admitted multicast request in network GÉANT



(c) The number of admitted multicast requests in network AS1755.

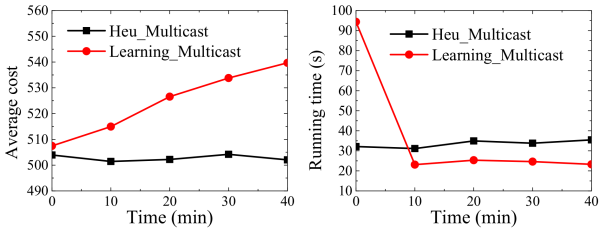


(d) The average cost of implementing each admitted request in network AS1755

Fig. 9. The performance of algorithms `Heu_Multicast`, `Consolidated` and `Greedy` in networks GÉANT and AS1755 in the test-bed.

of multicast requests from 100 to 400. The results on the testbed are shown is Fig. 9. It can been seen from Fig. 9(a) and Fig. 9(c), the number of request admitted by algorithm `Appro_Multicast` is the highest in both of the real networks GÉANT and AS1755 [10]. We can also see from Fig. 9(b) and Fig. 9(d), the average costs of implementing a multicast request by algorithm `Appro_Multicast` and `Consolidated` are much lower than that of algorithm `Greedy`, while the average cost does not too much differ of implementing a multicast by algorithm `Appro_Multicast` and `Consolidated`. The arguments are similar as those in Fig. 5.

We finally evaluate the performance of algorithms `Learning_Multicast` and `Heu_Multicast` in the testbed during a monitoring period of 40 minutes, by setting the length of each time slot as 10 minutes, the number of multicast requests in each time slot as 200, the percentage of new requests of each time slot as 20%, and the admit rate ϑ as 95%. The evaluation results are shown in Fig. 10. It can been seen from Fig. 10(a) that the average cost of algorithm `Learning_Multicast` is higher than that of algorithm `Heu_Multicast`. From Fig. 10 (b), we can see that algorithm `Learning_Multicast` stabilizes very quickly within 10 minutes, and consumes less time than that by algorithm `Heu_Multicast`. The reasons are similar as those in Fig. 6



(a) Average cost of implementing each admitted multicast request.

(b) Running time for each algorithm for processing a multicast request.

Fig. 10. The performance of algorithms `Heu_Multicast` and `Learning_Multicast`.

Notice that the advantage of adopting an overlay and underlay architecture in our test-bed is that it enables fast implementations of the proposed algorithms in some well-established controller frameworks. However, the average delays and running time are highly related to the physical network, which is the bottleneck in the test-bed. We thus consider the expansion of the test-bed to support faster

communications between the controller and switches as our future work.

X. Conclusion and Future Work

In this paper we studied the delay-aware network slicing problems with and without computing resource capacity constraint in an IoT edge network consisting of multiple cloudlets. We first proposed optimal exact solutions to the problems of the delay-aware network slicing with a single or multiple requests, by formulating the problems into ILPs. For the problem with a single multicast request, we then devised an approximation algorithm with an approximation ratio for the delay-aware NFV-enabled multicasting problem without computing resource constraint, subject to the delay requirement of each multicast request. Given multiple multicast requests, we then propose an efficient heuristic that aims to maximize the number of multicast requests that can be admitted by the network, considering that the computing resource at each cloudlet is limited and the delay requirement needs to be met. When users have different levels of delay requirements, we considered the problem of delay-oriented network slicing, for which we designed a learning-based algorithm based on reinforcement learning. We finally evaluated the performance of the proposed algorithms by experimental simulations and implementations in a real test-bed. Results demonstrate that the proposed algorithms outperform the other heuristics.

The future potential studies built upon this work include: (1) this paper considered the slicing of edge networks for multicast applications according to their delay requirements. There however are some other slicing metrics, such as security and service types. One future direction is to explore the network slicing algorithms with different slicing metrics and the non-trivial interplay among the metrics; (2) Another one is to explore the dynamic scaling in/out of existing multicast slices, considering the uncertainties of networks, such as uncertain delays of processing and transmission. The scaling of multicast slices in the current time slot impacts the admissions of future multicast requests significantly. We plan to design online learning algorithms for this problem.

Acknowledgements

We would like to thank the three anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped us improve the quality and presentation of the paper greatly. The work of Qiufen Xia, Zichuan Xu, and Guowei Wu is partially supported by the National Natural Science Foundation of China (Grant No. 61802047, 61802048, and 61872053), the fundamental research funds for the central universities in China (Grant No. DUT19RC(4)035 and DUT19GJ204), DUT-RU Co-Research Center of Advanced ICT for Active Life, and the "Xinghai Scholar Program" in Dalian University of Technology, China. The work by Pan Zhou is supported by the National Natural Science Foundation of China (Grant No. 61972448). The work by Jiankang Ren is supported by the National Science Foundation for Post-doctoral Scientists of China (Grant No. 2016M591431 and 2018T110221).

References

[1] S. M. Banik, S. Radhakrishnan, and C. N. Sekharan. Multicast routing with delay and delay variation constraints for collaborative applications on overlay networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No.3, pp. 421-431, IEEE, 2007.

[2] P. Caballero, A. Banchs, G. D. Veciana, and X. Costa-Pérez. Network slicing games: enabling customization in multi-tenant mobile networks. *IEEE/ACM Transactions on Networking*, Vol. 27, No. 2, pp. 662-675, IEEE, 2019.

[3] R. Cohen, L. Eytan, J. Naor, and D. Raz. On the effect of forwarding table size on SDN network utilization. *Proc. of INFOCOM*, IEEE, 2014.

[4] R. Cohen, L. Eytan, J. Naor, and D. Raz. Near optimal placement of virtual network functions. *Proc. of INFOCOM*, IEEE, 2015.

[5] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). http://www.hjp.at/doc/rfc/rfc1098.html, 1989.

[6] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra. Exploiting congestion games to achieve distributed service chaining in NFV networks. `IEEE Journal on Selected Areas in Communications`, Vol, 35, No. 2, pp. 407-420, IEEE, 2017.

[7] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network configuration protocol (NETCONF). http://www.hjp.at/doc/rfc/rfc6241.html, 2011.

[8] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. Network slicing in 5G: survey and challenges. *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 94-100, IEEE, 2017.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, NY, 1979.

[10] GÉANT. http://www.geant.net. Accessed in April. 2020.

[11] http://www.cc.gatech.edu/projects/gtitm/. Accessed in April. 2020.

[12] L. Guo, J. Pang, and A. Walid. Joint placement and routing of network function chains in data centers. *Proc. of INFOCOM*, IEEE, 2018.

[13] A. Gushchin, A. Walid, and A. Tang. Scalable routing in SDN-enabled networks with consolidated middleboxes. *Proc. of HotMiddlebox*, ACM, 2015.

[14] G. A. Rummery , and M. Niranjan. On-line Q-learning using connectionist systems. *Cambridge, England: University of Cambridge, Department of Engineering*, pp. 1-21, 1994.

[15] Hewlett-Packard Development Company. L.P. Servers for enterprise – bladeSystem, rack & tower and hyperscale. http://www8.hp.com/us/en/products/servers/, 2015.

[16] H. Huang, S. Guo, J. Wu, and J. Li. Service chaining for hybrid network function. *IEEE Transactions on Cloud Computing*, Vol. 7, No. 4, pp. 1082-1094, IEEE, 2019.

[17] H. Huang, P. Li, and S. Guo. Traffic scheduling for deep packet inspection in software-defined networks. *Concurrency and computation: practice and experience*, Vol. 29, No.16, pp. e3967, Wiley, 2017.

[18] L. Huang, H. Hung, C. Lin, and D. Yang. Scalable steiner tree for multicast communications in software-defined networking. *Computing Research Repository (CoRR)*, vol. abs/1404.3454, 2014.

[19] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo and Y. Xu. Dynamic routing for network throughput maximization in software-defined networks. *Proc. of INFOCOM*, IEEE, 2016.

[20] R Zagarella. Why multicast will be essential for industrial IoT. *https://www.nnnco.com.au/blog/article/why-multicast-will-be-essential-for-industrial-iot/*, NNN Australia, 2018.

[21] S. Knight et al. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, Vol. 29, No. 9, pp. 1765 - 1775, IEEE, 2011.

[22] T-W. Kuo, B-H. Liou, K. C. Lin, and M-J Tsai. Deploying chains of virtual network functions: on the relation between link and server usage. *Proc. of INFOCOM*, IEEE, 2016.

[23] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat. A resource allocation framework for network slicing. *Proc. of INFOCOM*, IEEE, 2018.

[24] Y. Li, L. T. X. Phan, and B. T. Loo. Network functions virtualization with soft real-time guarantees. *Proc. of INFOCOM*, IEEE, 2016.

[25] D. H. Lorenz and D. Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, Vol. 28, pp. 213-219, Elsevier, 2001.

[26] LP Solve. *http://lpsolve.sourceforge.net/5.5/*, accessed 12/2018.

[27] T. Lukovszki and S. Schmid. Online admission control and embedding of service chains. *Proc. of SIROCCO*, Springer, 2015.

[28] J. Martins *et al.* ClickOS and the art of network function virtualization. *Proc. of NSDI*, USENIX, 2014.

[29] L. Mamatas, S. Clayman, and A. Galis. Software-defined infrastructure. *IEEE Communications Magazine*, Vol. 53, No. 4, pp 166-174, IEEE, 2015.

[30] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. https://www.hjp.at/doc/rfc/rfc7348.html, RFC, 7348, 1-22.

[31] Mininet. http://mininet.org/. Accessed in Jan 2020.

[32] M. Mongioví, A. K. Singh, X. Yan, B. Zong, and K. Psounis. Efficient multicasting for delay tolerant networks using graph indexing. *Proc. of INFOCOM*, IEEE, 2012.

[33] H. Moens and F. D. Turck. VNF-P: A model for efficient placement of virtualized network functions. *Proc. of CNSM*, IEEE, 2014.

[34] OpenFlow. https://www.opennetworking.org. Accessed in Jan 2020.

[35] Open vSwitch. `https://www.openvswitch.org/`. Accessed in Jan 2020.

[36] M. Pan and S. Yang. A lightweight and distributed geographic multicast routing protocol for IoT applications. *Computer Networks*, Vol. 112, pp. 95-107, Elsevier, 2017.

[37] I. Petri, A. Zamani,D. Balouek-Thomert, O. Rana,Y. Rezgui, and M. Parashar. Ensemble-based network edge processing. *Proc. of UCC*, IEEE, 2018.

[38] Z. A. Qazi, C. C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. *Proc. of SIGCOMM*, ACM, 2013.

[39] Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[40] O. Rana, M. Shaikh, M. Ali, A. Anjum, and L. Bittencourt. Vertical workflows: Service orchestration across cloud & edge

resources. *Proc. of the 6th International Conference on Future Internet of Things and Cloud* (FiCloud), IEEE, 2018.

[41] B. Ren, D. Guo, G. Tang, X. Lin, and Y. Qin. Optimal service function tree embedding for NFV Enabled multicast. *Proc. of ICDCS*, IEEE, 2018.

[42] B. Ren, D. Guo, Y. Shen, G. Tang, and X. Lin. Embedding service function tree with minimum cost for NFV-enabled multicast. *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 5, pp. 1085-1097, 2019.

[43] H. Ren, Z. Xu, W. Liang, Q. Xia, P. Zhou, O. F. Rana, A. Galis, and G. Wu. Efficient algorithms for delay-aware NFV-enabled multicasting in mobile edge clouds with resource sharing. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 31, No.9, pp. 2050-2066, 2020.

[44] R. S. Sutton, and A. G. Barto. Reinforcement learning: An introduction. *MIT press*, 2018.

[45] H. Soni, W. Dabbous, T. Turletti, and H. Asaeda. NFV-based scalable guaranteed-bandwidth multicast service for software-defined ISP networks. *IEEE Transactions on Network and Service Management*, Vol.14, No. 5, pp. 1157-1170, 2017.

[46] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. *Proc. of SIGCOMM*, ACM, 2002.

[47] Ryu controller `https://osrg.github.io/ryu`.

[48] Z. Xu, W. Liang, and Q. Xia. Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands. *IEEE Transactions on Cloud Computing*, Vol.6, No. 3, pp. 694-707, IEEE, 2018.

[49] Z. Xu, W. Liang, A. Galis, and Y. Ma. Throughput maximization and resource optimization in NFV-enabled networks. *Proc. of ICC'17*, IEEE, 2017.

[50] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. Approximation and online algorithms for NFV-enabled multicasting in SDNs. *Proc. of ICDCS*, IEEE, 2017.

[51] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis. Efficient NFV-enabled multicasting in SDNs. *IEEE Transactions on Communications*, Vol. 63, No. 7, pp. 2052-2070, IEEE, 2019.

[52] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao. Task offloading with network function services in a mobile edge-cloud network. *IEEE Transactions on Mobile Computing*, Vol.18, No. 11, pp. 2672-2685, IEEE, 2019.

[53] Z. Xu, Y. Zhang, W. Liang, Q. Xia, O. Rana, A. Galis, G. Wu, and P. Zhou. NFV-enabled multicasting in mobile edge clouds with resource sharing. *Proc. of ICPP*, ACM, 2019.

[54] Y. Ma, W. Liang, J. Wu and Z. Xu. Throughput maximization of NFV-enabled multicasting in mobile edge cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol.31, No.2, pp. 393-407, IEEE, 2020.

[55] Y. Zhang, N. Beheshti, L. Beliveau, *et. al.* StEERING: A software-defined networking for inline service chaining. *Proc. of ICNP*, IEEE, 2013.

[56] S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. L. Garcia. Network function virtualization enabled multicast routing on SDN. *Proc. of ICC*, IEEE, 2015.

[57] S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. L. Garcia. Routing algorithms for network function virtualization enabled multicast topology on SDN. *IEEE Transaction on Network and Service Management*, Vol.12, No.4, pp.580–594, IEEE, 2015.

Yugen Qin is working toward the ME degree in the School of Software, Dalian University of Technology. His research interests include mobile edge computing and algorithmic game theory in SDN.

Qiufen Xia received her PhD degree from the Australian National University in 2017, the ME degree and BSc degree from Dalian University of Technology in China in 2012 and 2009, all in Computer Science. She is currently a lecturer at the Dalian University of Technology. Her research interests include mobile edge computing, query evaluation, big data analytics, big data management in distributed clouds, and cloud computing.

Zichuan Xu (M'17) received his PhD degree from the Australian National University in 2016, ME degree and BSc degree from Dalian University of Technology in China in 2011 and 2008, all in Computer Science. He is currently a Research Associate at Department of Electronic and Electrical Engineering, University College London, UK. His research interests include mobile edge computing, software-defined networking, wireless sensor networks, routing protocol design for wireless networks, algorithmic game theory, and optimization problems.

Pan Zhou is currently an associate professor with the Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China. He received his Ph.D. in the School of Electrical and Computer Engineering at the Georgia Institute of Technology in 2011, Atlanta, USA. He received his B.S. degree in the Advanced Class and M.S. from school of EIC in HUST in 2006. He was a senior technical member at Oracle Inc. America during 2011 to 2013. His research interest includes: network security, machine learning and big data analysis, information networks.

Alex Galis is a Professor in Networked and Service Systems at University College London. He has co-authored 10 research books and more that 250 publications in the Future Internet areas: system management, networks and services, networking clouds, 5G virtualisation and programmability. He was a member of the Steering Group of the Future Internet Assembly (FIA) and he led the Management and Service-aware Networking Architecture (MANA) working group. He acted as TPC chair of 14 IEEE conferences. He is also a co-editor of the IEEE Communications Magazine feature topic on Advances In Networking Software. He acted as a Vice Chair of the ITU-T SG13 Group on Future Networking. He is involved in IETF and ITU-T SG13 network slicing activities and he is also involved in IEEE SDN initiative.

Omer F. Rana received the B.S. degree in information systems engineering from the Imperial College of Science, Technology and Medicine, London, U.K., the M.S. degree in microelectronics systems design from the University of Southampton, Southampton, U.K., and the Ph.D. degree in neural computing and parallel architectures from the Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering with Cardiff University, Cardiff, U.K. His current research interests include problem solving environments for computational science and commercial computing, data analysis and management for large-scale computing, and scalability in high performance agent systems.

Jiankang Ren received the B.Sc., M.E., and Ph.D. degrees in Computer Science from Dalian University of Technology in China, in 2008, 2011, and 2015, respectively. He was a Visiting Scholar with the Computer and Information Science Department, University of Pennsylvania, USA, from September 2013 to September 2014. He is currently an Associate Professor with the School of Computer Science and Technology at Dalian University of Technology. His research interests include cyber-physical systems (CPS), cloud computing, and computational intelligence.

Guowei Wu received his Ph.D degree from Harbin Engineering University in 2003, China. He is now a professor at the School of Software, Dalian University of Technology (DUT) in China. His research interests include embedded real-time system, cyber-physical systems (CPS), and smart edge computing. He has published over 100 journal and conference papers.