Seamless Support of Low Latency Mobile Applications with NFV-Enabled Mobile Edge-Cloud

Binxu Yang, Wei Koong Chai and George Pavlou
University College London, United Kingdom
Email: {binxu.yang.13, w.chai, g.pavlou}@ucl.ac.uk

Konstantinos V. Katsaros
Intracom Telecom, Greece
Email: konkat@intracom-telecom.com

***Abstract*—Emerging mobile multimedia applications, such as augmented reality, have stringent latency requirements and high computational cost. To address this, mobile edge-cloud (MEC) has been proposed as an approach to bring resources closer to users. Recently, in contrast to conventional fixed cloud locations, the advent of network function virtualization (NFV) has, with some added cost due to the necessary decentralization, enhanced MEC with new flexibility in placing MEC services to any nodes capable of virtualizing their resources. In this work, we address the question on *how* to optimally place resources among NFV-enabled nodes to support mobile multimedia applications with low latency requirement and *when* to adapt the current resource placements to address workload changes. We first show that the placement optimization problem is NP-hard and propose an online dynamic resource allocation scheme that consists of an adaptive greedy heuristic algorithm and a detection mechanism to identify the time when the system will no longer be able to satisfy the applications' delay requirement. Our scheme takes into account the effect of current existing techniques (i.e., auto-scaling and load balancing). We design and implement a realistic NFV-enabled MEC simulated framework and show through extensive simulations that our proposal always manages to allocate sufficient resources on time to guarantee continuous satisfaction of the application latency requirements under changing workload while incurring up to 40% less cost in comparison to existing overprovisioning approaches.**

I. Introduction

Over the last decade, advances in wireless access technologies (e.g., WiFi and LTE) have enabled an explosion of resource-hungry mobile applications, challenging current mobile devices' processing ability. In particular, mobile multimedia applications with low latency requirements (of the order of hundreds of milliseconds (ms) [1]), such as video streaming, gaming, augmented reality and face recognition, are computationally expensive for today's mobile devices resulting in fast exhaustion of battery life and long processing delays [2]. Mobile edge-cloud (MEC) [3] approaches (e.g., *cloudlet* [4], Telco cloud [5], *follow-me cloud* [6]) were initially devised to address the aforementioned issue [2], [4]. Micro-clouds are installed at fixed locations such as access points (APs) to which mobile users offload computationally expensive tasks to leverage additional resources from virtual machines (VMs). However, such solutions require deployment of micro-clouds in a large number of fixed locations in order to achieve low delay and incur significant operational costs [2]. By limiting the number and capacity of micro-clouds to save costs in turn sacrifices the performance of such solutions (e.g., long latency). Thus, there is a tradeoff between cost efficiency and service quality (e.g., response time).

Recently, with the advent of network function virtualization (NFV) [7] and software defined networking (SDN) [8], the concept of NFV-enabled MEC emerged [8] whereby services can be hosted at any conventional network node that has virtualized resources (e.g., APs, routers, etc.). NFV was first proposed to facilitate network function deployment for Internet Service Providers (ISPs). It decouples network functions from the underlying hardware by leveraging virtual resources provided by commodity servers. Here, we consider an NFV use case for supporting both network functions and MEC services. Using NFV to support MEC services allows ISPs to rent their network infrastructure in the form of VMs to application service providers (ASPs). At the same time, SDN facilitates network configurations by decoupling the control and data planes. The combination of NFV and SDN enables flexible service-hosting node deployment (i.e., VMs instantiation). This allows ISPs to flexibly instantiate and shutdown service-hosting node strategically based on user demands, thereby improving cost efficiency.

However, such an NFV-enabled MEC still needs to address the aforementioned cost *vs.* performance tradeoff whereby the number of NFV-enabled nodes serving as service-hosting nodes should be kept low while service disruption due to service elasticity is minimized. Current mitigation techniques, such as auto-scaling and load balancing (ALB) [9], [10], could cope with service elasticity only to the extent before the capacity limit of service-hosting nodes is reached. Subsequently, new service-hosting node locations are required to provide more physical capacity. These locations need to be carefully derived such that the distance between users and resources is small and the physical capacity is enough to avoid long network access and queueing delays at VMs respectively. Therefore, it is challenging to dynamically place service-hosting nodes among NFV-enabled nodes to simultaneously achieve both cost efficiency and low latency over time.

The placement of service-hosting nodes in MEC environments has been investigated as an offline static network planning problem[1] on how to optimally place a fixed number of micro-clouds in the network to minimize the network access

[1]We refer to such an offline formulation as *static* placement problem hereafter.

latency [11], [12] assuming known / predicted unchanged load. On the other hand, work on the resource allocation in an online MEC system has focused on the dynamic routing of user requests to fixed clouds [11], [12], [13]. Locations of physical micro-cloud hardware are first fixed (e.g., after solving the static placement problem), and the dynamic problem studied is the mapping of user requests to these predetermined locations where VMs are hosted. These prior works did not consider the flexibility afforded by the NFV-enabled MEC where service-hosting node locations can be changed over time.

In this work, we study the problem of online dynamic placement of service-hosting nodes in order to minimize ISPs' operational costs while satisfying the service-level response time requirements. We take a longitudinal view and investigate not only *how* service-hosting nodes should be instantiated but also *when* this should happen while explicitly taking into account the added help provided by ALB. We first introduce an SDN based NFV-enabled MEC model and formulate our dynamic problem by adopting integer programming in Section II. In Section III, we first show that the problem formulated is NP-hard and then, we detail our solution that consists of (1) a capacity violation detection (CVD) mechanism to estimate the time when ALB cannot cope with service elasticity and (2) an online adaptive greedy (OAG) heuristic algorithm to dynamically choose the locations of service-hosting nodes and associated network paths based on the most current service-hosting node deployment. In Section IV, using real mobility traces [14] and a three-level metropolitan scale cellular network [11], we present our evaluation results obtained from a packet-level simulator. We show that our approach satisfies the service-level response time requirement while achieving a cost saving of up to 40% in comparison to current practices.

II. System Model and Problem Formulation

A. System Model

We consider stateless mobile multimedia applications (e.g., face recognition, augmented reality, etc.) to be pre-installed into VMs in service-hosting nodes. Mobile users send raw files (e.g., picture frames) to VMs allocated in service-hosting nodes for processing rather than executing application instances locally in their mobile devices. These application instances are hosted at VMs that are instantiated beforehand at the network planning stage and VMs' sizes are derived according to the expected user demand for the service.

We define "service-hosting node" to be any NFV-enabled network nodes that has virtualized resources. We depict in Fig. 1 the three-level hierarchical wireless metropolitan area network considered in this work . Let $G = (V, E, P)$ denote the network, where V is the set of NFV-enabled nodes, E is the set of links and P is the set of paths between pair of nodes in V. The three-level network consists of APs (denoted by $b \in B$ and $B \subset V$), aggregation nodes and metropolitan level core network nodes. Each node v has limited virtual resources, k_v (e.g., CPU, memory) and could serve as service-hosting nodes by allocating resources to VMs. We assume all

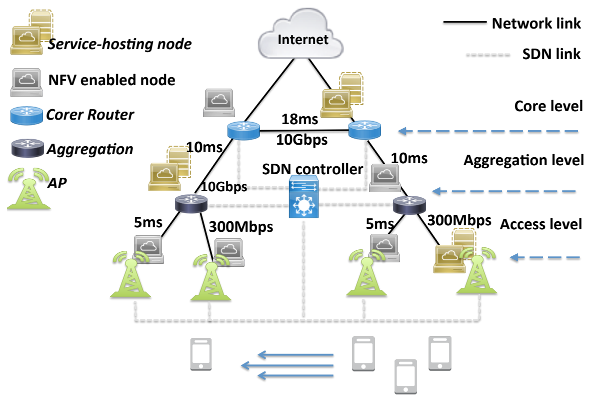


Fig. 1: Three-level Hierarchical MEC System Model.

nodes have the same capacity, k_v. Each AP is connected to a single aggregation node and all user requests from an AP are served by a single NFV-enabled service-hosting node through the same path, $p_{bv} \in P_{bv}$, between AP b and v (v is selected to host service). If a handover occurs, the involved user's requests will be routed to the new service-hosting node without service migration since we consider stateless applications. Further, we consider discrete time, $t \in T$. The total load incurred by mobile users at AP, b, at time slot, t, is denoted by A_b^t.

The network configuration of service-hosting nodes and NFV-enabled nodes is conducted by a centralized SDN controller that has a global view of resource utilization in all nodes and network links. It monitors the system and applies configurations (e.g., service deployment, network path set-up depicted as dash lines on Fig. 1).

B. Problem Statement

For the considered system model, mobile users move between APs and change request rate or workload over time. VMs in service-hosting nodes may be overloaded, leading to increased waiting time during request processing. However, for interactive low latency applications, it is crucial to respond to user requests promptly as users are sensitive to latency and may quit their applications. From an ASP's point of view, losing users due to long queueing delays along the network path and service-hosting node, leads to loss of revenue. This also results in the decrease of ISP revenue at the same time. Overprovisioning solves the problem of latency but it is obviously not cost-efficient for both ASPs and ISPs. In general, the operational cost of ISP is a function of the number of active servers, and cost savings can be achieved by cloud consolidation [15]. Alternatively, ALB can alleviate overloading due to minor changes to the overall workload but is always limited by the physical capacity of existing NFV-enabled nodes. Hence, allocation of new NFV-enabled nodes as service-hosting nodes will be needed in such cases to alleviate increased delays.

From an ISP's perspective, given time varying workload ($\sum_b^B A_b^t, \forall t$), the question is then on how to allocate VMs (e.g., service-hosting node) within its NFV-enabled infrastructure at different time instances for ASPs' services so that profit is maximized (i.e., satisfying user requests) while the operational cost is minimized. We use integer programming to formulate the problem with two binary decision variables, Y_v^t and $X_{p_{bv}}^t$, which represent respectively the location of service-hosting node (i.e., $Y_v^t = 1$ if at time t, v is chosen as the location of a service-hosting node) and the path between b and v (i.e., $X_{p_{bv}}^t = 1$ if p_{bv} is chosen). We use c_v to denote the operational cost for a chosen NFV-enabled node, v.

To satisfy the latency requirement, we decompose the service-level response time into the following:

1) *Network access time* – Denote by w_b^t the bandwidth consumption of flows departing from AP b. As long as w_b^t does not exceed any link capacity, BW_e, we can represent access delay and application delay requirement in terms of number of network hops. Then, to satisfy application's network access latency requirement, D, we simply need to constrain the number of hops between b and v, d_{bv}, to be below D, (i.e., $d_{bv} \leq D$).
2) *Service processing time* – We refer to the time unit a VM serves a request. We assume that as long as there is an available resource unit, the processing delay is bounded and can be represented by a mean expected value, μ.

The integer programming problem is formulated as below[2]:

$$\text{Min} \sum_{v \in V} c_v Y_v^t, \forall t \in T, \tag{1}$$

Subject to

$$\sum_{p_{bv} \in P_{bv}} \sum_{b \in B} A_b^t X_{p_{bv}}^t - k_v Y_v^t \leq 0, \forall v \in V, \forall t \in T, \tag{2}$$

$$\sum_{b \in B} \sum_{v \in V} \sum_{p_{bv}(e) \in P_{bv}(e)} w_b^t X_{p_{bv}(e)}^t \leq BW_e, \forall e \in E, \forall t \in T, \tag{3}$$

$$\sum_{p_{bv} \in P_{bv}} \sum_{v \in N_b} X_{p_{bv}}^t = 1, \forall b \in B, \forall t \in T, \tag{4}$$

$$X_b^t \in (0,1), \forall b \in B, \forall t \in T, \tag{5}$$

$$Y_v^t \in (0,1), \forall v \in V, \forall t \in T, \tag{6}$$

Constraint (2) guarantees the aggregated resource demands $\sum_{p \in P_{bv}} \sum_{b \in B} A_b^t X_{p_{bv}}$ from APs that are served by v is no more than its physical capacity limit k_v. This ensures a fixed service time at all time by allocating a dedicated resource unit for each uploading demand A_b^t. Constraint (3) guarantees that for all edges, the aggregated bandwidth consumption is less than the link capacity, where $P_{bv}(e)$ denotes all paths between b and v that traverse edge e; (4) guarantees that flows from the same b are assigned to the single v where $N_b = \{v | d_{bv} \leq D\}$ and the distance is no more than D. Constraints (3)-(4) ensure the network access latency is within D while constraints (2)-(4) guarantee a bounded service-level response time. Constraints (5) and (6) limit the decision variable to be either 0 or 1.

Our problem is NP-hard since a special case of the problem without bandwidth capacity constraints can be reduced from the capacitated set covering problem (CSCP). Since the CSCP problem is NP-hard [16], our problem is NP-hard too.

[2]Note that by fixing $T = \{t_0\}$, the problem is reduced to a static placement problem mentioned in Section I.

III. Dynamic resource allocation approach

Due to the NP-harness, our problem cannot be solved in polynomial time for both static and dynamic versions of the problem and thus, we turn to heuristics. Some heuristics [16] have been proposed to solve the static CSCP by relaxing it further into a capacitated plant location problem. However, we highlight that such offline approach is infeasible when we consider optimality of the system in real-time system when the stringent latency requirement of the considered applications does not allow for constant re-computation and re-deployment of service-hosting nodes.

A. Approach Overview

In our solution, we take into account existing techniques for coping with service elasticity (i.e., ALB) and find the time point when these mitigation tools will reach their limits (e.g., due to increasing load) and cause the MEC system to violate the service-level requirement of the considered application(s). A new service-hosting node deployment must be computed and put in place in time before service quality starts to degrade.

We propose a solution composed of (1) a capacity detection violation mechanism that takes into account the effect of ALB to address the question on *when* re-optimization is required and (2) a fast adaptive heuristic algorithm to address the question on *how* to cost efficiently adjust the current service-hosting node deployment based on the predicted increased load. The procedures of our approach are as follows.

1) We derive the initial optimal static planning in an offline fashion by solving the static placement problem at time t_0 using CPLEX [17].
2) We exploit ALB to cope with service elasticity based on the initial placement.
3) When the workload reaches a preset cloud capacity threshold, the system triggers our capacity violation detection based on the projected workload over a time window $\Delta t = t' - t$ where t' is the prediction time slot.
4) If it is detected that the ALB's limit will be reached within a time horizon, Δt, our adaptive greedy heuristic is invoked to derive the desired new service-hosting nodes' locations based on the previous placement solution.

B. Auto-Scaling and Load Balancing

Auto-scaling and load balancing are two current techniques to prevent MEC systems from constantly performing re-optimization due to workload variations. We adopt a reactive auto-scaling approach that is triggered once a specific capacity threshold is reached. However, auto-scaling incurs additional delays which could affect service-level response times. This effect can be mitigated by setting a smaller auto-scaling threshold to invoke the auto-scaling mechanism in advance.

For load balancing, we adopt a proximity-aware approach that considers both the residual capacity in service-hosting

Algorithm 1 Capacity Violation Detection (CVD)

```
Input: $G(V,E), B, S(X,Y,t), A^{t'}, v', k_{v'}$
Output: $t'$ and $L_{v'}$ or no re-optimization
1: if $\sum_{p_{bv'} \in P_{bv'}} \sum_{b \in B} A_b^{t'} X_{p_{bv'}}^t \geq k_{v'} Y_{v'}^t$ then
2:     $S(\hat{X},\hat{Y},t') = VALB(S(X,Y,t), G(V,E), A^{t'})$
3:     if $\sum_{p_{bv'} \in P_{bv'}} \sum_{b \in B} A_b^{t'} \hat{X}_{p_{bv'}}^{t'} \geq k_{v'} Y_{v'}^t$ then
4:         $L_{v'} = \sum_{p_{bv'} \in P_{bv'}} \sum_{b \in B} A_b^{t'} \hat{X}_{p_{bv'}}^{t'} - k_{v'} Y_{v'}^t$
5:         trigger OAG algorithm return $t', L_{v'}$
6:     else
7:         $Break$
8:     end if
9: end if
```

nodes and the topological proximity between service-hosting nodes and APs, so that the network latency is always bounded by the maximum number of hops allowed, D. In this approach, a flow from an AP to the overloaded service-hosting node will only be redirected when the new service-hosting node v is within the distance cover, N_b, and the residual capacity is enough to accommodate the load A_b^t at time t.

C. Capacity Violation Detection (CVD)

The aforementioned techniques have their limits, after which further increase in the request rate will incur increasing queuing delays at service-hosting nodes. The core idea of CVD is to identify early enough the time when such limitations are reached to allow the system to proactively allocate new resources. We first assume that we can reasonably predict the workload $A^{t'} = \cup_{b \in B} A_b^{t'}$ in advance based on existing workload prediction techniques [9], [18]. Given current system state, $S(X,Y,t)$, we predict over time window Δt the aggregated workload $\sum_{p_{bv'} \in P_{bv'}} \sum_{b \in B} A_b^{t'} X_{p_{bv'}}^t$ at v' (i.e., v' is the service-hosting node that invokes the detection) and check if the predicted workload results in a capacity violation at v' (Line 1 in Algorithm 1) for the current state. If the current state cannot accommodate future workload, we estimate the future system state by virtually running ALB on the current system state with the predicted workload.

After applying virtual ALB (VALB), CVD derives a new routes $\hat{X}$ for the virtual state $S(\hat{X},\hat{Y},t')$ where the service-hosting node locations $\hat{Y}$ are the same as Y from $S(X,Y,t)$ (Line 2). We check if the resulting routing $\hat{X}$ from $S(\hat{X},\hat{Y},t')$ still fails to accommodate $A^{t'}$ (i.e., violating the upfront capacity limit at v') (Line 3). If yes, it means ALB will reach its limit and it records the excess load that cannot be served by v' as $L_{v'}$. The online OAG heuristic is triggered then. It must be stressed that the new state $S(\hat{X},\hat{Y},t')$ is only estimated without actual ALB taking place.

D. Online Adaptive Greedy (OAG) Heuristic

The idea of OAG (Algorithm 2) is to search for a new service-hosting node location (i.e., within the distance constraint) that can accommodate the excess flow, $L_{v'}$, in the projected time but also one that can satisfy as many flows as possible to increase potential gain via load balancing to the new service-hosting node. Our OAG algorithm simultaneously determines the new placement of service-hosting node(s) and the corresponding routes. The heuristic is adaptive as it takes into account the current system state and evolves to a new state with the newly selected service-hosting node. Then, OAG greedily chooses the node v that has simultaneously the highest number of APs within its distance cover, denoted by $benefit(v)$ and can satisfy the excess load $L_{v'}$. We denote $R_{v'v} = \{b | d_{bv} \leq D, d_{bv'} \leq D, b \in B\}$ as the set of APs within the distance cover of both v and v' (node that invoked CVD). We then denote the chosen node by $v_{b_{max}} \in V \backslash v'$.

The details of the OAG algorithm are described below.

1) Line 4-8 find the set of APs, $B_{v'} = \{b | d_{bv'} \leq D, b \in B\}$, within the coverage of v'.
2) For each network location, $v \in V$, line 9-19 derive the corresponding $benefit(v)$ (i.e., the number of APs that could benefit from adding service-hosting node v). We add APs that belongs to v' but are also located within the distance constraint of v into $R_{v'v}$.
3) Line 20-29 record the total load from APs in $R_{v'v}$, L_v.
4) For each AP in the new set $b_{v'} \in B_{v'}$, we search v from $N_{b_{v'}} = \{v | d_{b_{v'}v} \leq D, v \in V\}$ and find $v_{b_{max}}$ that has the highest $benefit(v)$ and can support excess load $L_{v'}$.
5) If no $v_{b_{max}}$ has been found due to $L_{v'}$, we assign the v that has the largest L_v as v_{bmax}. This means no single node location that can host all excess flows $L_{v'}$ from v'. Then, OAG will be triggered again with a reduced $L_{v'} = L_{v'} - L_{v_{b_{max}}}$ to find the next location to add (Line 28-30).
6) We direct flows in $R_{v'v}$ from v' to $v_{b_{max}}$ and solve the routing problem by max-min fairness [19].

IV. Performance Evaluation

A. Simulation Setup

We implement a packet-level MEC framework based on an openflow module [20] of OMNeT++ [21]. We consider an augmented reality application [22] with 1,800 mobile users following the mobility traces of San Francisco taxi [14] within an area of $46km^2$. For simplicity, we assume homogeneous requests of the same size. Users upload street views (each frame is of size 0.5MB [22]) captured by mobile devices and wait for the notations (e.g., building name, available parking places, etc.) to be returned from MEC. Following the three-level network in Fig. 1, we set up 30 APs, 5 aggregation nodes and 5 core network nodes. The bandwidth and additional network delays [23] caused by background traffic are shown in Fig. 1. Once a frame arrives at the service-hosting node, it requires $\mu = 230ms$ for a VM with 600MHz CPU to process [22] after which, a response packet of 4KB will be sent to users. For this considered scenario, we aim to achieve a response time requirement of less than 480ms [1].

We further consider two different service-hosting node sizes [24] in this work: (Full) a service-hosting node of 21 servers supporting a maximum of 132 VMs and (Half) a service-hosting node of 10 servers supporting a maximum of 66 VMs.

Algorithm 2 Online Adaptive Greedy (OAG)

```
Input: $G(V,E), B, D, S(X,Y,t), v', L_{v'}, A^{t'}$
Output: $S(\hat{X},\hat{Y},t')$
1: $v_{bmax} \leftarrow \emptyset$
2: $B_{v'} \leftarrow \emptyset$
3: $\forall v \in V, R_{v'v} \leftarrow \emptyset$
4: for all $b \in B$ do
5:     if $d(b,v') \leq D$ then
6:         $B_{v'} \leftarrow B_{v'} \cup b$
7:     end if
8: end for
9: for all $v \in V$ do
10:     for all $b \in B$ do
11:         if $d(b,v) \leq D$ then
12:             $benefit(v) \leftarrow benefit(v) + 1$
13:             if $b_{v'} \in B_{v'}$ then
14:                 $R_{v'v} \leftarrow R_{v'v} \cup b_{v'}$
15:                 $L_v \leftarrow L_v + A^t_{b_{v'}}$
16:             end if
17:         end if
18:     end for
19: end for
20: for all $b_{v'} \in B_{v'}$ do
21:     for all $v \in N_{b_{v'}}$ and $v \neq v'$ do
22:         if $L_v \geq L_{v'}$ then
23:             if $benefit(v) \geq benefit(v_{bmax})$ then
24:                 $v_{bmax} \leftarrow v$
25:             end if
26:         end if
27:     end for
28: end for
29: if $v_{bmax} == \emptyset$ then
30:     $v_{bmax} \leftarrow argmax(L_v)$
31:     trigger OAG with $L_{v'} = L_{v'} - L_{v_{bmax}}$
32: end if
33: $\hat{X} \leftarrow MaxMinFaireness(v_{bmax}, R_{v'v_{bmax}})$
34: $\hat{Y} \leftarrow Y \cup v_{bmax}$
35: return $S(\hat{X},\hat{Y},t')$
```

Each server has a 2.1GHz CPU of 18 cores. The initial node capacity, placement and routes are derived by CPLEX solver in an offline manner given the aforementioned CPU consumption of each frame, initial locations of mobile users and user-to-AP association. In addition, a maximum of four hops from AP to service-hosting node is set (i.e., $D = 4$) to constrain the network access delay. Solving the static placement problem described above, we get two service-hosting nodes (72 and 108 VMs) at core network level for full service-hosting node and three service-hosting nodes (60, 61, 59 VMs) at core network level for the half service-hosting node size respectively.

Simulation duration is set to 1 hour. It starts from the aforementioned initial state. We gradually increase the workload of the network starting from 0.3FPS to 3.0FPS in steps of 0.1FPS every 400s. Whenever the VM size reaches a threshold of 80%, VM auto-scaling is triggered with an instantiation time of 10ms-600ms [25]. For Algorithm 2, we also set the workload prediction time window, $\Delta t = 400s$ [9]. Moreover, we assume the network bandwidth is sufficient so that there are no network bottlenecks[3]. Hence, we only consider queueing delays incurred at service-hosting nodes.

B. Evaluation Results

For evaluation, we compare the following schemes:

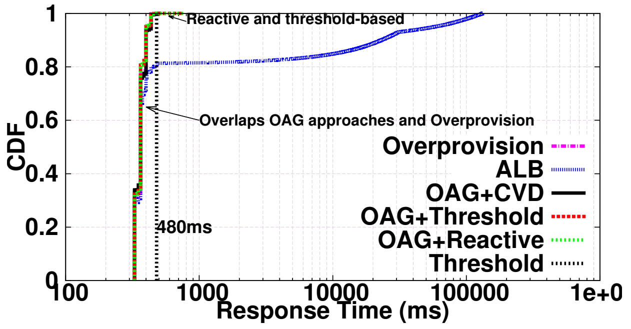
[3]We leave bandwidth violation detection for our future work in which we could apply a network calculus approach similar to [26].

TABLE I: Performance Comparison.

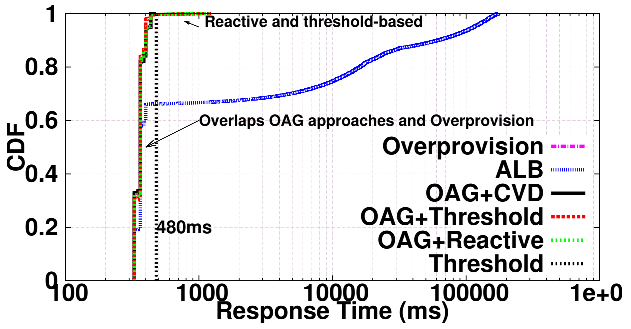
	Full service-hosting node				Half service-hosting node			
	QoS	Max	Nb	Saving(%)	QoS	Max	Nb	Saving(%)
Overprovision	Yes	480ms	3,3	0%	Yes	472	6,6	0%
ALB	No	132s	2,2	42.6%	No	175s	3,3	51.5%
OAG+CVD	Yes	480ms	2,3	33.6%	Yes	478ms	3,6	38.4%
OAG+Threshold	No	758ms	2,3	34.4%	No	1.28s	3,6	41.2%
OAG+Reactive	No	758ms	2,3	34.3%	No	1.23s	3,6	41.2%

1) *Overprovisioning* – We first solve the static placement problem and then, for each chosen location, we overprovision the maximum possible physical capacity to serve user requests. ALB are never needed in this case. We denote this by *Overprovision* in Table I.
2) *Simple ALB* – ALB uses the initial solution from the static placement problem. The network performs ALB on the initial service-hosting locations when needed but without further re-optimization. We denote this by *ALB*.
3) *OAG with CVD* – Our proposed solution, combining Algorithm 1 and 2. We denote this by *OAG+CVD*.
4) *OAG with threshold-based detection* – This adopts OAG but applies a threshold-based detection mechanism that invokes OAG when reaching 80% of service-hosting node's physical capacity. We denote this by *OAG+Threshold*.
5) *Reactive OAG* – This adopts OAG but only invokes the OAG when the service-hosting node's physical capacity is full. We denote this by *OAG+Reactive*.

Table I shows our results with respect to satisfaction of the response time requirement ("QoS" column), maximum response time ("Max" column), number of resulting service-hosting nodes at the start and end of the simulation ("Nb" column) and the savings of allocated resources in comparison to *Overprovision* ("Saving" column). Besides the costly *Overprovision* approach, only our proposal, *OAG+CVD*, manages to satisfy the delay requirement of the application. We show the cumulative distribution function (CDF) in Fig. 2(a) and Fig. 2(b) that *OAG+CVD* overlaps with the *Overprovision* approach in both cases, which indicates the seamless transition to the new system state without crossing the 480ms application constraint. In contrast, *OAG+Reactive* and *OAG+Threshold* exceed this 480ms threshold due to the late detection. We observe that *ALB* results in lowest number of service-hosting nodes, but it comes with delay penalties. When we compare the allocated virtual resources over time against *Overprovision*, ALB achieves respectively a saving of 42.6% and 51.5% in full and half service-hosting node settings. In contrast, our *OAG+CVD* leads to a more modest saving (i.e., 33.6% and 38.4% respectively in the two cases) but achieves the latency requirement. We also observe from Table I that all approaches involving OAG add new service-hosting nodes based on the initial ones. For the full size setting, the service-hosting node number changes from 2 to 3 and for half size setting, it increases from 3 to 6 in response to the increased workload. Both *OAG+Threshold* and *OAG+Reactive* achieve lower costs compared to *OAG+CVD* because they fail to instantiate new service-hosting nodes on time to satisfy the



(a) CDF Response Time of Different Approaches (Full Size)



(b) CDF Response Time of Different Approaches (Half Size)

Fig. 2: (Color Online) CDF Response Time

applications latency requirements.

V. Conclusions

We address the challenge of seamless support for delay sensitive mobile multimedia applications within an NFV-enhanced mobile edge-cloud (MEC) environment. Specifically, we formulate an optimization problem for placing MEC services at NFV-enabled nodes so that resources are optimally allocated to satisfy the applications latency requirements while incurring minimum costs to ISPs. Since the problem is NP-hard, we designed an online adaptive greedy heuristic (OAG) algorithm to find the best placement of MEC services so that sufficient resources are always available to ensure no latency violation. While most previous work has focused on optimizing the system for a specific time snapshot, we are also concerned with the system performance over time when the workload may change significantly. To address this, we propose a capacity violation detection (CVD) mechanism that estimates the time when current existing mitigation tools (i.e., auto-scaling and load balancing) will fail to cope with service load elasticity. Using this projected time, we can invoke our proposed OAG to pre-emptively allocate new virtual resources near users to ensure continuous satisfaction of the applications requirements. Using a realistic NFV-enabled MEC simulation framework, we evaluated our proposal against the current best practices. Our detailed packet-level results show that only our proposal always ensure MEC services respond to user requests on time. Our flexible service-hosting node solution also offers up to 40% cost saving in comparison to a costly overprovisioning approach at a fixed location and solves the latency violation problem when the number and capacity of service-hosting nodes is reduced for cost saving purposes.

Acknowledgment

The research leading to these results has received funding from the CHIST-ERA/EPSRC UK project CONCERT (EP/L01835/1), the EU FP7 Flamingo Network of Excellence project (318488) and the EC H2020 UMOBILE project (645124).

References

[1] P. Jain, J. Manweiler, and R. Roy Choudhury, "Overlay: Practical mobile augmented reality," in *ACM MobiSys*, 2015, pp. 331–344.
[2] K. Ha *et al.*, "The impact of mobile multimedia applications on data center consolidation," in *IEEE Conf. on IC2E*, 2013, pp. 166–176.
[3] M. ETSI, "Mobile-edge computing," *Introductory Technical White Paper, Sept.*, 2014.
[4] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
[5] J. Soares *et al.*, "Toward a telco cloud environment for service functions," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 98–106, 2015.
[6] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.
[7] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys & Tutorials*, 2015.
[8] N. Bouten *et al.*, "Towards nfv-based multimedia delivery," in *Symp. on IEEE IM*, 2015, pp. 738–741.
[9] D. Niu *et al.*, "Quality-assured cloud bandwidth auto-scaling for video-on-demand applications," in *IEEE INFOCOM*, 2012, pp. 460–468.
[10] Y. Zhu and Y. Hu, "Efficient, proximity-aware load balancing for dht-based p2p systems," *Parallel and distributed systems, IEEE Trans. on*, vol. 16, no. 4, pp. 349–361, 2005.
[11] A. Ceselli, M. Premoli, and S. Secci, "Cloudlet network design optimization," in *IFIP Networking*, 2015.
[12] Z. Xu *et al.*, "Capacitated cloudlet placements in wireless metropolitan area networks," in *IEEE LCN*, 2015, pp. 570–578.
[13] S. Wang *et al.*, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking*, 2015.
[14] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *IEEE Communication Systems and Networks and Workshops*, 2009, pp. 1–10.
[15] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, 2012.
[16] J. Current and J. Storbeck, "Capacitated covering models," *Environment and Planning B*, vol. 15, pp. 153–164, 1988.
[17] I. I. CPLEX, "V12. 1: User manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.
[18] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *IEEE CLOUD*, 2011, pp. 500–507.
[19] B. Radunović and J.-Y. L. Boudec, "A unified framework for max-min and min-max fairness with applications," *IEEE/ACM Trans. on Networking*, vol. 15, no. 5, pp. 1073–1083, 2007.
[20] D. Klein and M. Jarschel, "An openflow extension for the omnet++ inet framework," in *ICST Conf. on Simulation Tools and Techniques*, 2013.
[21] A. Varga, *OMNeT++ Simulator Home Page*, http://www.omnetpp.org.
[22] R. LiKamWa and L. Zhong, "Starfish: Efficient concurrency support for computer vision applications," in *ACM MobiSys*, 2015, pp. 213–226.
[23] K. Wang *et al.*, "Mobiscud: A fast moving personal cloud in the mobile network," in *ACM Workshop on All Things Cellular: Operations, Applications and Challenges*, 2015, pp. 19–24.
[24] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," in *Conf. on IEEE LCN*, 2013.
[25] A. Madhavapeddy *et al.*, "Jitsu: Just-in-time summoning of unikernels," in *12th USENIX Symp. on NSDI*, 2015, pp. 559–573.
[26] B. Yang *et al.*, "Cost-efficient low latency communication infrastructure for synchrophasor applications in smart grids," *IEEE Syst. J., DOI:10.1109/JSYST.2016.2556420.*