QoS-Aware Task Offloading in Distributed Cloudlets with Virtual Network Function Services

Mike Jia
Research School of Computer Science
The Australian National University
Canberra, ACT, Australia 2601
u5515287@anu.edu.au

Weifa Liang
Research School of Computer Science
The Australian National University
Canberra, ACT, Australia 2601
wliang@cs.anu.edu.au

Zichuan Xu
School of Software
Dalian University of Technology
Dalian, China 116024
z.xu@dlut.edu.cn

ABSTRACT

Pushing the cloud frontier to the network edge has attracted tremendous interest not only from cloud operators of the IT service/software industry but also from network service operators that provide various network services for mobile users. In particular, by deploying *cloudlets* in metropolitan area networks, network service providers can provide various network services through implementing virtualized network functions to meet the demands of mobile users. In this paper we formulate a novel task offloading problem in a metropolitan area network, where each offloaded task requests a specific network function with a maximum tolerable delay and different offloading requests may require different network services. We aim to maximize the number of requests admitted while minimizing their admission cost within a finite time horizon. We first show that the problem is NP-hard, and then devise an efficient algorithm through reducing the problem to a series of minimum weight maximum matching in auxiliary bipartite graphs. We also consider dynamic changes of offloading request patterns over time, and develop an effective prediction mechanism to release and/or create instances of network functions in different cloudlets for cost savings. We finally evaluate the performance of the proposed algorithms through experimental simulations. Experimental results indicate that the proposed algorithms are promising.

KEYWORDS

cloudlets; task offloading; request QoS requirement; functionality service virtualization; request admission cost minimization; network function virtualization; offloading algorithms; wireless metropolitan area networks; resource allocation of cloudlets.

1 INTRODUCTION

Mobile devices such as smart phones and tablets have become the main communication tools of users for business, social networking, and personal banking. Due to their portable size, the computing/storage and energy powering these mobile devices are critical, making their processing and storage ability very limited. One promising technique is to offload their tasks to nearby cloudlets via WiFi or bluetooth for processing. For example, an inspector takes a photo of a stranger in his monitoring area, and uploads the photo to the cloudlets to verify the identity of the stranger, assuming that there is a database of personal identities on the cloudlets. Cloudlet technology also looks particularly promising for emerging augmented reality (AR) applications and products. In the recent 2017 F8 Developers Conference, Facebook CEO Mark Zuckerberg laid out a vision of the near future where artists could display virtual artwork in public spaces [1], and friends could share virtual notes and objects with each other, using their mobile devices. From these examples, it can be seen that different offloading tasks have different network functions and quality of service requirements. As different network functions in cloudlets are implemented by different virtual machines (referred to as Virtualized Network Functions (VNFs)), we may group all offloading tasks with the same network function services together and implement them in virtual machines. To share the computing resources among offloaded tasks with the same network service while meeting their individual QoSs, network service providers usually instantiate some instances of the VMs of each network function service in each cloudlet.

Provisioning network services in a mobile edge cloud, having instantiated instances of network function services in its cloudlets, poses several challenges. That is, how many instances are to be instantiated at which cloudlets, such that the computing resource of cloudlets can be maximally utilized while the cost and delay of instance instantiation can be minimized? How should offloaded tasks be assigned to different cloudlets while meeting their QoSs? and how can the admission cost of offloaded task requests be minimized by utilizing existing instances of their requested network function services? Finally, how can the number of required instances be predicted, and how should the creation and removal of instances be managed in the network? In this paper we will address these challenges, by comprehensively studying the problem of task offloading with network function service requirements in a mobile edge cloud. To the best of our knowledge, we are the first to explore the possibility of utilizing existing VNF instances of network function services in cloudlets for cost-effective task offloading while meeting different QoS requirements of offloaded tasks, by formulating a novel QoS-aware task offloading optimization problem, and providing an optimization framework. To respond to dynamic changes of offloading request patterns over time to further reduce request admission costs, we also develop an effective prediction mechanism to predict the instance demands in the future through the removal and creation of different numbers of VNF instances of each network function service at each cloudlet.

MSWiM '17, November 21–25, 2017, Miami, FL, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The main contributions of this paper are as follows. We first formulate a novel QoS-aware task offloading problem in a wireless metropolitan area network consisting of Access Points (APs) and cloudlets (servers) that are co-located with the APs, where each offloading task request has a specific network function requirement with a given tolerable delay, and different offloading tasks request have different VNFs. We assume some instances of the VNFs have already been instantiated on the cloudlets, and others can be dynamically created if there are sufficient resources in cloudlets. We aim to maximize the number of offloading requests admitted within a finite time horizon while minimizing the admission cost of requests and meeting their individual end-to-end delay requirements. We achieve this by fully making use of the VNF instances in the cloudlets to reduce the admission cost and shorten the response time of the requests. We then devise an efficient online algorithm for offloading request admissions through a non-trivial reduction that reduces the problem to a series of minimum-weight maximum matching problems in auxiliary bipartite graphs. We also investigate offloading request patterns over time, and develop an effective prediction mechanism that can predict the numbers of instances of each network function at different cloudlets, through releasing the occupied resources by idle VNF instances back to the system, and creating new VNF instances of other highly demanded network functions in cloudlets to meet the need of future offloading task requests. We finally evaluate the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising.

The remainder of the paper is arranged as follows. Section 2 will survey the state-of-the-arts on this topic, and detail the difference of this work from previous studies in task offloading. Section 3 will introduce the system model, notations and problem definition. Section 4 will devise algorithms for the problem. Section 5 will provide some experimental results on the performance of the proposed algorithm, and Section 6 concludes the paper.

2 RELATED WORK

Offloading tasks to cloudlets has been extensively studied in the past several years. Generally, the model for application offloading systems [2, 3] in mobile cloud computing consists of a client component on a mobile device, and a server component on the cloudlet to remotely execute offloaded tasks from the device. As the options for user applications are too numerous for server components to be stored in the cloudlet, most works [2, 4, 5] assume the use of virtual machines (VM) as a platform for task offloading. A VM image of a mobile device is transferred to the cloudlet, and tasks are remotely executed on the device's VM in the cloudlet, using task offloading operations. Once a task on the VM in the cloudlet has been executed, the result will be returned to its user.

Most previous studies assumed that each user connects to a dedicated VM in the cloudlet, without consideration of whether an existing VM for the same application could be used to serve multiple users. However, as many emerging applications and services are location-specific, it is more realistic to assume that multiple users in a local area will request the same computing service from cloudlets. In [6], the authors introduced a mobile task offloading architecture specifically for mobile augmented reality in a museum setting. In their model, a user turns on his mobile device's camera to capture a scene. The location and direction of the camera are calculated for each captured video frame and the data is sent to a nearby cloudlet where a rendering process generates the 2D image of a virtual exhibit, which is then overlayed on top of the original video frame. The resulting image is then sent back to the user's device for display, creating the impression of seeing the virtual exhibit in the real world through the camera viewport. Since the processing of each user video frame can be modeled as an individual task, it is possible for a VM instance on a cloudlet to serve multiple users. However, it then becomes a challenge to assign users to existing VM instances, or create additional instances to serve more users while ensuring the QoS requirement of each user is met, as they share computing resources on the cloudlet.

In recent years, several studies focused on network planning problems in deploying cloudlets for public use. For example, Jia et al [7] considered the assignment of user requests to different cloudlets in a WMAN, by developing a heuristic for it. Jia et al [8] also dealt with minimizing the maximum delay among offloaded tasks in a distributed cloudlet network through balancing the workload between cloudlets. Xu et al [9, 10] devised assignment algorithms for user offloading requests to different cloudlets, by proposing efficient approximation and online algorithms with performance guarantees. Xia et al [11] considered opportunistic task offloading under link bandwidth, mobile device energy, and cloudlet computing capacity constraints. Xia et al [12] studied the location-aware task offloading problem for mobile devices. All of these mentioned studies assumed that each offloaded task will be assigned the amounts of computing resources they demanded, there is no consideration for whether there are already VMs in cloudlets for serving them, not to mention whether such services meet their QoS requirements. However, the emergence of AR applications and IoT computing strongly suggest that many offloaded tasks may request for the same type of services in the near future. If the VM for that service has been established, the offloading cost will be less expensive and the service can be carried out immediately.

3 PRELIMINARIES

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

3.1 System model

Given a WMAN $G = (V \cup C, E)$ where V is the set of AP nodes and E is the set of links between AP nodes. There is a subset of nodes $C \subseteq V$ of cloudlets co-located with the AP nodes. Each cloudlet $c_j \in C$ has computing capacity cap_j with $1 \leq j \leq m$ and $m = |C|$. Assuming that time is divided into equal time slots. The amounts of available resources at the beginning of different time slots varies, due to request admissions and departures. Let $cap_j(t)$ be the available computing capacity of cloudlet c_j at time slot t with $1 \leq j \leq m$.

Computing resource in the cloudlets is used to instantiate a certain number of VNFs to implement offloading requests from mobile users. We thus assume that there is a set of network functions $f_i \in \mathcal{F}$ with $1 \leq i \leq N$ and $N = |\mathcal{F}|$, which are virtualized and implemented in VMs in cloudlets. If the implementation of an

offloading request with a network function $f_i \in \mathcal{F}$ demands *the basic resource unit* of f_i, we term this implementation as *an instance* of network function f_i for the request, or *an VNF instance* of f_i; otherwise (if the implementation of the request needs x (≥ 1) times the basic resource unit of f_i), we term that the request implementation takes x instances of f_i for each $f_i \in \mathcal{F}$. We further assume that each cloudlet has instantiated some instances of virtualized network functions. Denote by $n_{ij}(t)$ the number of instantiated instances of f_i in cloudlet c_j at time slot t.

Each mobile device can offload its tasks to cloudlets in G, via APs. Consider a set of requests for offloading their tasks to the cloudlets in G for processing, each request with a specified network service will be implemented in a VM of that function, which is termed as an instance of the specified virtualized network function (VNF). Let $S(t)$ be the set of user requests at time slot t. Each user request $r_k \in S(t)$ is represented by a tuple $(id_k, loc_k, VNF_k, \lambda_k, d_k)$, where id_k is the request identity, loc_k is the location of the request user, VNF_k is the computing service that r_k requests, which in fact is a virtualized network function, $\lambda_k \geq 0$ is the packet rate of r_k, and d_k is the end-to-end delay requirement of r_k.

3.2 End-to-end delay requirements of offloading requests

The end-to-end delay experienced by each admitted request r_k include the queuing delay that it spent in waiting for an available instance of its VNF, processing delay by its assigned VNF instance, instantiation delay of creating a new VNF when necessary, and network latency from its location loc_k to its assigned cloudlet c_j.

Queuing delay and processing delay: each offloaded packet with packet rate λ_k of r_k will be queued in the VM of VNF_k in a cloudlet prior to its processing by the VM, which will incur both queuing and processing delays when each packet passes through the VM of the VNF. To differentiate user requests with different delay requirements, the requests in each cloudlet c_k are partitioned into N groups with each group consisting of requests for the same service. We thus assume that there is an $M/M/n$ queue at each cloudlet for each type of service $f_i \in \mathcal{F}$. Each group of requests will eventually be processed by instances of network service $f_i \in \mathcal{F}$, with $1 \leq i \leq N$. The average queuing delay of the $M/M/n$ queue for function f_i at cloudlet c_j thus is

$$\tau_{kj}(\lambda) = \frac{1}{n_{ij}(t)\mu_i - \lambda}, \tag{3.1}$$

where λ is the sum of packet rates of all requests that require VNF f_i and are assigned to cloudlet c_j, and μ_i is the data processing rate of VNF f_i. Considering that the data processing rate of VNF f_i is μ_i, the processing delay of f_i thus is $\frac{1}{\mu_i}$.

Instantiation delay: without loss of generality, we assume that the instantiation delay of an VNF instance is a given constant d_i^{ins} for VNF f_i.

Network latency: assuming that data traffic in network G is transferred via a shortest path between each pair of source and destination, the network latency of request r_k thus is the accumulative delay incurred in the edges of a shortest path p_{loc_k, c_j} from its source location loc_k to its assigned cloudlet c_j. Let $d(e)$ be the delay of link e of network G, the network latency d_k^{net} thus is

$$d_k^{net} = \sum_{e \in p_{loc_k, c_j}} d(e). \tag{3.2}$$

The end-to-end delay D_k experienced by a request r_k for VNF f_i at cloudlet c_j thus can be calculated by

$$D_k = \begin{cases} \tau_{kj} + \frac{1}{\mu_i} + d_k^{net}, & \text{if } n_{ij}(t) > 0 \\ d_i^{ins} + \frac{1}{\mu_i} + d_k^{net}, & \text{otherwise.} \end{cases} \tag{3.3}$$

The end-to-end delay requirement of each offloading request r_k thus is

$$D_k \leq d_k. \tag{3.4}$$

3.3 The admission cost

For each request $r_k \in S(t)$ with network function f_i ($= VNF_k^i$), its implementation can either make use of some of existing VNF instances of f_i in a cloudlet c_j if it joins in other admitted requests with the same network function f_i, and the delay requirements of all requests can be met. Specifically, let R_{ij} be the set of offloaded requests with network function f_i in cloudlet c_j when r_k is being considered, and assume that the admission of r_k to R_{ij} will not violate the delay constraint of any of them. The operational cost of admitting r_k in c_j then is the cost sum of its data packet transmission cost (between its location via its nearby AP) and cloudlet c_j and its processing cost $c(VNF_k^i)$ at c_j. Otherwise (the addition of r_k resulting in the violation of computing or delay constraints of requests in R_{ij}), if there are available computing resources in cloudlet c_j, we then allocate the demanded resources for r_k by increasing the number of instances for VNF_k^i in c_j, the admission cost $w(r_k)$ of r_k per packet thus is the cost sum of its packet transmission cost, the creation of new instances for r_k, and its processing cost in c_j.

3.4 Problem definition

Given a WMAN $G(V \cup C, E)$, a set of user requests $S(t)$ at time slot t with each request having an end-to-end delay requirement, and a finite time horizon T, assume that the set of network functions by the requests in $S(t)$ is $\mathcal{F}$, and some instances of each network function $f \in \mathcal{F}$ have already been installed in cloudlets C, *the operational cost minimization problem* in G is to find a schedule of request admissions such that as many requests as possible are admitted during a monitoring period of T while the cumulative operational cost of admitted requests is minimized, subject to the computing resource capacity constraint, and end-to-end delay requirement of each user request.

Theorem 3.1. *The operational cost minimization problem in $G(V \cup C, E)$ is NP-hard.*

Proof We consider an extreme case where there are only two cloudlets in the network with identical computational capacities. We assume that each request in $S(t)$ has a different service (i.e., a different network function), we can ignore the delay requirement of each request. Our task is to assign the requests to the two cloudlets to see whether all of the requests can be admitted. Clearly, for each request $r_k \in S(t)$, we need to create a VM for implementing its network function that is associated with computing resource

demand c_k, subject to the computing capacity constraints on the two cloudlets.

We reduce the well-known summation problem to the mentioned assignment problem in polynomial time as follows. Given n positive integers $a_1, a_2, \ldots, a_n$, the summation problem is to partition the n integers into two subsets such that the sum of integers in each subset is equal, which is NP-hard. As the special case of the minimum operational cost problem is equivalent to the summation problem, the operational cost minimization problem thus is NP-hard too.

4 ONLINE ALGORITHM

In this section we first consider admissions of requests in $S(t)$ in the beginning of each time slot t. We then deal with dynamic request admissions within a finite time horizon T, by proposing an efficient online algorithm for the operational cost minimization problem in WMAN $G(V \cup C, E)$.

4.1 Algorithm for offloading requests at each time slot

Given a set of arrived requests $S(t)$ in the beginning of each time slot t, we aim to admit as many requests in $S(t)$ as possible while minimizing the admission cost of the admitted requests and meeting their delay requirements. The basic idea behind our algorithm is to reduce the operational cost minimization problem in G to a series of minimum weight maximum matching problems in a set of auxiliary bipartite graphs. Each matched edge in the maximum matching of an auxiliary bipartite graph corresponds to an assignment of offloading requests to cloudlets in the network G, where the end-to-end delay requirement of each admitted request can be met. The detailed description of this reduction is as follows.

For each cloudlet c_j, we construct a bipartite graph $G_j(t) = (X_j \cup \{x_{0,j}\}, Y_j, E_j; w)$, where X_j is the set of VNF instances in cloudlet c_j, and $x_{0,j}$ represents available resources for creating new instances for any of network functions in c_j, and Y_j is the set of requests $r_k \in S(t)$. There is an edge in E_j between a node $v_{ij} \in X_j$ and $r_k \in Y_j$ if sharing the resources for r_k does not violate the resource and delay requirements of other running requests for the network function. Specifically, there is an edge between $r_k \in X_j$ and the instance node of f_i in cloudlet c_j if (i) the VNF of r_k is f_i; (ii) the demanded instance of f_i by r_k is no greater than $n_{ij}(t)$; (iii) the addition of r_k into the set of admitted requests sharing the instance does not violate the delay constraints of other admitted requests in R_{ij}; and (iv) the total delay incurred by the assignment of r_k is no greater than d_k. The weight assigned to this edge is the cost of implementing request r_k in cloudlet c_j, which consists of the routing cost between the mobile device location and the cloudlet and the cost of processing the packet at the VNF instance of f_i. There is an edge between r_k and $x_{0,j}$ if its demanded computing resource is no greater than $cap_j(t)$ and its delay is no greater than d_k (including the instance creation delay). The weight of the edge thus is the sum of the routing cost, the processing cost and the instance creation cost for the request.

Assume that there are m cloudlets in G. An auxiliary bipartite graph $G(t) = \bigcup_{j=1}^{m} G_j(t) = (X(t), Y(t), E(t); w)$ is then derived from G, where $X(t) = \cup_{j=1}^{m}(X_j \cup \{x_{0,j}\})$, $Y(t) = \sum_{j=1}^{m} Y_j = \{r_1, r_2, \ldots, r_n\}$, and $E(t) = \cup_{j=1}^{m} E_j$.

To admit requests in $S(t)$ in the beginning of each time slot t, the admission algorithm proceeds iteratively. Let $G_1(t) = G(t)$. Within iteration l with $1 \le l \le m$, a minimum weight maximum matching M_l in $G_l(t)$ is found. Then, allocate the demanded resources for the requests in M_l, remove all matched requests in M_l from $S(t)$, update the available instances and cloudlet resources at each cloudlet in the network, and construct the next auxiliary bipartite graph $G_{l+1}(t)$. This procedure continues until there are no matchings in $G_{l+1}(t)$.

The union of all found minimum weight maximum matchings $\cup_{l=1}^{m} M_l$ forms a solution to the problem, i.e., each matched edge corresponds to an admission of a request in $S(t)$. The weighted sum $\sum_{l=1}^{L} c(M_l)$ of the edges in $\cup_{l=1}^{L} M_l$ is the implementation cost of admitted requests in $S(t)$, where L is the number of iterations which depends on requests in $S(t)$. Alternatively, $L \le \max_{1 \le l \le L}\{deg(G_l(t))\}$ which $deg(G_l(t))$ is the maximum degree of nodes in auxiliary graph $G_l(t)$. The details are given in `Algorithm` 1.

Algorithm 1 Admission_Algorithm_Each_Time_Slot $(G(t), S(t))$

Require: m cloudlets with each having its available resource capacity $cap_j(t)$, the number of instances of VNFs of each $f_i \in \mathcal{F}$, and a set of requests $S(t)$ at each time slot t.

Ensure: maximize the number of requests admitted (i.e., a subset $S'(t) \subseteq S(t)$) for each time slot t while minimizing the total admission cost. For each request r_k, if it is admitted, then to which cloudlet it will be sent and to which instance of VNF it should join/create will be given in the solution.

/* perform request admissions for requests in $S(t)$ */
$M_t \leftarrow \emptyset$; $cost_t \leftarrow 0$; /* the assignment of requests in $S(t)$ while minimizing their implementation cost $cost_t$ */
Construct the weighted bipartite graph $G(t)$;
$G_1(t) \leftarrow G(t)$; $l \leftarrow 1$; $cost_t \leftarrow 0$;
while there is a minimum weight maximum matching M_l in $G_l(t)$ **do**
    Find the minimum weight maximum matching M_l in $G_l(t)$, by invoking an efficient algorithm for the weighted maximum matching;
    if $M_l \neq \emptyset$ **then**
        $M_t \leftarrow M_t \cup M_l$;
        $c(M_l) \leftarrow \sum_{e \in M_l} w(e)$;
        $cost_t \leftarrow cost_t + c(M_l)$;
        Allocate resources to the requests in M_l;
        Update the amounts of available resources and instances of each network function at each cloudlet;
        $S(t) \leftarrow S(t) \setminus r(M_l)$; /* Remove requests in M_l from $S(t)$, where $r(M_l)$ is the set of requests in M_l */
        $l \leftarrow l + 1$;
        Construct $G_l(t)$ according to the updated resources, instances of VNFs, and the request set $S(t)$;
return M_t corresponds to the assignment of requests in $S(t)$, while $cost_t$ is their implementation cost.

4.2 Online algorithm for the minimum operational cost problem

In the previous subsection, we considered the admissions of offloading requests within one time slot. In reality, requests arrive into or depart from the system dynamically, request admissions at the current time slot should take into account their impact on the admissions of requests in future.

Notice that on one hand, new VNF instances of some network functions have been created to admit newly arrived requests. On the other hand, idle VNF instances will be released back to the

system if they will not be used in the near future. Two types of simple solutions can be adopted to handle such resource releases: (1) never release VNF instances in case of being used by future requests; (2) immediately release VNF instances that become idle. The first solution is based on the rationale that some VNF instances will be shared with subsequent admitted requests. Thus, in spite of the departures of some admitted requests, their occupied resources (or VNF instances) will still be kept by their VMs without releasing back to the system. Consequently, more and more VNF instances of each network function in the system will become idle, while the available resources at each cloudlet become more scarce. This will incur unnecessary operational costs, while at the same time preventing new requests from being admitted due to the lack of VNF instances or computing resources. The second solution is to avoid maintaining idle instances of VNFs. However, if a VNF instance is demanded by a request right after its release, the delay requirement of an admitted request might be violated, considering that creating a new instance for it will incur a delay. To avoid such situations, we make VNF creation and release decisions based on a smart prediction method that predicts the idle VNF instance releases and new VNF instance creations, such that the operational cost is minimized while the delay requirements of admitted requests are still met.

In the following we propose a prediction mechanism to predict idle VNF instance releases and new VNF instance creations to respond to changing request patterns over time. Thus, the system will perform resource collection, by releasing the occupied resources by idle VNF instances back to the system if the cost overhead on maintenance of these idle VNFs is beyond a given threshold after a certain time slots. Specifically, let $n_{ij}(t)$ be the number of VNF instances of f_i in cloudlet c_j at time slot t, its actual usage number is $n'_{ij}(t)$ ($\leq n_{ij}(t)$), the number of idle VNF instances of f_i in cloudlet c_j in time slot t thus is

$$\delta_{ij}(t) = n_{ij}(t) - n'_{ij}(t). \tag{4.1}$$

Let each idle instance of f_i in cloudlet c_j incur a fixed cost γ_{ij} at each time slot, and there is a given cost overhead threshold θ ($\geq n_0 \cdot \max_{f_i \in \mathcal{F}}\{c(f_i)\}$). The system will release the occupied resources by idle VNF instances at a specific time slot if the accumulative cost of these idle VNF instances at that time slot is greater than the given threshold θ. Clearly, at least n'_{ij} VNF instances of f_i should be kept in order to meet the end-to-end delay requirements of the running requests in $R_{ij}(t)$. However, consider the worst scenario where an VNF instance of f_i is just released back to the system at the current time slot, only to have the same instance be created again at the next time slot to accommodate a new request. To avoid this, we develop an efficient prediction method to determine the expected number of instances of each network function to be kept in the system. To determine which idle VNF instances should be released to the system, we make use of historic offloading request traces (patterns) at each cloudlet to predict the number of VNF instances needed of each network function in that cloudlet in future. Specifically, we adopt an auto-regression method to predict the number of VNF instances $\hat{n}_{ij}(t)$ of f_i in cloudlet c_j at the next time slot,

$$\hat{n}_{ij}(t) = \alpha_1 n_{ij}(t-1) + \alpha_2 n_{ij}(t-2) + \ldots + \alpha_k n_{ij}(t-k), \tag{4.2}$$

where $\alpha_{k'}$ (> 0) is a constant with $0 \leq \alpha_{k'} \leq 1$, $\sum_{l=1}^{t} \alpha_l = 1$, and $\alpha_{k_1} \geq \alpha_{k_2}$ if $k_1 < k_2$. Thus, the number of VNF instances of f_i in cloudlet c_j should be kept after time slot t is $\max\{\hat{n}_{ij}(t), n'_{ij}(t)\}$.

Similarly, if the number of VNF instances of a network function f_i keeps growing at each time slot, by adding extra computing resources to its VM, more VNF instances of that network function will be created. This incurs an extra cost at each instance creation. Instead, we may create the expected number of VNF instances at once to meet its future need, instead of adding computing resources for each new request to its VM incrementally. This can be achieved by using the similar auto regression method. That is, let $a_{ij}(t)$ be the number of new VNF instances of f_i in cloudlet c_j added at time slot t. If the number of instances added since the last time slot t_0 exceeds a given threshold Ξ, i.e., $\sum_{l=t_0}^{t} a_{ij}(l) \geq \Xi$, then the predicted number of new instances $\hat{a}_{ij}$ of f_i added at time slot t is

$$\hat{a}_{ij}(t) = \beta_1 a_{ij}(t-1) + \beta_2 a_{ij}(t-2) + \ldots + \beta_k a_{ij}(t-k), \tag{4.3}$$

where Ξ is the given threshold, $\beta_{k'}$ (> 0) is a constant with $0 \leq \beta_{k'} \leq 1$, $\sum_{l=1}^{k} \beta_l = 1$, and $\beta_{k_1} \geq \beta_{k_2}$ if $k_1 < k_2$. Thus, the number of VNF instances of f_i after time slot t installed in cloudlet c_j should be $\hat{a}_{ij}(t) + n_{ij}(t)$.

So far we assumed that there are sufficient resources at each cloudlet to meet the need of creating different VNF instances. However, if there are not enough residual resources at each cloudlet to meet different instance creations, then which VNF instances should we create? To fairly allocate the computing resource for instance creations, we proportionally scale down the number of instances of each different network functions at each cloudlet. In other words, let RC_j be the residual computing resource and DI_j the total computing resource demanded by different instance creations in cloudlet c_j. If $DI_j \leq RC_j$, this implies that all needed numbers of VNF instances can be created; otherwise, let $\mu_j = \frac{RC_j}{DI_j}$ be the ratio, for each requested number of VNF instances, e.g., the total computing resource for creating $a_{ij}(t)$ VNF instances for f_i is $a_{ij}(t)C(f_i)$, then we actually create $a'_{ij}(t) = \lfloor \frac{a_{ij}(t)C(f_i)\cdot\mu_j}{C(f_i)} \rfloor$ VNF instances for f_i at c_j. The details are given in `Algorithm 2`.

5 PERFORMANCE EVALUATION

In this section we evaluate the performance of the proposed algorithms by experimental simulations. We also study the impact of different parameters on algorithmic performance.

5.1 Experimental settings

We assume that a WMAN $G(V, E)$ follows a network topology [7] consisting of 100 APs, where the network is generated using the Barabasi-Albert Model [13], and there are 20 cloudlets randomly deployed in G. Each cloudlet c_j has a computing capacity cap_j within the range from 2,000 to 4,000 MHz [14]. We allow 20 network functions to be available on the cloudlets, where each instance of a network function requires between 40 and 400MHz. We assume that the delay of a link between two APs in the network is between 2 milliseconds (ms) and 5ms [15]. The running time obtained is based on a machine with a 3.4GHz Intel i7-4770 CPU and 16GiB RAM.

Algorithm 2 Admission_Algorithm_Finite_Horizon $(G(t), S(t))$

Require: m cloudlets with each having its available resource capacity $cap_j(t)$, a number of instances of VNFs of each $f_i \in \mathcal{F}$, and a set of requests $S(t)$ at each time slot t for a finite time horizon $1 \le t \le T$, each idle instance of VNF f_i has a cost γ_{ij} and the given cost threshold Ξ.

Ensure: maximize the number requests admitted (i.e., a subset $S'(t) \subseteq S(t)$) for all t during the finite time horizon T with $1 \le t \le T$ while minimizing the total operational cost. For each request r_k, if it is admitted, then to which cloudlet it will be sent and to which instance of VNF it should join/create will be given in the solution.

$cost \leftarrow 0$; $M \leftarrow \emptyset$; /* the total cost of admitted task offloading requests to the system during a period of T, and request assignment M */;
for all t with $1 \le t \le T$ **do**
    /* STAGE one: (a) perform release some occupied resources by idle instances if needed */
    $l_{ij} \leftarrow t_0$ /* The resource release procedure was performed in the last time slot t_0 with $t_0 < t$ */;
    for each cloudlet c_j **do**
        for each $f_i \in \mathcal{F}$ **do**
            if $\sum_{l=t-l_{ij}}^{t} \delta_{ij}(l) \cdot \gamma_{ij} \ge \theta$ **then**
                Predict the number $\hat{n}_{ij}(t)$ of instances of f_i to be kept in c_j by Eq. (4.2);
                Keep $\max\{n'_{ij}(t), \hat{n}_{ij}(t)\}$ instances of f_i in cloudlet c_j;
                Release the occupied resources by the rest $n_{ij}(t) - \max\{n'_{ij}(t), \hat{n}_{ij}(t)\}$ instances of f_i in cloudlet c_j;
                Update the amounts of available resources at cloudlet c_j;
                $l_{ij} \leftarrow t$; /* reset the start time slot of the next idle VNF instances of f_i release in c_j */
    /* STAGE two: (b) increase the number of instances of a network function f_i */;
    $I_{ij} \leftarrow t_0$ /* the number of instances of f_i was increased in the last time slot t_0 with $t_0 < t$ */;
    for each cloudlet c_j **do**
        for each $f_i \in \mathcal{F}$ **do**
            if $\sum_{l=t-I_{ij}}^{t} a_{ij}(l) \cdot \gamma_{ij} \ge \Xi$ **then**
                Predict the number of instances of f_i to be increased $\hat{a}_{ij}$ by Eq. (4.3);
                Let RC_j be the residual computing resource of cloudlet c_j, and DI_j be the total computing resource needed by creating new instances;
                if $RC_j < DI_j$ **then**
                    There will be $\lfloor \frac{RC_j}{DI_j} \cdot (n_{ij}(t) + \hat{a}_{ij}(t)) \rfloor$ instances of f_i in cloudlet c_j at time slot t;
                else
                    There will be $n_{ij}(t) + \hat{a}_{ij}(t)$ instances of f_i in cloudlet c_j at time slot t;
                Update the available resources at cloudlet c_j;
                $I_{ij} \leftarrow t$; /* reset the start time slot of the next VNF instance increase of f_i in cloudlet c_j*/
    /* STAGE two: perform request admissions for the requests in $S(t)$ */
    M_t and $cost_t$ will be returned by applying `Algorithm 1` to $G(t)$;
    $M \leftarrow M \cup M_t$; $cost \leftarrow cost + cost_t$.
return M corresponds to the assignment of requests, while $cost$ is the total admission cost to the system during a period of T;

Unless otherwise stated, the default settings for network parameters will be as follows. The default number of requests per time slot is 1,000, each request has a packet rate between 10 and 80 packets per seconds (similar to the range of application frame rates in typical interactive applications [16]), and a delay bound d_k between 0.2 and 1.2 seconds. The network function requested by each request is randomly selected from the 20 different network functions.

Using the hourly price of a general purpose m3.xlarge Amazon EC2 instance as reference, we assume the operating cost is 0.25 per MHz in each time slot, while the cost of instantiating a new function instance varies between 20 to 50. We assume the cost of transferring a packet between two APs to be proportional to the latency, and so the cost of transferring a packet along a network link varies between 0.002 and 0.005.

We evaluate the proposed algorithms against a greedy baseline which is described as follows. The greedy algorithm assigns each request r_k to the cloudlet with the highest rank in terms of the product of its available number of service chain instances and the inverse of the implementation cost of admitting r_k in the cloudlet. The rationale of this method is to find a cloudlet with high number of available service chain instances and low implementation cost, such that as many as requests are admitted while the implementation cost is minimized. We refer to this highest-rank-first baseline heuristic and the proposed algorithm as HRF and ALG respectively. Each experiment plot is the average of 100 simulation runs.

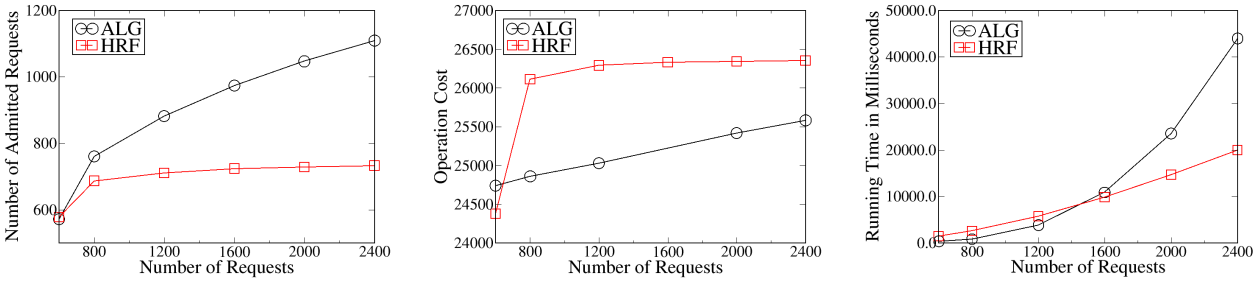
5.2 Algorithm performance within a single time slot

We first investigate the performance of the proposed algorithm ALG and algorithm HRF within a single time slot, by varying the number of requests within the time slot from 600 to 2,400 and creating some instances of each NFV in each cloudlet randomly.

Fig. 5.1 shows the results. From Fig. 5.1 (a), we can see that algorithm ALG admits much more requests than algorithm HRF, while also delivering a lower operation cost, as seen from Fig. 5.1 (b). The reason is as follows.

As both algorithms target requests with the cheapest resource requirements, an increasing number of low cost requests are admitted as we scale the number of initial requests. However, because algorithm ALG matches (assigns) several requests to cloudlets simultaneously, multiple instances of network functions are placed among the cloudlets, spreading the workload when subsequent requests for the network function are admitted. In contrast, as algorithm HRF admits requests one by one, new queues for network functions are instantiating less frequently. As a result, many requests with tight delay tolerances fail to be admitted by a cloudlet that has already instantiated their requested network functions, and thus fewer requests are admitted compared to algorithm ALG.

Initially, when the number of requests is low, algorithm HRF delivers a slightly lower operation cost compared to algorithm ALG, as HRF has instantiated fewer new queues and instances. However, as the number of admitted requests increases, the operation cost delivered by algorithm HRF increases sharply as the algorithm is forced to allocate resources for more expensive requests. The operation cost delivered by algorithm HRF plateaus, as HRF reaches the limit in the number of requests it can admit. In contrast, the growth in operation cost delivered by algorithm ALG is much slower, as the initially higher operation cost from instantiating more queues



(a) The number of requests admitted by different algorithms

(b) Operation cost delivered by different algorithms

(c) Running time of different algorithms in milliseconds

Figure 5.1: Performance of `Algorithm ALG` and `HRF` when the number of requests varies from 600 to 2400, while the number of cloudlets in the network is 20.

and network instances allows subsequent requests be more cheaply admitted to the network.

Fig. 5.1 (c) illustrates the running time of algorithms `ALG` and `HRF` with the growth of the number of user requests. As can be seen, the running time increases dramatically for both algorithms with the number of requests, while the running time of algorithm `ALG` increases at a faster rate than that of algorithm `HRF`.

We next evaluate the performance of algorithms `ALG` and `HRF` within a single time slot, by varying the number of cloudlets between 4 and 24, while creating some instances of each NFV in each cloudlet randomly. Fig. 5.2 shows the result.

From Fig. 5.2. (a) we can see that algorithm `ALG` admits more requests than algorithm `HRF`. Since low cost requests are admitted into the network first, the remaining requests are increasingly expensive and require more cloudlet resources to meet their demands. Due to algorithm `ALG` instantiating multiple network function instances on different cloudlets, requests that have short delay tolerances are more easily admitted, resulting in a higher number of admitted requests compared to algorithm `HRF`.

Fig. 5.2. (b) displays the operation cost of the cloudlets when the number of requests is 1,500, and the number of cloudlets in the network ranges from 4 to 24. Both algorithms have similar plots, and deliver operation costs that have an approximately linear correlation with the number of cloudlets. While algorithm `ALG` delivers a slightly higher operation cost compared to algorithm `HRF` as the number of cloudlets increases, it should be noted that algorithm `ALG` admits significantly higher numbers of requests compared to algorithm `HRF`.

Fig 5.2. (c) illustrates the running time of algorithm `ALG` and algorithm `HRF` as the number of cloudlets in the network increases. The running time of algorithm `HRF` increases linearly with the number of cloudlets. Interestingly the running time of algorithm `ALG` decreases slightly as the number of cloudlets increases. This is because the number of requests admitted in each round of matching is limited to the number of cloudlets. As the number of cloudlets increases, more requests are admitted per round of matching, resulting in fewer rounds of matching and a shorter running time. However it is clear that the change in running time when increasing the number of cloudlets is negligible compared to changes in the number user requests, as the number of requests is orders of magnitude larger than the number of cloudlets.

5.3 Online algorithm performance

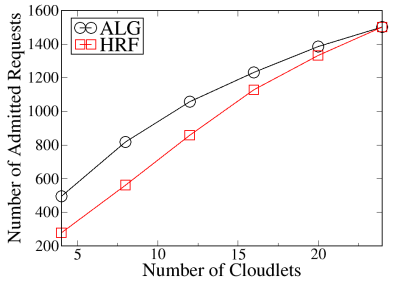
We now consider a time horizon consisting of 100 time slots. The number of requests in each time slot samples the Poisson distribution with a mean of 500, and each admitted request spans 1 to 5 time slots randomly.

Fig. 5.3(a) shows the accumulative number of requests admitted by algorithms `ALG` and `HRF` across the time horizon. We can see that algorithm `ALG` outperforms algorithm `HRF` by an average of 31%, due to more efficient allocation of resources. Fig. 5.3(b) shows the accumulative operation cost delivered by algorithms `ALG` and `HRF` across the time horizon. We can see that algorithm `ALG` has a lower operation cost compared to algorithm `HRF` by an average of 90%.

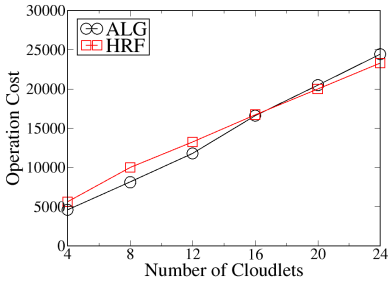
Fig. 5.4 illustrates how the idle cost threshold affects the total number of admitted requests and the total operation cost across a time horizon consisting of 100 slots. Both plots show an increase in total admitted requests and total operation cost with the threshold. A low threshold results in the prediction model being more frequently invoked, and since the cloudlet must maintain at least enough network function instances to handle existing requests, overusing the prediction mechanism can lead to over-provisioning resources to network function instances. As resources are constrained, this restricts the number of requests that can be admitted. As the threshold increases, resources are more efficiently allocated within each time slot, leading to a slight increase in admitted requests. However operation cost also increases with the threshold, as when the prediction model is less frequently used, fluctuation in the number of required instances across time slots are more common and incur additional instantiation costs.

6 CONCLUSIONS

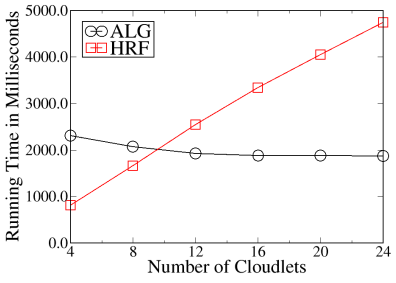
In this paper, we studied a novel task offloading problem in a wireless metropolitan area network, where each offloading task has a maximum tolerable delay and different requests need different types of services from the cloudlets in the network. We focused on maximizing the number of offloading request admissions while minimizing their admission cost within a given time horizon. To this



(a) The number of requests admitted by different algorithms

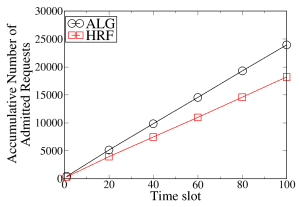


(b) Operation cost delivered by different algorithms

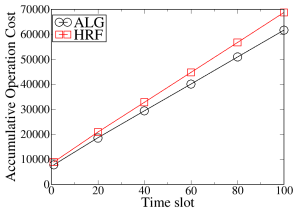


(c) Running time of different algorithms in milliseconds

Figure 5.2: Performance of algorithms ALG and HRF when the number of cloudlets in the network varies from 4 to 24 while the number of requests is fixed at 1,500.

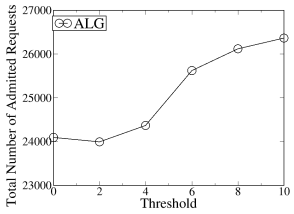


(a) The accumulative number of requests admitted by different algorithms

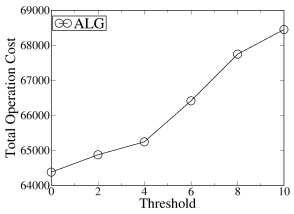


(b) The accumulative operation cost delivered by different algorithms

Figure 5.3: Performance of algorithms ALG and HRF for a time horizon with 100 time slots, where the number of requests in each time slot follows a Poisson distribution with a mean of 500.



(a) The total number of requests admitted by algorithm ALG



(b) The total operation cost delivered by algorithm ALG

Figure 5.4: Threshold impact on the performance of algorithm ALG for a time horizon with 100 time slots.

end, we developed an efficient algorithm for the problem through a novel reduction that reduces the problem to a series of minimum weight maximum matching problems in auxiliary bipartite graphs, and an effective prediction mechanism to predict instance releases and creations in different cloudlets within the network for further cost savings. We finally evaluated the performance of the proposed algorithm through experimental simulations. Experimental results indicate that the proposed algorithm is promising.

REFERENCES

[1] "Facebook f8 developers conference 2017," https://www.fbf8.com/, 2017, accessed: 2017-04-25.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," *Proceedings of the sixth conference on Computer systems*. ACM, 2011.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ACM, 2010.

[4] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," *INFOCOM, 2012 Proceedings IEEE*, IEEE, 2012.

[5] E. Y. Chen and M. Itoh, "Virtual smartphone over ip," *Proc of World of Wireless Mobile and Multimedia Networks (WoWMoM)*, IEEE, 2010.

[6] B. G. Rodrıguez-Santana, A. M. Viveros, B. E. Carvajal-Gámez, and D. C. Trejo-Osorio, "Mobile computation offloading architecture for mobile augmented reality, case study: Visualization of cetacean skeleton," *International Journal of Advanced Computer Science & Applications*, vol. 1, no. 7, pp. 665–671.

[7] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," To appear in *IEEE Transactions on Cloud Computing*, IEEE, 2015.

[8] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *Proc of INFOCOM*, IEEE, 2016.

[9] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Capacitated cloudlet placements in wireless metropolitan area networks," *Proc. of 2015 IEEE 40th Conference on Local Computer Networks (LCN)*, IEEE, 2015.

[10] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, IEEE, 2016.

[11] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," *Proc of 2013 IEEE 38th Conference on Local Computer Networks (LCN)*, IEEE, 2013.

[12] Q. Xia, W. Liang, Z. Xu, and B. Zhou, "Online algorithms for location-aware task offloading in two-tiered mobile cloud environments," *Proc of 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, IEEE, 2014.

[13] R. Albert, H. Jeong, and A.-L. Barabási, "Internet: Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[14] "Hewlett-packard company - enterprise computer server systems and network solutions," https://www.hpe.com/au/en/servers.html/, 2017, accessed: 2017-04-25.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, IEEE, 2011.

[16] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.