QoS-aware Dynamic Fog Service Provisioning

Ashkan Yousefpour, Ashish Patil,
Genya Ishigaki, Jason P. Jue
The University of Texas at Dallas
Email: {ashkan, jjue}@utdallas.edu

Inwoong Kim[†], Xi Wang[†], Hakki C. Cankaya[*],
Qiong Zhang[†], Weisheng Xie[*]
[†]Fujitsu Laboratories of America
[*]Fujitsu Network Communications

***Abstract*—Recent advances in the areas of Internet of Things (IoT), cloud computing and big data have attributed to the rise of growing number of complex and useful applications. As IoT becomes more prevalent in our daily life, more data-intensive, delay-sensitive, and real-time applications are expected to emerge. Ensuring Quality of Service (QoS) in terms of bandwidth and low-latency for these applications is essential, and fog computing has been seen as one of the main enablers for satisfying these QoS requirements. Fog puts compute, storage, and networking resources closer to the user.**

In this paper, we first introduce the Dynamic Fog Service Provisioning problem, which is the problem of dynamically deploying new fog services (applications) on fog nodes, or releasing the services that are already deployed on fog nodes, in order to meet QoS constraints. The problem is formulated as an INLP task, and two heuristics are proposed to solve it efficiently. Finally, the heuristics are evaluated using a simulation based on the real-world traffic traces, and mobile augmented reality as the IoT application.

I. Introduction

The Internet of Things (IoT) is shaping the future of connectivity, processing, and reachability. With IoT, every "thing" is connected to the Internet, so there will be sensors, cameras, computing devices, and actuator as part of our environment, and essentially our life. IoT is proliferating exponentially, and the IoT devices are expected to generate massive amounts of data in short time, that may require immediate processing.

Many IoT applications, such as augmented reality, connected and autonomous cars, drones, industrial robotics, surveillance, and real-time manufacturing have strict latency requirements, in some cases below 10 ms [1][2]. These applications cannot tolerate the large and unpredictable latency of the cloud, when cloud resources are deployed far from where the application data are generated. Fog computing has been recently proposed to bring low latency and reduced bandwidth to IoT networks, by locating the compute, storage, and networking resources closer to the users. According to the OpenFog Consortium, fog can decrease latency and bandwidth usage, reduce costs, and enhance QoS for delay-sensitive applications.

Certain IoT applications are bursty in resource usage, both in space and time dimensions. For instance, in "situation awareness" applications [3], the cameras in the vicinity of an accident generate more requests than the cameras in other parts of the highway (space), while security motion sensors generate more traffic when there is a suspicious activity in the area (time). Thus, the resource usage of the delay-sensitive fog applications is dynamic in time and space [3]. Although cloud computing is seen as the dominant solution for dynamic resource usage patterns, it is difficult for developers to specify application delay constraints while developing software to run in the cloud. Thus, there is a need for dynamic fog application provisioning, to be able to dynamically deploy and release applications on fog nodes.

In the next section, we discuss some of the related research studies, and in Section III we describe the Dynamic Fog Service Provisioning (DFSP) problem. Formulation of the DFSP problem is discussed in Section IV, and two efficient heuristics for solving the DFSP problem are introduced in Section V. Finally, we explain the numerical results of our experiments in Section VI, and present conclusions and suggestions for future research in Section VII.

II. Related Work

Fog service provisioning problems share similar concepts with virtual machine (VM) placement problems in edge networks, such as in [4] and [5]. There are several recent studies on the resource allocation problem for edge/fog computing. The authors in [6] implement a fog computing platform that dynamically pushes software modules to the end devices, while the authors in [7] propose a model for allocation of computing resources on edge networks. In other studies such as [8], the authors introduce the QoS-aware service allocation problem for fog as a basic optimization problem. More recent works in [9] and [10] study a conceptual framework for the service provisioning problem in the fog. The authors, in these simulation studies propose interesting concept of fog cells, that is a software running on IoT nodes for handling services.

There are a few studies that are similar in concept to dynamic fog service provisioning, but have a different goal. Most notably among these are the works on service migration problems in edge clouds, in response to user movement and network performance, such as in [11] and [12]. Some other framework studies, such as [13] and [14], propose platforms and programming models for service provisioning in the fog.

Despite the solid contributions in the aforementioned studies on fog/edge service provisioning, they cannot efficiently scale to the large magnitude of IoT networks, mainly due to their high computation overhead. More specifically, these studies require solving of compute-intensive optimization tasks (in some cases, per request), require knowledge of user mobility patterns or workload of IoT devices, or assume software

requirements for IoT nodes (e.g. virtualization). However, the proposed framework this paper is lightweight and is designed for scalable IoT networks. Moreover, the proposed framework does not make any assumptions about IoT devices, and does not require knowledge about IoT nodes or their mobility. In the rest of the paper, we discuss our proposed DFSP framework.

III. Dynamic Fog Service Provisioning

The main goal of Dynamic Fog Service Provisioning (DFSP) is to provide a better QoS for the clients, in terms of reduced IoT service delay and reduced fog-to-cloud bandwidth usage, while minimizing the resource usage of the Cloud Service Provider (CSP). More specifically, DFSP is about dynamically provisioning (deploying or releasing) IoT applications on the fog nodes, to comply with the QoS terms (delay threshold, penalties, etc.) defined in the Service Level Agreement (SLA) between the CSP and its clients, with minimal resource cost for the CSP. We refer to the CSP as a Fog Service Provider (FSP), since it is able to deploy services in the fog. In this framework, the client is defined as IoT application developer or the entity who owns the IoT nodes and signs an SLA with the FSP. We focus in this paper on the average service delay as the QoS measure of IoT nodes.

A. Fog Entities Interaction Model

When a client signs a contract with the FSP, the FSP guarantees to provide fog services (using DFSP) with delays below a maximum delay threshold th. The FSP also provides a level of the desired QoS q, which governs how strict the delay requirements are. For example, for a given service a, if $q_a = 99\%$ and $th_a = 5$ ms, it means that the client requires that the service delay of application a on the corresponding IoT nodes must be less than 5 ms, 99% of the time.

One trivial solution for the FSP is to deploy the particular service on all the fog nodes close to the client's IoT nodes. Nevertheless, this is not an efficient solution, because it over-provisions resources for that particular service. If the FSP has many clients, it is likely that it does not have adequate resources on fog nodes for all the clients' services, whether it owns the fog nodes, or it rents them from an Edge Network Provider. Moreover, blindly deploying all the services on fog nodes will waste the available resources. Therefore, the FSP should employ a dynamic approach (e.g. DFSP) to provision its services, while minimizing the cost and complying with the QoS terms of the SLA of its clients. In Section IV we explain how to solve the DFSP problem to achieve the above goals.

B. System Architecture

The general system architecture is shown in Fig. 1. IoT nodes send requests to the cloud. Some of the requests are for traditional cloud-based applications, which will be sent directly to the cloud, without any processing in the fog. The IoT requests that are for fog applications will either be processed by fog nodes, if the corresponding service is deployed on the fog nodes, or forwarded to the cloud, if the service is not deployed on fog nodes. Fog nodes may also offload requests to other fog nodes, such as in [15], to balance their workload. The traffic rate of IoT requests is monitored by a traffic monitor agent, such as an SDN controller. The "Fog Service Controller" (FSC) uses the monitored incoming traffic rate to the fog nodes to decide when is it necessary to deploy or release a service, based on the demand for a particular service. The FSC solves an instance of the DFSP problem periodically to make decisions to deploy or release services, while minimizing the FSP's cost. The DFSP problem is discussed in detail in Section IV. It is assumed that the FSC only manages fog nodes in a particular geographical location.

C. Fog Service Controller (FSC)

The FSC (may be replicated for different geographical areas) solves an instance of the DFSP problem periodically. The FSC has a database (labeled "Service Database") of the services (collection of application containers) that the FSP's clients develop, and also maintains the SLA parameters of each service (Fig. 1). The FSC uses the Service Database to deploy required services on the fog nodes. Using SLA parameters, the FSC can obtain the delay and QoS requirements of a service. When the traffic demand for a particular service increases, the FSC may deploy a new service on the corresponding fog nodes, to reduce the IoT service delay. On the other hand, if demand for a particular service is not significant, the FSC may release the service, to save on resource cost. Both the deploy and the release operations are performed as per the QoS requirements in the SLA. For instance, when the traffic demand is low but the QoS requirement of a service is strict, the service may not be released.

D. Application Development and Deployment

When the FSP's clients develop new applications and push them to the cloud, the applications are automatically pulled into the FSC's Service Database. This ensures that the Service Database always has the newest version of the applications. This deployment model hides the platform heterogeneity, location of fog nodes, and complexities of migration process of the applications from the application developers. Note that traditional cloud-based applications that do not require low delay of fog need not be pulled into the Service Database.

E. Fog Node Architecture

Fog nodes can be routers, access points, switches, gateways, firewalls, or dedicated servers for fog computing. Fog nodes could be SDN-enabled devices, such as switches or routers, or could be directly connected to SDN-enabled devices. In this case, the incoming traffic rate to fog nodes could be monitored by the SDN controller. The FSC communicates with the SDN controller through the SDN controller's northbound API, to obtain the fog nodes' incoming traffic information.

As a second option, if fog nodes are not SDN-enabled devices nor connected to such devices, they can independently monitor their incoming traffic and report the traffic rate to the FSC. In this case, fog nodes will need to have a traffic monitoring agent running on them, which reports the incoming

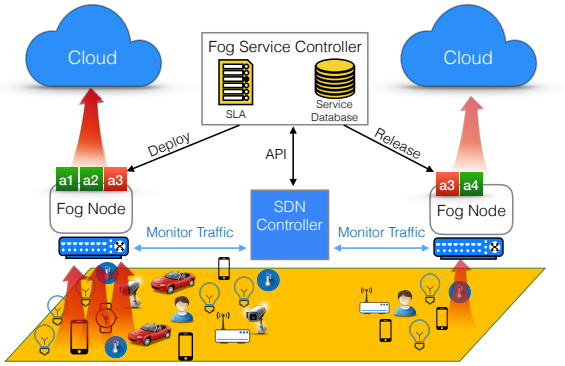


Fig. 1. Dynamic Fog Service Provisioning Architecture. Only traffic from IoT to cloud is shown.

traffic rate for fog services to the FSC. For simplicity of the presentation, throughout the paper we assume the first option (SDN controller) is employed.

Fog nodes are assumed to run some container orchestration software, such as Docker, Kubernetes, or OpenStack, which automates the deployment and release of service containers. The reason for using containers instead traditional VMs is that containers are light-weight in comparison to VMs and provide lower set-up delay, as they share the host OS [16][6]. When an application is deployed on a fog node, it advertises the fog node's IP address to the corresponding IoT nodes that run the application, so that the IoT node's requests are sent to the fog node (fog discovery). It is worth mentioning that if DFSP is not used by FSPs, services must be permanently deployed either in the cloud, which results in unacceptable delays, or on fog nodes, which results in loss of revenue if resource usage is not planned intelligently. DFSP allows the FSP to dynamically adopt to the variable traffic demand.

IV. DFSP Problem Formulation

In this section, we define, formulate and discuss the DFSP Problem.

Definition 1. *Dynamic Fog Service Provisioning (DFSP)* is an online optimization task that aims to dynamically provision (deploy or release) services on fog nodes, to comply with the QoS terms of the SLA, while minimizing the cost of resources.

A. Notation and Variables

To formulate the DFSP problem, we introduce some notation. All notation is summarized in Table I. Let a set of fog nodes be denoted by F, a set of cloud servers by C, and a set of fog services (applications) by A. A service can be implemented using a collection of containers. Let the desired QoS level for service a be denoted by $q_a \in (0,1)$, and delay threshold for service a by th_a.

The physical network, set of fog nodes and cloud servers are modeled as a graph $G = (V, E)$, such that the node set V includes the fog nodes F and cloud servers C ($V = F \cup C$), and the edge set E includes the logical links between the nodes. Each edge $e(src, dst) \in E$ is associated with three numbers: u_e, the communication cost of logical link e per unit bandwidth per unit time; r_e, the transmission rate of logical link e; and d_e, the propagation delay of logical link e.

The main decision variables of the DFSP problem are the

TABLE I
TABLE OF NOTATION

F	set of fog nodes
C	set of cloud servers
A	set of fog services/applications
I_a	set of IoT nodes running service a
Φ	Fog Service Controller (FSC)
q_a	desired quality of service for service a; $q_a \in (0,1)$
th_a	delay threshold for service a
d_{aj}	average service delay of IoT nodes connected to fog node j for service a
p_a	penalty that FSP must pay if the delay of service a is violated as per SLA
u_e	communication cost per unit bandwidth of link $e = (i,j)$
r_e	transmission rate (bandwidth) of link $e = (i,j)$
d_e	propagation delay of link $e = (i,j)$
τ	time interval between two instances of solving the optimization problem, in seconds
C_j^P	unit cost of process. at fog node j (per million instructions)
$C_k'^P$	unit cost of process. at cloud ser. k (per million instructions)
C_j^S	unit cost of storage at fog node j (per byte per second)
$C_k'^S$	unit cost of storage at cloud server k (per byte per second)
L_a^S	storage size of service a (i.e. its constituent containers), in bytes
L_a^P	required amount of processing for service a per request, in million instructions per request
L_a^M	required amount of memory for service a, in bytes
$\lambda_{aj}^{\text{out}}$	rate of dispatched traffic for service a from fog node j to the associated cloud server (request/second)
λ_{aj}^{in}	incoming traffic rate from IoT nodes to fog node j for service a (request/second)
$\lambda_{ak}'^{\text{in}}$	incoming traffic rate to cloud server k for service a (request/second)
$\lambda_{ajj'}$	traffic rate of service a from fog node j to fog node j' (request/second)
$h(j)$	maps fog node j to the associated cloud node k
$h^{-1}(k)$	set of fog nodes j that send their traffic to cloud server k (associated fog nodes to cloud server k)
K_j^S	storage capacity of fog node j, in bytes
K_j^P	processing capacity (service rate) of fog node j, in MIPS
K_j^M	memory capacity of fog node j, in bytes
$K_k'^S$	storage capacity of cloud server k, in bytes
$K_k'^P$	processing capacity (service rate) of cloud server k, in MIPS
$K_k'^M$	memory capacity of cloud server k, in bytes
l_a^{rq}	average size of requests of service a, in bytes
l_a^{rp}	average size of responses of service a, in bytes
d_{aj}^{PF}	waiting time (processing delay plus queueing delay) for requests of service a at fog node j
d_{ak}^{PC}	waiting time (processing delay plus queueing delay) for requests of service a at cloud server k
x_{aj}	binary variable showing if service a is hosted on fog node j
x'_{ak}	binary variable showing if service a is hosted on cloud server k
v_{aj}	binary variable showing if average service delay of IoT nodes connected to fog node j for service a is greater than th_a
$V_a^{\%}$	the percentage of IoT service delay samples of service a that do not meet the delay requirement.

placement binary variables, defined below:

$$x_{aj} = \begin{cases} 1, & \text{if service } a \text{ is hosted on fog node } j, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$x'_{ak} = \begin{cases} 1, & \text{if service } a \text{ is hosted on cloud server } k, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Incidentally, we denote by x_{aj}^{cur}, the current configuration of the service a on fog node j, which can be regarded as an input to the optimization problem to find the future placement of fog nodes.

To formulate cost, we need to introduce some cost and traffic parameters. C_j^P and $C_k'^P$ indicate the unit cost of processing at fog node j and cloud server k, respectively; C_j^S and $C_k'^S$ are the unit cost of storage per unit time at fog node j and cloud server k, respectively. L_a^S represents the storage size of service a, L_a^M is the amount of required memory for service a, and L_a^P represents the amount of required processing for service a per request. For traffic, let λ_{aj}^{in} and $\lambda_{ak}'^{\text{in}}$ denote the incoming traffic rate for service a to fog node j and cloud server k, respectively; and let $\lambda_{ajj'}$ be the traffic rate of service a offloaded from fog node j to fog node j'. Lastly, $\lambda_{aj}^{\text{out}}$ denotes the rate of dispatched traffic of service a from fog node j to its associated cloud server k ($k = h(j)$, where function $h(.)$ maps a given fog node to the id of associated cloud server). The unit of all the traffic rates is in requests per second.

B. Objective

The DFSP problem can be formulated as problem **P1**:

$$\textbf{P1}: \text{Minimize } (C_C^{\text{proc}} + C_F^{\text{proc}}) + (C_C^{\text{stor}} + C_F^{\text{stor}}) + (C_{FC}^{\text{comm}} + C_{FF}^{\text{comm}} + C_{\Phi F}^{\text{depl}})$$
$$\text{Subject to QoS constraints.}$$

The cost components are defined below.

$$C_C^{\text{proc}} = \sum_{k \in C} \sum_{a \in A} C_k'^P L_a^P \lambda_{ak}'^{\text{in}} x'_{ak} \tau, \quad (3)$$

$$C_F^{\text{proc}} = \sum_{j \in F} \sum_{a \in A} C_j^P L_a^P \lambda_{aj}^{\text{in}} x_{aj} \tau, \quad (4)$$

$$C_C^{\text{stor}} = \sum_{k \in C} \sum_{a \in A} C_k'^S L_a^S x'_{ak} \tau, \quad (5)$$

$$C_F^{\text{stor}} = \sum_{j \in F} \sum_{a \in A} C_j^S L_a^S x_{aj} \tau, \quad (6)$$

$$C_{FC}^{\text{comm}} = \sum_{j \in F} \sum_{a \in A} u_{(j,h(j))} \lambda_{aj}^{\text{out}} (l_a^{rq} + l_a^{rp}) \tau, \quad (7)$$

$$C_{FF}^{\text{comm}} = \sum_{j \in F} \sum_{j' \in F} \sum_{a \in A} u_{(j,j')} \lambda_{ajj'} l_a^{rq} \tau, \quad (8)$$

$$C_{\Phi F}^{\text{depl}} = \sum_{j \in F} \sum_{a \in A} u_{(\Phi,j)} (1 - x_{aj}^{\text{cur}}) x_{aj} L_a^S. \quad (9)$$

C_C^{proc} and C_F^{proc} are cost of processing in cloud and fog, respectively; C_C^{stor} and C_F^{stor} are cost of storage in cloud and fog, respectively. C_{FC}^{comm} is the cost of communication between fog and cloud, C_{FF}^{comm} is the cost of communication between fog nodes, and $C_{\Phi F}^{\text{depl}}$ is the communication cost of service deployment, from the FSC Φ to fog nodes. A service deployed on a fog node may be released when the demand for the application is small. Therefore, we assume the services do not store any state information on fog nodes [2], and we do not consider costs for state migrations. Moreover, the deployment cost of a service in the cloud is not considered in our optimization problem, as this cost is negligible for the FSP. τ is the time interval between two instances of solving the optimization problem.

C. Constraints

1) Service Delay: Service delay is defined as the time interval between the moment when an IoT node sends a service request and when it receives the response for that request. In order to measure service delay at the fog, we will use average values for the delay between IoT nodes and fog nodes, so that the equation for service delay does not include indices of IoT nodes.

The average service delay of IoT nodes connected to fog node j for service a is

$$d_{aj} = (2d_{(I_a,j)} + d_{aj}^{PF} + \frac{l_a^{rq} + l_a^{rp}}{r_{(I_a,j)}}) x_{aj} + (2(d_{(I_a,j)} + d_{(j,k)}) + d_{ak}^{PC} + (\frac{l_a^{rq} + l_a^{rp}}{r_{(I_a,j)}} + \frac{l_a^{rq} + l_a^{rp}}{r_{(j,k)}}))(1 - x_{aj}) \quad (10)$$

where $k = h(j)$. I_a is the set of IoT nodes implementing service a. $d_{(I_a,j)}$ and $r_{(I_a,j)}$ are the average propagation delay and average transmission rate of the logical links between IoT nodes in I_a and fog node j, respectively, and they are given as inputs to the DFSP problem. We claim that using average values is reasonable, because, firstly, fog nodes are usually placed near IoT nodes, which means IoT nodes connected to the same fog node have similar values of propagation delay and transmission rate. Secondly, if exact values were to be calculated, the FSC would need to have information about all the IoT nodes communicating with the fog nodes, which may not be practical.

2) SLA Violation: To measure quality of a given service a, we need to see what percentage of IoT requests do not meet the delay threshold th_a (SLA violations). We first need to check if average service delay of IoT nodes connected to fog node j for service a (labeled as d_{aj}) is greater than the threshold th_a defined in SLA for service a. Let us define a binary variable v_{aj} to indicate this:

$$v_{aj} = \begin{cases} 1, & \text{if } d_{aj} > th_a \\ 0, & \text{otherwise} \end{cases}. \quad (11)$$

We define another variable that measures the SLA violation (SLAV) of a given service according to the defined QoS parameters in the SLA. Recall that the QoS requirement in the SLA is that the percentage of delay samples from IoT nodes that exceed the delay threshold should be no more than

$1-q_a$. We denote by $V_a^{\%}$ the percentage of IoT service delay samples of service a that do not meet the delay requirement. $V_a^{\%}$ can be calculated as follows

$$V_a^{\%} = \frac{\sum_j \lambda_{aj}^{\text{in}} v_{aj}}{\sum_j \lambda_{aj}^{\text{in}}}. \tag{12}$$

Note that $V_a^{\%}$ is measured by the FSC, as a weighted average of v_{aj}, with λ_{aj}^{in} as the weight. In practice, the FSC can obtain λ_{aj}^{in}, the rate of incoming traffic to fog nodes, from the traffic monitor agent, and find the percentage of traffic for service a at different fog nodes.

We can now add the cost of SLA violations to **P1**. Let p_a denote the penalty that the FSP must pay if delay samples are violated by 1% per unit time. For instance if $q_a = 97\%$, any violation percentage $V_a^{\%}$ greater than 3% must be paid for by FSP. The total cost of SLAV for FSP will be

$$C^{viol} = \sum_a \max(0, V_a^{\%} - (1-q_a)) \times p_a \tau. \tag{13}$$

Once we have discussed all the constraints of **P1**, we will add C^{viol} to **P1** and rewrite the problem.

3) Resource Capacity: Resource utilization of fog nodes and cloud servers shall not exceed their capacity.

$$\sum_{a\in A} x_{aj} L_a^S < K_j^S, \text{and } \sum_{a\in A} x_{aj} L_a^M < K_j^M, \quad \forall j \in F, \tag{14}$$

$$\sum_{a\in A} x'_{ak} L_a^S < K_k'^S, \text{and } \sum_{a\in A} x'_{ak} L_a^M < K_k'^M, \quad \forall k \in C. \tag{15}$$

4) Traffic Rates: In order to accurately evaluate the waiting times of fog nodes and cloud servers in our delay model, we need to account for the different processing times of the IoT requests. In other words, we need to consider a larger weight for the requests with big processing times than those with smaller processing times. Therefore, for delay measurements, we express the incoming traffic rates to the fog nodes in the units of instructions per unit time. This further allows us to model the fog nodes as M/M/1 queueing systems.

Let Λ_j^{F} denote the arrival rate of instructions (in MIPS) to fog node j. This is the arrival rate of instructions of the incoming requests that are accepted for processing by fog node j, since a service is available on fog node j for the requests. Fog node j needs to be stable, that is $\Lambda_j^{\text{F}} < K_j^P$, or

$$\Lambda_j^{\text{F}} = \sum_{a\in A} L_a^P \lambda_{aj}^{\text{in}} x_{aj} < K_j^P, \quad \forall j \in F. \tag{16}$$

If service $\tilde{a}$ is not implemented on fog node j, the incoming traffic denoted by the rate $\lambda_{\tilde{a}j}^{\text{in}}$ will not be accepted by fog node j, and hence will not be counted in Λ_j^{F}. In that case, the traffic will be sent to the cloud. The rate of this "rejected" traffic is denoted by

$$\lambda_{aj}^{\text{out}} = \lambda_{aj}^{\text{in}}(1 - x_{aj}), \qquad \forall j \in F, \forall a \in A. \tag{17}$$

Let $\lambda_{ak}'^{\text{in}}$ denote the incoming traffic rate to cloud server k for service a. Then we will have $\lambda_{ak}'^{\text{in}} = \sum_{j\in h^{-1}(k)} \lambda_{aj}^{\text{out}}$, where $h^{-1}(k)$ is a set of fog nodes j that send their traffic to cloud server k. Similar to Eq. (16), the arrival rate of instructions (in MIPS) to the cloud server k, Λ_k^C, can be obtained as

$$\Lambda_k^C = \sum_a L_a^P \lambda_{ak}'^{\text{in}} x'_{ak} < K_k'^P, \quad \forall k \in W. \tag{18}$$

5) Variable Limits: If incoming traffic rate to a cloud server for a particular service is 0, the service could be released to save space. On the other hand, even if there is small traffic incoming to a cloud server for a particular service, the service must not be removed from the cloud server, since the cloud server is the last resort for the requests of that service. The following constraint guarantees the above statement.

$$\frac{\lambda_{ak}'^{\text{in}}}{W} \le x'_{ak} \le W \lambda_{ak}'^{\text{in}}, \qquad \forall k \in C, \forall a \in A, \tag{19}$$

where W is an arbitrary large number ($W > \max_{a,k}(\lambda_{ak}'^{\text{in}})$).

The following are other limit constraints for the decision binary variables:

$$0 \le \sum_{a\in A} x_{aj} \le |A|, \qquad \forall j \in F, \tag{20}$$

$$0 \le \sum_{j\in F} x_{aj} \le |F|, \qquad \forall a \in A, \tag{21}$$

$$0 \le \sum_{a\in A} x'_{ak} \le |A|, \qquad \forall k \in C, \tag{22}$$

$$1 \le \sum_{k\in C} x'_{ak} \le |C|, \qquad \forall a \in A. \tag{23}$$

The last equation has 1 on the left side of the inequality, since a service has to be placed on at least one cloud server.

6) Waiting Times: We have all the components of Eq. (10), except for the waiting times of fog nodes and cloud servers (d_{aj}^{PF} and d_{ak}^{PC}). We adopt a commonly used M/M/1 queueing system model [17] for fog nodes with arrival rate of Λ_j^{F} and service rate K_j^P. Hence,

$$d_{aj}^{PF} = \frac{1}{K_j^P - \Lambda_j^{\text{F}}}, \qquad \forall a \in A, \forall j \in F. \tag{24}$$

Note that the requests for different services have different processing times. Nevertheless, as discussed before, in the definition of Λ_j^{F} we account for the different processing times by inclusion of L_a^P.

Similarly, cloud server k with arrival rate Λ_k^C and service rate K'^P can be seen as an M/M/1 queueing system, therefore d_{ak}^{PC} could be derived as

$$d_{ak}^{PC} = \frac{1}{K_k'^P - \Lambda_k^C}, \qquad \forall a \in A, \forall k \in W. \tag{25}$$

D. Optimization Problem

P1 can be rewritten as **P2** with the same constraints:

$$\begin{aligned}\textbf{P2}: \text{Minimize } & (C_C^{\text{proc}} + C_F^{\text{proc}}) + (C_C^{\text{stor}} + C_F^{\text{stor}}) + \\ & (C_{FC}^{\text{comm}} + C_{FF}^{\text{comm}} + C_{\Phi F}^{\text{depl}}) + (C^{viol}) \\ \text{Subject to } & (1-2)(14-25).\end{aligned}$$

Note that in this optimization problem, we only consider the costs of fog-to-fog and fog-to-cloud communication. In other

words, the cost of communication between IoT and fog is not considered in the optimization problem, since this is usually outside the control of FSP.

The DFSP problem is an Integer Nonlinear Programming (INLP) task, due to equations (3), (11), (24), and (25). Since the complexity of INLP solvers is high, we will propose two heuristics for solving the DFSP problem in the next section.

V. Heuristics for DFSP problem

In this section, we describe our proposed heuristics for efficiently solving the DFSP problem. We propose two heuristics: Min-Viol heuristic, which aims to minimize the SLA violations, and Min-Cost heuristic, whose goal is minimizing the total cost. The Min-Viol and Min-Cost heuristics are shown in Algorithm 1 and Algorithm 2, respectively. Both heuristics will be periodically called by the FSC every τ seconds. The proposed heuristics are discussed in more detail in the following subsection.

Algorithm 1 DFSP Min-Viol Heuristic

Input: Service a size stats, G, q_a, th_a, λ^{in}_{aj}, cost parameters
Output: Placement of service a that minimizes SLA violation
Assumption: An instance of service a is running in cloud
1: Read service a's incoming traffic rate to fog nodes
2: List $L \leftarrow$ sort fog nodes in descending order of traffic rate
3: CALCVIOLPERC(x_{aj}, λ^{in}_{aj}, th_a) ▷ $V^{\%}_a$ updated
4: **while** $V^{\%}_a > 1 - q_a$ **do** ▷ Deploy
5: j = get from the head of L a fog node j, on which service a is not deployed
6: **if** fog node j has enough storage and memory **then**
7: Deploy service a on fog node j ▷ $x_{aj} = 1$
8: CALCVIOLPERC(x_{aj}, λ^{in}_{aj}, th_a) ▷ $V^{\%}_a$ updated
9: **end if**
10: **end while**
11: *canRelease* = true
12: **while** canRelease **do** ▷ Release
13: j = get from the tail of L a fog node j, on which service a is deployed
14: Set $x_{aj} = 0$
15: CALCVIOLPERC(x_{aj}, λ^{in}_{aj}, th_a) ▷ $V^{\%}_a$ updated
16: **if** $V^{\%}_a \leq 1 - q_a$ **then**
17: Release service a on fog node j ▷ $x_{aj} = 0$
18: **else** ▷ if releasing would cause violation
19: Set $x_{aj} = 1$, and *canRelease* = false ▷ exit
20: **end if**
21: **end while**
22: **return** x_{aj}

A. Description

1) Min-Viol: The high-level rationale behind the Min-Viol heuristic is to deploy services on the fog nodes with high traffic rates, and to release services on the fog nodes with low traffic rates, while keeping the violation low, as per the terms of the SLA. First, the incoming traffic rates to the fog nodes are read from the traffic monitor agent (e.g. SDN controller), based on which the fog nodes are sorted in descending order (lines 1-2). Next, using the method CALCVIOLPERC(.) which uses Eq. (12), the percentage of the violating IoT requests for service a ($V^{\%}_a$) is calculated. As long as this percentage is more than $1 - q_a$ (lines 4-10), the heuristic keeps deploying services on the fog nodes, and once $V^{\%}_a \leq 1 - q_a$, it exits the loop. Note that the services are deployed first on the fog nodes with higher incoming traffic rate, so that $V^{\%}_a$ decreases faster.

When $V^{\%}_a \leq 1 - q_a$, we might still be able to release services on the fog nodes with small incoming traffic rate, without violating QoS requirements in the SLA. The second loop (lines 12-21) releases services on the fog nodes with smaller incoming traffic rate. First, we try to find a fog node with small incoming traffic rate (from the end of sorted list L) and check if releasing the already-deployed service would cause any SLA violation (lines 14-15). If releasing does not cause violation (line 16), we go ahead and release the service (line 17); otherwise, we do not release the service and exit the algorithm (line 19), because releasing service on the next fog nodes with higher incoming traffic would cause even more SLA violation. Min-Viol requires service a's average size statistics (i.e. service storage size, size of request and reply, amount of processing), graph G, q_a, and th_a as input.

2) Min-Cost: The main idea of the Min-Cost heuristic is similar to that of the Min-Viol heuristic; however, the major concern in the Min-Cost heuristic is minimizing the cost. The Min-Cost heuristic tries to minimize the cost by checking if deploying or releasing services will increase the revenue (or equally will decrease cost). Similar to the Min-Viol heuristic, the incoming traffic rates to fog nodes are read from the traffic monitor agent (e.g. SDN controller), based on which the fog nodes are sorted in descending order (lines 1-2). Next, we iterate through fog nodes and check if deploying a service make sense, in terms of minimizing the cost (lines 3-9). Line 5 checks if the cost savings of deploying service a on fog node j, is larger than the expenses (or losses), when the service a is not deployed on fog node j. The cost savings of deploying a service are due to the reduced cost of communication between fog and cloud, the reduced costs of storage and processing in the cloud, and the (possible) reduced cost of SLA violations, when the service a is implemented on fog node j. Conversely, the expenses are the cost of service deployment (communication cost), and the increased costs of storage and processing in the fog. All the mentioned costs are calculated and compared for the duration of one interval (τ). Similar to deploying (lines 3-9), for releasing, we iterate through fog nodes, in increasing order of their incoming traffic rates and check if releasing a service make sense, in terms of minimizing the cost (lines 10-14). Likewise, line 11 checks if the cost savings of releasing service a on fog node j, is greater than the expenses, when the service a is deployed on fog node j. The cost savings of releasing a service are due to the reduced costs of storage and processing in the fog when the service is released, while the expenses are due to the increased cost of communication between fog and cloud, the increased

Algorithm 2 DFSP Min-Cost Heuristic

Input: Service a size stats, G, q_a, th_a, λ^{in}_{aj}, cost parameters
Output: Placement of service a that minimizes cost

Assumption: An instance of service a is running in cloud
1: Read service a's incoming traffic rate to fog nodes
2: List $L \leftarrow$ sort fog nodes in descending order of traffic rate
3: **for all** fog node j in L **do** ▷ Deploy
4: **if** fog node j has enough storage and memory **then**
5: **if** cost savings of deploying a on $j >$ expenses of deploying service a on j **then**
6: Deploy service a on fog node j ▷ $x_{aj} = 1$
7: **end if**
8: **end if**
9: **end for**
10: **for all** fog node j in L(reverse) **do** ▷ Release
11: **if** cost savings of releasing a on $j >$ expenses of releasing service a on j **then**
12: Release service a on fog node j ▷ $x_{aj} = 0$
13: **end if**
14: **end for**
15: **return** x_{aj}

costs of storage and processing in the cloud, and the (possible) increased cost of SLA violations. Min-Cost's inputs are similar to those of Min-Viol; additionally, Min-Cost requires the unit cost parameters (processing, storage, and communication) to calculate the cost using equations (3)-(9).

B. Complexity

The time complexity of the proposed heuristics shown in Algorithm 1 and Algorithm 2 are discussed below.

1) Min-Viol: The asymptotic complexity of Line 2 is $O(|F| \log |F|)$ due to sorting of fog nodes; steps in lines 4-10 run at most $|F|$ times, and since the complexity of Algorithm 2 is $O(|F|)$, lines 4-10 take $O(|F|^2)$; similarly, the complexity of lines 12-21 is $O(|F|^2)$. Therefore, the overall complexity of Algorithm 1 is $O(|F|^2)$. Since the FSP runs Algorithm 1 for all fog services, the resulting complexity of solving the DFSP problem using the Min-Viol heuristic will be $O(|A||F|^2)$.

2) Min-Viol: Similar to above, the asymptotic complexity of Line 2 is $O(|F| \log |F|)$. The rest of the lines take $O(|F|)$, and hence the resulting complexity of solving the DFSP problem using the Min-Viol heuristic will be $O(|A||F| \log |F|)$.

VI. Numerical Evaluation

In this section, we discuss the settings and results of the numerical evaluation of the two proposed heuristics.

A. Simulation Settings

The evaluation is done using a Java program that simulates a network of fog nodes, IoT nodes, and cloud servers. The logical topology consists of 3 cloud servers, 10 fog nodes, and 20 services. Most parameters of the simulation are summarized in Table II, while the rest are explained here ($U(a, b)$ indicates a random uniform distribution between a and b).

The propagation delay can be estimated by halving the round-trip time of small packets; and as evaluated in our previous study [15], it is assumed to be $U(1, 2)$ ms between the IoT nodes and the fog nodes, and $U(15, 35)$ ms between the fog nodes and the cloud servers. The transmission medium of the IoT nodes and the fog nodes is assumed to be either WiFi (one hop), or WiFi and a 1 Gbps Ethernet link (two hops). The fog nodes and the cloud servers are assumed to be 6-10 hops apart, while their communication path consists of 10 Gbps and 100 Gbps (up to 2) links. The transmission rates of the links between the FSC and the fog nodes are assumed to be 10 Gbps.

In order to evaluate our heuristics and obtain realistic results for the simulation, we have employed real-world traffic traces, taken from MAWI Working Group traffic archive [18]. MAWI's traffic archive is maintained by daily trace captures at the transit link of WIDE to their upstream ISP. We have used the traces of *2017/04/12-13* in this paper for modeling the incoming traffic rates to fog nodes from IoT nodes.

To model the cloud, we have chosen a particular class B subnet as the IP addresses of the FSP's cloud servers. For modeling the fog nodes, we have selected 10 cities from the traffic trace, and assumed that there is one fog node in each city (which can later serve the IoT requests coming from the city where the fog node resides). We have only used the TCP and UDP packets in the traffic trace to account for IoT requests. In other words, the tuple <IP address, port number> (destination) represents a particular fog/cloud service, to which IoT requests are sent. To model the traffic rates of the fog nodes, the rate of the mentioned TCP/UDP traffic trace is further normalized in a way that the fog nodes' queues are stable.

In the simulation, we assume that there is enough memory on fog nodes and the cloud servers for processing the IoT requests. The processing capacity of each fog node, K^P_j, is $U(800, 1300)$ MIPS [19], and the processing capacity of each cloud server, K'^P_k, is assumed to be 20 times that of the fog nodes. The storage capacity of fog nodes and cloud servers, K^S_j and K'^S_k, are only assumed to be more than 10 GB, to host at most 20 services of the maximum size.

We have considered mobile augmented reality as the service running on IoT nodes, which has a delay threshold of 10 ms [1]. The average size of request and response of the mobile augmented reality applications are taken as $U(10, 26)$ KB and $U(10, 20)$ B, respectively, according to [20], and the required amount of processing for the services is $U(50, 500)$ million instructions (MI) per request [10]. The start-up delay of the service containers is set at 50 ms [16]; however, it does not notably affect the results, as the interval of monitoring traffic and running the heuristics for deploying services in a real-world setting will be in the order of tens of seconds to minutes.

B. Results

The results of the simulation are shown in Fig. 2. The figures on the left column of Fig. 2 are obtained using the 48 hours of trace data of *2017/04/12-13*, while the figures on the right

TABLE II
SIMULATION PARAMETERS

q_a	$U(90, 99.999)\%$	th_a	10 ms
u_e	0.0002 per Mb	p_a	$U(1, 2)$ per % per sec
link rate (core)	10 Gbps, 100 Gbps	L_a^P	$U(50, 200)$ MI per req
link rate (edge)	54 Mbps, 1 Gbps	L_a^S	$U(50, 500)$ MB
$C_j^P, C_k'^P$	0.01 per MI	K_j^P	$U(800, 1300)$ MIPS
$C_j^S, C_k'^S$	0.04 per Gb per sec	l_a^{rq}	$U(10, 26)$ KB
K_j^S, $K_k'^S$	$\geq$10 GB	l_a^{rp}	$U(10, 20)$ B

column are obtained using 4 hours (12:00PM-4:00PM) of trace data of *2017/04/12*. In the simulation with the 48-hour traffic trace, both the interval of traffic change and τ are 15 minutes. In the simulation with the 4-hour traffic trace, the interval of traffic change is 1 minute, and τ is 3 minutes. In all figures, the label "All Cloud" indicates a setting where the IoT requests are sent directly to the cloud (that is, fog nodes do not process the IoT requests). "Min-Viol" and "Min-Cost" represent the two proposed heuristics, and "Static Fog" is a technique where the services are deployed statically at the beginning, instead of dynamically deploying them. Static Fog uses the Min-Cost heuristic, and finds a one-time placement of the fog services at the beginning, using the average traffic rates of the fog nodes as the input.

The figures on the first row represent the normalized incoming traffic rates to the fog nodes. The figures on the second row demonstrate the average service delay of the IoT nodes. Being far from the IoT nodes, All Cloud results in the highest service delay, and the Min-Viol heuristic achieves the lowest delay, as its goal is to minimize the violation. Min-Cost has the next lowest service delay in the 48-hour trace, and a comparable service delay to the Static Fog in the 4-hour trace.

The third row of figures shows the average cost of the various settings over time. All Cloud has the highest cost, because it cannot satisfy the QoS requirement of the low delay requirements of the mobile augmented reality applications, which results in a service penalty. Being designed for minimizing cost, the Min-Cost heuristic achieves the lowest cost, interestingly even when its average service delay is more than that of the Min-Viol heuristic; the Min-Cost heuristic is a cost-minimizing heuristic for FSPs that minimizes the cost, sometimes at the cost of violating the agreed QoS to the customer. On average, the Static Fog's cost is more than that of Min-Cost, as its static placement cannot keep up with the changing traffic demand. It can be seen that when the traffic rates are not high (figures on the left column), Min-Viol's cost is closer to that of Min-Cost. On the other hand, when the traffic rates are high (figures on the right column), Min-Cost's service delay (hence the SLA violation) gets close to that of Min-Viol, and the gap between their cost increases, mainly due to the greater number of deployed fog services in Min-Viol.

The figures on the fourth row illustrate the percentage of the SLA violations. The All Cloud approach (not shown) has SLA violation of 100%. Min-Viol has the lowest SLA violation, and Min-Cost has the second lowest SLA violation. It can be

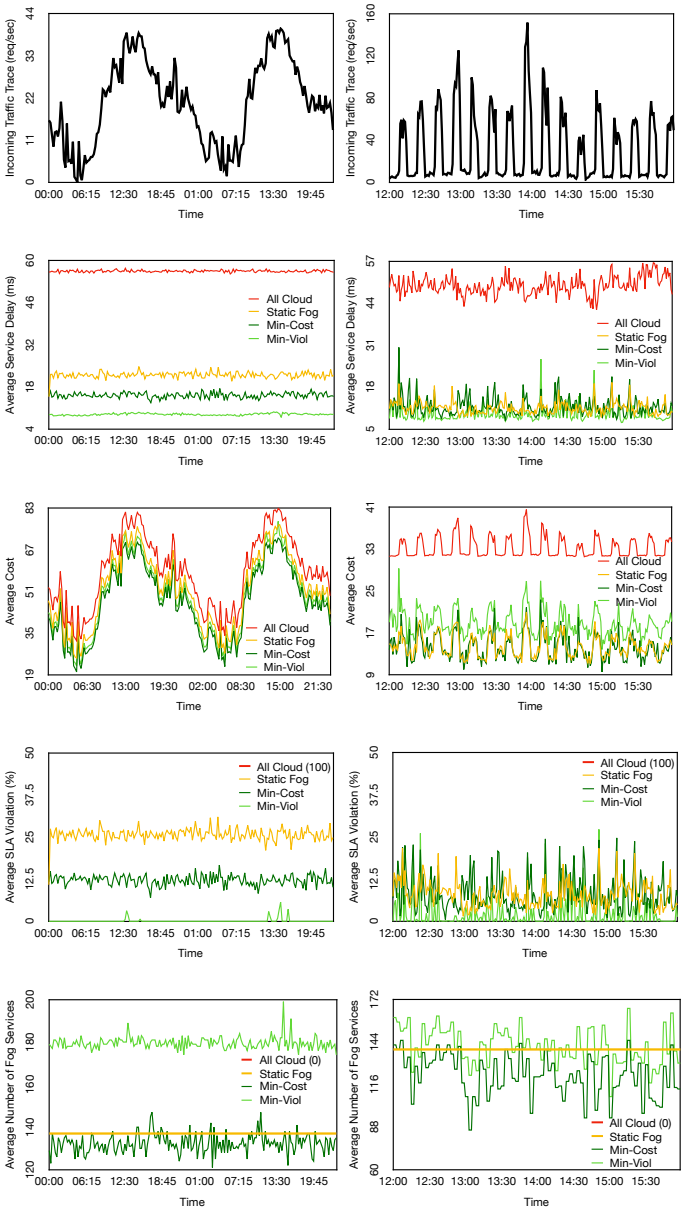


Fig. 2. Simulation Results. Left: 48-hour trace. Right: 4-hour trace

observed that when we run the heuristics more frequently in comparison to the rate of traffic change (the figure on the left column), the SLA violation of Min-Viol is minimized. This is because, in each instance of running the heuristics, the best deployment strategy for the current traffic is determined. If the deployment strategy is not updated frequently, the current placement of the services may not be good for the future traffic demand. This is the main reason that the Fog Static approach performs worse that the proposed heuristics.

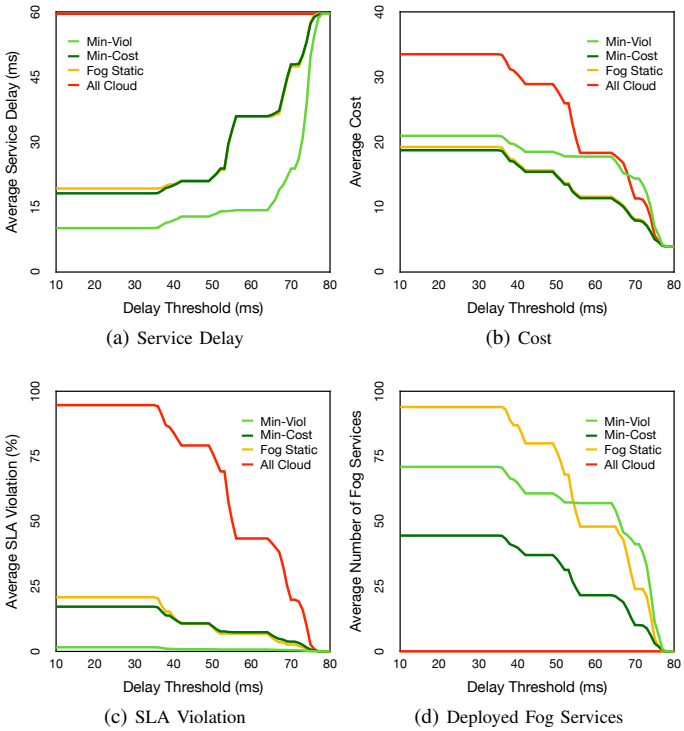
The figures on the last row of Fig. 2 show the average number of services deployed on fog nodes. The All Cloud approach (not shown) does not deploy any services on fog nodes, and as expected, the number of deployed services on

the fog nodes for the Fog Static approach does not change over time. It is evident that, being oblivious to the cost, Min-Viol deploys more services on the fog nodes than Min-Cost, to minimize the service delay, thus bringing the SLA violations to the agreed QoS terms of the SLA. Min-Cost has the lowest number of deployed services on the fog nodes, as it tries to maintain a low-cost deployment.

The next set of diagrams shown in Fig. 3 illustrate the effects of delay threshold th_a for service a on the simulation results. These figures are obtained with the trace of 4 hours (12:00PM-4:00PM) of *2017/04/12*. The interval of traffic change is 1 minute, and τ is 2 minutes. The rest of the parameters of the simulation are same as before. Fig 3a depicts how delay threshold affects the average service delay. When the delay threshold increases, the average service delay of all the fog approaches (Min-Viol, Min-Cost, and Fog Static) increases, and finally gets equal to that of the All Cloud. This is because when the delay threshold is large, the heuristics tends not to deploy services on the fog nodes, hence requests will have large service delays as they will be served in the cloud. As expected, Min-Viol has the lowest average service delay among all the approaches (when the delay threshold is below 77 ms).

Fig. 3b illustrates how the decrease in the average cost of all four approaches when the delay threshold increases. This is the case, because when the delay threshold is high, less services are deployed on the fog nodes, and less requests violate their delay threshold requirements. When the delay threshold is high, there will be less SLA violations, and the cost of All Cloud gets closer to that of the other fog approaches. When the delay threshold is larger than 76 ms, there will be no SLA violations even if the services are not deployed on the fog nodes. Hence, all the services will be deployed in the cloud, and all the approaches will have the same cost value. When the delay threshold is not too large, as expected, Min-Cost and Fog Static have the lowest cost, since both the approaches aim at minimizing the cost.

Fig. 3c shows the performance of the four approaches with regards to SLA violations. Min-Viol's SLA violation is the lowest, while the All Cloud approach has the highest SLA violation. The violations of Min-Cost and Fog Static are moderate, because it is optimized by the minimization of the cost. As the delay threshold gets higher, the SLA violation of all the approaches gets smaller. The SLA violation does not change when the delay threshold is less than 38 ms. The reason is that, with the values of the simulation parameters in the current set up (propagation delays, arrival rates, SLA violation penalties, etc.), the heuristics place the same number of fog services on the fog nodes when the delay threshold is below 38 ms. Nevertheless, when the delay threshold gets larger than 38 ms, less fog services are deployed, and the average SLA violation gets lower. This constant SLA violation period is more clear in Fig. 3d. In the figure, the number of deployed services on fog nodes remains constant when the delay threshold is less than 38 ms. It is interesting to note that even though Min-Cost has significantly less number of deployed services on fog nodes than Fog Static, it achieves a similar performance to that of the Fog Static in all the figures.



(a) Service Delay (b) Cost

(c) SLA Violation (d) Deployed Fog Services

Fig. 3. Impact of delay threshold th_a on the simulation results.

VII. Conclusion

Fog is a layer sitting between cloud and IoT, that brings low latency, location awareness, and reduced bandwidth to the IoT. We introduced an interaction model for the entities of a fog ecosystem, that clearly assigns the role of each entity, in order to attain the benefits of the fog. We discussed how DFSP could substantially benefit Fog Service Providers and their clients, in terms of QoS and cost savings. The DFSP problem is introduced as an INLP task, and two heuristics are proposed to solve it efficiently. Finally, we presented the results of our experiments based on real-world traffic traces, which supports our claims regarding the usefulness of DFSP.

As future work, one can solve the INLP problem for different inputs and see how close the heuristics are to the optimal solution obtained by solving the INLP problem. It is also interesting to see how the distance of the FSC, relative to the fog nodes, can affect the proposed scheme. Lastly, one can envisage a scenario with multiple FSCs, and research network design problems, such as Fog Service Controller placement problem and synchronization among Fog Service Controllers..

References

[1] C. C. Byers, "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[2] R. S. Montero, E. Rojas, A. A. Carrillo, and I. M. Llorente, "Extending the cloud to the network edge.," *IEEE Computer*, vol. 50, no. 4, pp. 91–95, 2017.

[3] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *10th ACM International Conference on Distributed and Event-based Systems*, pp. 258–269, 2016.

[4] Y.-J. Yu, T.-C. Chiu, A.-C. Pang, M.-F. Chen, and J. Liu, "Virtual machine placement for backhaul traffic minimization in fog radio access networks," in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–7, 2017.

[5] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, 2017.

[6] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, "Dynamic module deployment in a fog computing platform," in *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, pp. 1–6, 2016.

[7] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Edge Computing (EDGE), 2017 IEEE International Conference on*, pp. 47–54, 2017.

[8] V. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *Communications (ICC), 2016 IEEE International Conference on*, pp. 1–5, 2016.

[9] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*, pp. 32–39, 2016.

[10] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*, pp. 89–96, 2017.

[11] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[12] W. Zhang, Y. Hu, Y. Zhang, and D. Raychaudhuri, "Segue: Quality of service aware edge cloud service migration," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*, pp. 344–351, 2016.

[13] S. Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. B. Hadj-Alouane, M. J. Morrow, and P. A. Polakos, "A platform as-a-service for hybrid cloud/fog environments," in *Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on*, pp. 1–7, 2016.

[14] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pp. 258–269, 2016.

[15] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the internet of things," in *Edge Computing (EDGE), 2017 IEEE International Conference on*, pp. 17–24, 2017.

[16] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, 2017.

[17] Y. Xiao and M. Krun, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2017.

[18] "Wide mawi working group traffic archive," http://mawi.wide.ad.jp.

[19] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, and C. Z. Patrikakis, "A cooperative fog approach for effective workload balancing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 36–45, 2017.

[20] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The impact of mobile multimedia applications on data center consolidation," in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pp. 166–176, 2013.