FRUGAL: Provisioning of Fog Services over 5G Slices To Meet QoS with Minimal Cost

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

***Abstract*—Recent advances in the areas of Internet of Things (IoT), cloud computing and big data have been attributed to the rise of a growing number of complex and useful applications. On the other hand, The fifth generation (5G) wireless technology is envisioned to provide faster Internet access, with lower latency, and ubiquitous mobile coverage compared to its predecessors. 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 in 5G, and fog computing is seen as one of the primary enablers for satisfying these QoS requirements. Fog puts compute, storage, and networking resources closer to the user.**

In this report, we show the system model of FRUGAL, a framework for QoS-aware Fog-supported 5G Slice Provisioning (QFSP). QFSP concerns the dynamic deployment or release of application services over a 5G slice on fog nodes, or dimensioning (enlarge or shrink) of a fog-supported 5G slice, in order to meet the latency and QoS constraints of applications while minimizing cost.

I. SYSTEM MODEL

In this paper we study the QFSP problem, which is to dimension (enlarge or shrink) a fog-supported 5G slice to comply with the latency (i.e. QoS) constraints of the application while also minimizing the cost of the slice resources. The Slice Owner (SO) wishes to run (possibly latency-sensitive) application(s) over her slice. The SO solves an instance of the QFSP problem to find a "good" solution for utilizing her 5G slice; the SO wishes to comply with the latency constraints of her application and wishes to do so with minimal resource cost. A slice can be shrunk to free up extra resources, which may save costs, or may be enlarged if needed, to serve the rising demands, which may reduce the violation costs. Moreover, within a slice, application components (i.e. services) may be dimensioned or placed in different locations to comply with the latency constraints and/or to minimize the cost.

A 5G slice is define as networking, compute, storage, and memory resources reserved to serve one or more applications in the 5G networks. Networking resources are modeled in terms of reserved bandwidth between two physical nodes, whereas compute, storage, and memory are modeled in terms of reserved resources within a physical node. A slice can be comprised of several *applications* that span across fog, cloud, and edge networks.

An IoT application may be composed of several *services* that are essentially the components of the application and can run in different locations. Such services are normally implemented as containers, virtual machines (VMs), or unikernels. For instance, an application may have services such as authentication, firewall, caching, and encryption. Some application services are delay-sensitive and have tight delay thresholds, and may need to run closer to the users/data sources (e.g. caching), for instance at the edge of the network or on fog computing devices. On the other hand, some application services are delay-tolerant and may have high availability requirements, and may be deployed farther from the users/data sources along the fog-to-cloud continuum or in the cloud (e.g. authentication). We label these fog-capable IoT services that could run on the fog computing devices or the cloud servers as *fog services*.

To formulate the QFSP problem, we introduce some notation. Let a set of (telecom) edge nodes be denoted by F, a set of cloud servers by C, and a set of fog services running over the SO's slice by A. Some services comprise the components of a complex IoT application that will eventually be run over a 5G slice. Some edge nodes have fog computing capabilities (e.g. compute, storage, memory), while some edge nodes solely act as the *access points* in the edge network and are the entry points of the incoming IoT traffic to the telecom network. Let $f(j) = 1$, if edge node $j \in F$ have fog computing capability (hence called a *fog node*), and $f(j) = 0$, otherwise.

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. Let p_a denote the penalty that the SO experiences if latency requirements of service a are violated by 1% per unit time, and let $V_a^{\%}$ be the percentage of IoT service delay samples of service a that do not meet the delay requirement. For instance if $q_a = 97\%$, any violation percentage $V_a^{\%}$ greater than 3% must be compensated for by the SO.

The underlying communication network, set of edge nodes and cloud servers are modeled as a graph $G = (V, E)$, such that the node set V includes the edge nodes F and cloud servers C ($V = F \cup C$), and the graph edge set E includes the logical links between the nodes in V. Each graph edge $e(src, dst) \in E$ is associated with two numbers: r_e, the transmission rate of logical link e (megabits per second); and d_e, the propagation delay of logical link e (milliseconds). For the sake of correct indexing, if $e(src, dst)$ is the logical link between an edge node and a cloud server, we rename r_e and d_e to r'_e and d'_e, respectively. These parameters are maintained

by the Physical Infrastructure Provider (PIP) and are shared with the SO.

The main decision variables of the QFSP problem are the resource allocation variables, defined in the table I.

A. Optimization Problem

The QFSP problem can be formulated as the following minimization of Cost(t) over all time intervals (starting from $t = 0$ to $t = T^{\text{end}}$):

$$\begin{aligned}\textbf{P1}: \ & \min \sum_{0\le t\le T^{\text{end}}} \text{Cost}(t),\\ & \text{Cost}(t) = [C_C^{\text{proc}}(t) + C_F^{\text{proc}}(t)] + \\ & [C_C^{\text{stor}}(t) + C_F^{\text{stor}}(t)] + [C_C^{\text{mem}}(t) + C_F^{\text{mem}}(t)] + \\ & [C_{FC}^{\text{net}}(t) + C_{FF}^{\text{net}}(t)] + C^{\text{viol}}(t), \\ & \text{Subject to QoS constraints.}\end{aligned} \tag{1}$$

The cost components are defined below.

$$C_C^{\text{proc}}(t) = \sum_{k\in C} X_k'^P C_k'^P T_t, \tag{2}$$

$$C_F^{\text{proc}}(t) = \sum_{j\in F} X_j^P C_j^P T_t, \tag{3}$$

$$C_C^{\text{stor}}(t) = \sum_{k\in C} (X_k'^S C_k'^S + \sum_{a\in A} \lambda_{ak}'^{\text{in}} C_k'^R) T_t, \tag{4}$$

$$C_F^{\text{stor}}(t) = \sum_{j\in F} (X_j^S C_j^S + \sum_{a\in A} \lambda_{aj}^{\text{in}} C_j^R) T_t, \tag{5}$$

$$C_C^{\text{mem}}(t) = \sum_{k\in C} X_k'^M C_k'^M T_t, \tag{6}$$

$$C_F^{\text{mem}}(t) = \sum_{j\in F} X_j^M C_j^M T_t, \tag{7}$$

$$C_{FC}^{\text{net}}(t) = \sum_{j\in F} \sum_{a\in A} Y'_{jh_a(j)} u_{(j,h_a(j))} T_t, \tag{8}$$

$$C_{FF}^{\text{net}}(t) = \sum_{j\in F} \sum_{j'\in F} Y_{jj'} u_{(j,j')} T_t, \tag{9}$$

$$C^{\text{viol}}(t) = \sum_{j\in F} \sum_{a\in A} [V_a^{\%} - (1-q_a)]^+ \lambda_{aj}^{\text{in}} p_a T_t. \tag{10}$$

The notation $[.]^+$ in the definition of C^{viol} is defined as $[x]^+ = \max(x, 0)$.

$C_k'^R$ and C_j^R are the unit cost of request in cloud server k and fog node j, respectively (e.g. per 1000 requests).

C_C^{proc} and C_F^{proc} are cost of processing in cloud and fog, C_C^{stor} and C_F^{stor} are cost of storage in cloud and fog, and C_C^{mem} and C_F^{mem} are cost of memory in cloud and fog, respectively. The cost of processing is defined similar to the pricing of compute instances in Amazon EC2 [1] and the cost of storage is defined similar to the pricing of storage instances in Amazon S3, which is the storage cost plus the request cost (GET, POST, etc.) [2]. 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^{viol} is the cost (penalty) of delay violations. A service deployed on a fog node may be released when the demand for the service is small. Therefore, we assume the services are stateless, that is they do not store any state information on fog nodes [3], [4], and we do not consider costs for state migrations. We consider a discrete-time system model where time is divided into time periods called *re-configuration periods*. T is the time interval between two instances of solving the optimization problem.

B. Constraints

The constraints of the optimization problem are introduced here.

1) Maximum Allowed Slice Capacity: The PIP exerts a maximum allowed slice capacity (MASC) for a given slice. In other words, the SO cannot allocate to her slice more resources than the maximum allowed slice capacity imposed by the PIP. Let the MASC for compute, storage, and memory at fog node j be K_j^P, K_j^S, K_j^M, and at cloud server k be $K_k'^P$, $K_p'^S$, $K_k'^M$, respectively. Also, let the MASC for networking (bandwidth) between fog node j and fog node j be $K_{jj'}^N$ and between fog node j and cloud server k be $K_{jk}'^N$. First, we express the MASC constraints for the nodes in the SO's slice:

$$X_j^P < K_j^P, \ \ \forall j \in F; \ X_k'^P < K_k'^P, \ \ \forall k \in C; \tag{11}$$

$$X_j^S < K_j^S, \ \ \forall j \in F; \ X_k'^S < K_k'^S, \ \ \forall k \in C; \tag{12}$$

$$X_j^M < K_j^M, \ \forall j \in F; \ X_k'^M < K_k'^M, \ \forall k \in C. \tag{13}$$

Note that in certain scenarios the MASC for cloud servers may be a very large value, and thus we can relax those constraints in such scenarios. Similar to the MASC constraints for the nodes, MASC constraints for the links in the SO's slice are

$$Y_{jj'} \le K_{jj'}^N, \qquad \forall (j,j') \in E, \tag{14}$$

$$Y'_{jk} \le K_{jk}'^N, \qquad \forall (j,k) \in E. \tag{15}$$

2) Traffic Offloading: We aim to define the QoS-aware Fog-supported 5G Slice Provisioning (QFSP) problem general enough and agnostic to the underlying traffic offloading scheme such that different offloading frameworks can be used. The simplest offloading scheme of fog nodes is *no-offload*. In the *no-offload* framework, when a request arrives to the fog node, the fog node will process it if its queue is not full, or drops the request if the queue is full. It can be seen that *no-offload* is not a good traffic offloading policy, since requests may be blocked or overloaded fog nodes may have to process more requests which will result in high latency. Another simple offloading policy is *cloud-only-offload*, where offloading happens only from a fog node to the cloud. Another offloading policy is *single-neighbor-multi-offload*, when a fog node can offload traffic to one of its neighbors (e.g. the one with the smallest waiting time, as in our prior work [5]) and offloading can happen multiple times among the fog nodes until it reaches the *max-offload-limit* and is hence offloaded to the cloud. One can envisage many such offloading policies.

In this subsection we select one such offloading policy to show how an arbitrary offloading scheme can fit in our

TABLE I
VARIABLES OF QFSP PROBLEM

Service Variables	
$x^P_{aj} \geq 0$	Computing resources allocated to service a over the SO's slice on fog node j (in MIPS)
$x^S_{aj} \geq 0$	Storage allocated to service a over the SO's slice on fog node j (in megabits)
$x^M_{aj} \geq 0$	Memory allocated to service a over the SO's slice on fog node j (in megabits)
$x'^P_{ak} \geq 0$	Computing resources allocated to service a over the SO's slice on cloud server k (in MIPS)
$x'^S_{ak} \geq 0$	Storage allocated to service a over the SO's slice on cloud server k (in megabits)
$x'^M_{ak} \geq 0$	Memory allocated to service a over the SO's slice on cloud server k (in megabits)
Traffic Variables	
$0 \leq t_{ajj'} \leq 1$	Fraction of traffic for service a offloaded from fog node j to fog node j'
$0 \leq t'_{ajk} \leq 1$	Fraction of traffic for service a offloaded from fog node j to cloud server k
Slice Node Variables	
$X^P_j \geq 0$	Computing resources allocated to the SO's slice on fog node j (in MIPS)
$X^S_j \geq 0$	Storage allocated to the SO's slice on fog node j (in megabits)
$X^M_j \geq 0$	Memory allocated to the SO's slice on fog node j (in megabits)
$X'^P_k \geq 0$	Computing resources allocated the SO's slice on cloud server k (in MIPS)
$X'^S_k \geq 0$	Storage allocated to the SO's slice on cloud server k (in megabits)
$X'^M_k \geq 0$	Memory allocated to the SO's slice on cloud server k (in megabits)
Slice Link Variables	
$Y_{jj'} \geq 0$	Networking resources (bandwidth) allocated to the (logical) link between edge node j and edge node j' in the SO's slice (in megabits per second)
$Y'_{jk} \geq 0$	Networking resources (bandwidth) allocated to the (logical) link between edge node j and cloud server k in the SO's slice (in megabits per second)

framework. We choose and describe the *multi-neighbor-single-offload*, where the traffic from a fog node can be offloaded to all of the fog node's fog neighbors and also to the cloud. In *multi-neighbor-single-offload* an offloaded traffic to a fog node cannot be offloaded again and must be processed by the receiving node.

Let I_{aj} denote the incoming IoT traffic for service a over the SO's slice at fog node j, and let $\lambda_{aj'j}$ denote the offloaded traffic for service a from fog node j' to fog node j over the SO's slice. Then the total incoming traffic of fog node j for service a over the SO's slice, v_{aj}, will be

$$v_{aj} = I_{aj} + \sum_{j' \in F} \lambda_{aj'j}. \tag{16}$$

Note that in *multi-neighbor-single-offload* an offloaded traffic to a fog node cannot be offloaded again; nonetheless, not all of I_{aj} is *"accepted"* by fog node j for processing. Let us define $t_{ajj'}$ as the fraction of traffic for service a that is offloaded from fog node j to fog node j' (when $j' = j$, t_{ajj} indicates the fraction of *accepted* traffic by fog node j). Similarly, t'_{ajk} is the fraction of traffic for service a that is offloaded from fog node j to cloud server k. Obviously, these fractions should add up to one for a given service a at a given fog node j, or

$$t'_{ajh_a(j)} + \sum_{j' \in F} t_{ajj'} = 1, \quad \forall a \in A, \forall j \in F. \tag{17}$$

The *accepted* incoming traffic to fog node j for service a over the SO's slice is

$$\lambda^{\text{in}}_{aj} = t_{ajj} \times I_{aj} + \sum_{j' \in F} \lambda_{aj'j}. \tag{18}$$

The fraction of I_{aj} that is not accepted by the fog node will be offloaded either to another fog node, or to a cloud server. The dispatched traffic from fog node j to cloud server $k = h_a(j)$ for service a over the SO's slice is denoted by $\lambda^{\text{out}}_{aj}$ and is derived by

$$\lambda^{\text{out}}_{aj} = t'_{ajh_a(j)} \times I_{aj}. \tag{19}$$

The incoming traffic to cloud server k for service a over the SO's slice is

$$\lambda'^{\text{in}}_{ak} = I'_{ak} + \sum_{j \in H_a^{-1}(k)} \lambda^{\text{out}}_{aj}, \tag{20}$$

where $H_a^{-1}(k)$ set of indices of all fog nodes that route the traffic for service a to cloud server k, and I'_{ak} is the incoming IoT traffic for service a over the SO's slice at cloud server k.

Similar to Eq. (18) and Eq. (19), the offloaded traffic from fog node j to fog node j' for service a over the SO's slice is

$$\lambda_{ajj'} = t_{ajj'} \times I_{aj}. \tag{21}$$

3) Fog Computing Capability: If edge node j does not have fog computing capabilities, it should not accept any traffic for processing and it should offload all of the incoming traffic either to a fog node or a cloud server; that is, if $f(j) = 0$, then $t_{ajj} = 0$. Similarly, if edge node j does not have fog computing capabilities, other edge nodes $j' \in F$ should not offload any traffic to this node; that is, if $f(j) = 0$, then $t_{aj'j} = 0$. These two constraints can be linearly described by

$$t_{aj'j} \leq f(j), \quad \forall a \in A, \forall j, j' \in F. \tag{22}$$

4) Service Delay: IoT 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. (The closed-form equation of IoT service delay is explained in our prior study [6]). To obtain the service delay, we need to have the average propagation delay and average transmission rate between IoT nodes and their corresponding fog nodes. These values must be known by the SO, and in some cases can be approximated by round-trip delay measurement techniques. However, since obtaining these values are not always feasible, we change the scope of the definition of the IoT service delay to consider the delay only within fog to cloud domains.

IoT service delay captures the delay from the moment an IoT node sends a request until it receives the response for that request. This can be changed to capture the delay

from the moment the request reaches a fog node. This new service delay, d_{aj}, can be realized as the average *delay budget* for service a at fog node j within fog-cloud. The average delay budget for the *multi-neighbor-single-offload* scheme for service a at fog node j is equal to

$$d_{aj} = w_{aj} \times t_{ajj} + \sum_{j' \in F} [2d_{(j,j')} + \frac{l_a^{rq} + l_a^{rp}}{Y_{jj'}} + w_{aj'}] \times t_{ajj'} + [2d'_{(j,k)} + \frac{l_a^{rq} + l_a^{rp}}{Y'_{jk}} + w'_{ak}] \times t'_{ajk}; \qquad k = h_a(j) \quad (23)$$

Similarly, the average delay budget for the *single-neighbor-single-offload* scheme for service a at fog node j is equal to

$$d_{aj} = w_{aj} \times t_{ajj} + [2d_{(j,j^*)} + \frac{l_a^{rq} + l_a^{rp}}{Y_{jj^*}} + w_{aj^*}] \times t_{ajj^*} + [2d'_{(j,k)} + \frac{l_a^{rq} + l_a^{rp}}{Y'_{jk}} + w'_{ak}] \times t'_{ajk}; \qquad k = h_a(j) \quad (24)$$

where j^* is the *best neighbor* of fog node j. The variables t_{ajj}, $t_{ajj'}$, t_{ajk}, $Y_{jj'}$, and Y'_{jk} are calculated by the QFSP problem. In order to evaluate the delay budget, we need to have the average size of requests and reply of service a (l_a^{rq} and l_a^{rp}) and the propagation delay between fog nodes and from fog nodes to cloud servers ($d_{(j,j')}$ and $d'_{(j,k)}$), which are known or measured by the SO. We also need the average waiting times (queueing time plus processing time), w_{aj}, $w_{aj'}$, and w'_{ak}, which can be obtained either from the corresponding M/M/c queueing models of the fog nodes and the cloud servers (discussed in Section I-B8), or based on predictive performance modeling and black-box monitoring techniques [7]. In either case, the incoming traffic to fog nodes is required to obtain the average waiting times. The incoming traffic to the fog nodes (I_{aj}) can either be directly monitored by the SO (e.g. using the monitoring agent of an SDN controller [6]) or can be predicted by the SO using a learning approach (to be discussed in Section II).

In the first approach where I_{aj} is monitored (at the beginning of each configuration interval), the QFSP problem is solved in a reactive nature; whereas the second approach that predicts I_{aj} is proactive, since it provisions the resources ahead of time based on the estimated incoming IoT traffic. Moreover, the predicting approach has the advantage of getting an average of I_{aj} during a configuration interval, as apposed to the monitoring approach that only obtains an instance of I_{aj} during a configuration interval.

In the Section II we discuss how employ learning methods to predict I_{aj} and obtain the waiting times of the fog nodes and cloud servers.

5) SLA Violation: To measure the 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 delay budget of fog node j for service a is greater than the threshold th_a defined in SLA for service a. Let us define a binary variable v_{aj} to indicate this:

$$v_{aj} = \begin{cases} 1, & \text{if } d_{aj} > th_a \\ 0, & \text{otherwise} \end{cases}, \quad \forall j \in F, \forall a \in A. \qquad (25)$$

We define another variable that measures the SLA violation (SLAV) of a given service according to the defined QoS parameters in the SLA. 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

$$V_a^{\%} = \frac{\sum_j \lambda_{aj}^{\text{in}} v_{aj}}{\sum_j \lambda_{aj}^{\text{in}}}, \qquad \forall a \in A. \qquad (26)$$

Note that $V_a^{\%}$ is measured as a weighted average of v_{aj}, with λ_{aj}^{in} as the weight.

6) 5G Slice Resource Capacity: The amount of the resources allocated to the services and traffic within the SO's slice should not exceed the capacity of the slice; this applies to all the nodes and the links of the slice.

We first look at this constraints with respect to the nodes in the slice, i.e. compute, storage, and memory resources within a node of the slice:

$$\sum_{a \in A} x_{aj}^P < X_j^P, \quad \forall j \in F; \quad \sum_{a \in A} x'^P_{ak} < X'^P_k, \quad \forall k \in C; \quad (27)$$

$$\sum_{a \in A} x_{aj}^S < X_j^S, \quad \forall j \in F; \quad \sum_{a \in A} x'^S_{ak} < X'^S_k, \quad \forall k \in C; \quad (28)$$

$$\sum_{a \in A} x_{aj}^M < X_j^M, \ \forall j \in F; \quad \sum_{a \in A} x'^M_{ak} < X'^M_k, \ \forall k \in C. \quad (29)$$

Similarly, we also have constraints for capacity of the links in a slice: the amount of traffic routed over a particular slice link should not exceed the amount of allocated bandwidth to that slice link, which is expressed by

$$\sum_{a \in A} \lambda_{aj'j} (l_a^{rq} + l_a^{rp}) \leq Y_{jj'}, \qquad \forall (j, j') \in E, \quad (30)$$

$$\sum_{a \in A} \lambda_{aj}^{\text{out}} (l_a^{rq} + l_a^{rp}) \leq Y'_{jk}, \quad \forall (j, k) \in E. \qquad (31)$$

7) Arrival of Requests: Let Λ_{aj} denote the arrival rate of instructions (in MIPS) to fog node j for service a over the SO's slice. This is the arrival rate of instructions of the incoming requests that are accepted for processing by fog node j that is given by $\Lambda_{aj} = L_a^P \lambda_{aj}^{\text{in}}$. Similarly, the arrival rate of instructions (in MIPS) to the cloud server k for service a over slice the SO's slice, Λ'_{ak}, can be written as $\Lambda'_{ak} = L_a^P \lambda'^{\text{in}}_{ak}$, where $\lambda'^{\text{in}}_{ak}$ is the incoming traffic rate to cloud server k for service a over the SO's slice. L_a^P is the required amount of processing for service a per request.

The amount of computing resources allocated to the fog nodes and cloud servers for a given service on the SO's slice must be bigger than the incoming arrival of processing requests for that service (i.e. stability constraints):

$$x_{aj}^P \geq \Lambda_{aj}, \quad \forall j \in F, \forall a \in A, \qquad (32)$$

$$x'^P_{ak} \geq \Lambda'_{ak}, \quad \forall k \in C, \forall a \in A. \qquad (33)$$

8) Waiting Times: To get the waiting times, We adopt a commonly used M/M/c queueing system [8], [9], [10] model for a fog node with n_j processing units, each with service rate μ_j and total arrival rate of $\sum_{a\in A}\Lambda_{aj}$ (total processing capacity of fog node j will be $K_j^P = n_j\mu_j$).

To model what fraction of processing units each service can obtain, we assume that the processing units of a fog node are allocated to the deployed services proportional to their processing needs (L_a^P). For instance, if the requests for service a_1 need twice the amount of processing than that of the requests for service a_2 ($L_{a_1}^P = 2 \times L_{a_2}^P$), service a_1 should receive twice the service rate compared to service a_2. Correspondingly, we define f_{aj}, the fraction of service rate that service a obtains at fog node j as:

$$f_{aj} = \frac{x_{aj}L_a^P}{\sum_{a\in A} x_{aj}L_a^P}. \tag{34}$$

Each deployed service can be seen as an M/M/c queueing system with service rate of $f_{aj} \times K_j^P = f_{aj}n_j\mu_j$, and arrival rate of Λ_{aj} (both in MIPS). Thus, the waiting time for requests of service a at fog node j will be

$$w_{aj} = \frac{1}{f_{aj}\mu_j} + \frac{\mathcal{P}_{aj}^Q}{f_{aj}K_j^P - \Lambda_{aj}}, \tag{35}$$

where $\mathcal{P}_{aj}^Q$ is the probability that an arriving request to fog node j for service a has to wait in the queue. $\mathcal{P}_{..}^Q$ is also referred to as *Erlang's C formula* and is equal to

$$\mathcal{P}_{aj}^Q = \frac{(n_j\rho_{aj})^{n_j}}{n_j!}\frac{\mathcal{P}_{aj}^0}{1-\rho_{aj}}, \tag{36}$$

such that $\rho_{aj} = \frac{\Lambda_{aj}}{f_{aj}K_j^P}$ and

$$\mathcal{P}_{aj}^0 = \Big[\sum_{c=0}^{n_j-1}\frac{(n_j\rho_{aj})^c}{c!} + \frac{(n_j\rho_{aj})^{n_j}}{n_j!}\frac{1}{1-\rho_{aj}}\Big]^{-1}. \tag{37}$$

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

Similarly, cloud server k with n'_k processing units (i.e. servers), each with service rate μ'_k and total arrival rate of $\sum_{a\in A}\Lambda'_{ak}$ can be seen as an M/M/c queueing system (total processing capacity of cloud server k will be $K_k'^P = n'_k \times \mu'_k$). Therefore, similar to Eq. (35), w'_{ak}, the waiting time for requests of service a at cloud server k, could be derived as

$$w'_{ak} = \frac{1}{f'_{ak}\mu'_k} + \frac{\mathcal{P}_{ak}'^Q}{f'_{ak}K_k'^P - \Lambda'_{ak}}, \qquad \forall a \in A, \forall k \in C. \tag{38}$$

An equation similar to Eq. (36) is defined for $\mathcal{P}_{ak}'^Q$, the probability of queueing at cloud server k. Note that for simplicity, instead of modeling each cloud server a M/M/c queue, one may also model the whole cloud as an M/M/∞ queueing system.

9) Dependency of Compute, Storage, and Memory: The variables for compute, storage, and memory allocation are interrelated to each other; when there is no compute resources allocated to a node for a particular service, there is no need to allocate corresponding storage and memory resources to the node for that service. These dependencies for fog nodes can be formulated using the following constraints:

$$x_{aj}^S = \begin{cases} 0, & \text{if } x_{aj}^P = 0 \\ L_a^S, & \text{otherwise} \end{cases}, \quad \forall j \in F, \forall a \in A, \tag{39}$$

$$x_{aj}^M = \begin{cases} 0, & \text{if } x_{aj}^P = 0 \\ L_a^M, & \text{otherwise} \end{cases}, \quad \forall j \in F, \forall a \in A, \tag{40}$$

and for cloud servers using the following constraints:

$$x_{ak}'^S = \begin{cases} 0, & \text{if } x_{ak}'^P = 0 \\ L_a^S, & \text{otherwise} \end{cases}, \quad \forall k \in C, \forall a \in A, \tag{41}$$

$$x_{ak}'^M = \begin{cases} 0, & \text{if } x_{ak}'^P = 0 \\ L_a^M, & \text{otherwise} \end{cases}, \quad \forall k \in C, \forall a \in A. \tag{42}$$

L_a^S and L_a^M are the required minimum amount of storage and memory for service a, respectively. It is worth mentioning that since cloud is usually regarded as having ample resources, the constraints (41) and (42) could be easily modified to allocate more storage and memory resources for cloud servers when $x_{ak}'^P > 0$.

II. Learning Method for Incoming IoT Traffic

A. Predicting Incoming IoT Traffic

To predict the incoming traffic to Iot nodes, we employ the *follow the regularized leader (FTRL)* online learning method [11]. Our method for traffic prediction is mainly influenced by the demand prediction method discussed in [12].

To account for the amount of incoming traffic in different configuration intervals, we add time variable t to the variable I_{aj}, make it $I_{aj}(t)$, the *predicted* incoming IoT traffic for service a over the SO's slice at fog node j in the configuration interval t. Let $I_{aj}^*(t)$ be the actual (e.g. measured) incoming IoT traffic for service a over the SO's slice at fog node j in time t. Let us define $I_{aj}^{\max} = \max_{t\in[0,T^{\text{end}}]}(I_{aj}(t))$, which is known to or can be measured by the SO based on historical observations. Hence, the predicted incoming traffic $I_{aj}(t)$ will be in $[0, I_{aj}^{\max}]$. We can then divide the time into configuration intervals so that we have countable number of $I_{aj}(t)$'s. We define a convex loss function to minimize the prediction error for the incoming IoT traffic $I_{aj}(t)$ as

$$E_t(I_{aj}) = |I_{aj}(t) - I_{aj}^*(t)|, \quad \forall j \in F, \forall a \in A, \forall t. \tag{43}$$

Therefore, a loss minimization problem over all time slots then can be expressed as

$$\min \sum_{0\le t\le T^{\text{end}}} E_t(I_{aj}). \tag{44}$$

The solution to this minimization problem is a set of values for $I_{aj}(t) \in [0, I_{aj}^{\max}]$ for every $t \in [0, T^{\text{end}}]$.

FTRL chooses the predicted incoming traffic value $I_{aj}(t+1)$ that minimizes the cumulative loss function over the previous t time slots plus a regularizer (to avoid overfitting), that is

$$I_{aj}(t+1) = \arg \min_{I_{aj} \in [0, I_{aj}^{\max}]} [\sum_{s=0}^{t} E_s(I_{aj}) + R(I_{aj})]. \quad (45)$$

where I_{aj} reflects the possible predicted incoming IoT traffic rate for service a to fog node j in interval $t+1$. $R(.)$ is a convex regularizer. Since solving this optimization problem at each reconfiguration interval is not feasible, we convert it to the following problem

$$I_{aj}(t+1) = \arg \min_{I_{aj} \in [0, I_{aj}^{\max}]} h_t(I_{aj}), \quad (46)$$

where $h_t(.)$ is the surrogate loss function that should accurately estimates the original loss function and is efficient to evaluate. We use the surrogate function chosen in [12] as

$$h_t(I_{aj}) = [\sum_{s=0}^{t} (\bigtriangledown E_s(I_{aj})] \times I_{aj} + r_t(I_{aj}). \quad (47)$$

$\bigtriangledown E_s(I_{aj})$ is the gradient of $E_s(I_{aj})$ and $r_t(.)$ is a regularization function (similar to $R(.)$). The regularization function is chosen as

$$r_t(x) = 0.5 \sum_{s=0}^{t} (\frac{1}{\alpha_s} - \frac{1}{\alpha_{s-1}})(x - x(s))^2, \quad (48)$$

where $\alpha_s = \Theta(\frac{1}{\sqrt{s}})$ is e learning rate in time slot s [12]. Specifically, we will have

$$\alpha_t = \frac{\beta}{\sqrt{\sum_{s=0}^{t} (\bigtriangledown E_s(I_{aj}))^2}}, \quad (49)$$

and since $(\bigtriangledown E_s(I_{aj}))^2 = 1$, then $\alpha_t = \frac{\beta}{\sqrt{t+1}}$, where β is a hyper parameter based on the features and data (set to $\beta = I_{aj}^{\max}$) [12]. After this conversion, eq. (46) becomes a smooth minimization problem with no constraints. We then simply take the derivative of our chosen surrogate function $h_t(.)$ with respect to I_{aj} and set it to be equal to 0 to obtain the $I_{aj}(t)$ that minimizes eq. (46). Taking derivative is easy and its solution will be the predicated incoming IoT traffic.

B. Obtaining Waiting Times

Now having the predicted incoming IoT traffic, one can also predicts the waiting times. As discussed before, the online learning approach that predicts I_{aj} is proactive in nature, since based on predicted waiting times (and hence violation rate) the SO provisions the resources ahead of time based. Basically, the predicted incoming IoT traffic can be simply plugged in the equations for waiting times to get the predicted waiting times.

Note: We leave the investigation of other choices for surrogate functions and regularization functions as our future work. More noteworthy, one can also study the applicability of reinforcement learning techniques for incoming IoT traffic prediction.

III. TESTBED SETUP

For obtaining numerical evaluations and show the validity and performance of our proposed framework, we implement FRUGAL in the following testbed. We have 4 SDN-enabled HPE switches in our laboratory, which we use to make a tree topology: 1 switch as the root and 3 switches as the leaf nodes. We directly connect one computer to each leaf switch to act as a fog node. hence, we have 3 fog nodes in our testbed. We also connect WiFi access points to the switches to enable the passing traffic from the WiFi-enabled IoT devices.

The FRUGAL runs as an app (e.g. on the same computer that has the root SDN controller) that basically communicates with the SDN controllers through OpenFlow protocol, to monitor the IoT traffic, deploy or release fog service, or dimension the compute, storage or networking resources in the testbed. The FRUGAL solves the QFSP problem and determine what resources need to be deployed, release, or dimensioned.

The experiment will consist of 2 major components: (1) run few IoT applications (e.g. object recognition from camera images) whose traffic pass through these switches in our lab to some service that is hosted in the cloud and (2) analyzing the performance of FRUGAL with regards to service delay, delay violations, and resource usage. The switches, access points, and computers in our lab act as the edge of the network, where fog nodes locate and operate. For the cloud, we can use any cloud service provider such as AWS, Google, or research experiment cloud computing environments such as Chameleon Cloud or CloudLab. The cloud/fog services could be implemented as containers or VMs. For the agile service provisioning in the fog network, we can use Docker containers that can be easily provisioned using kubernetes or OpenStack. kybernetes and OpenStack gives us the flexibility and agility for resource management (deploying, releasing or dimensioning of compute, storage, and networking resources).

To generate IoT traffic, we have can do the following: (1) use the IoT boards we already have in our lab (e.g. Raspberry Pi and Arduino that have camera and other sensors) to generate actual traffic, (2) use the Iperf tool to generate constant bit rate traffic flows, (3) combination of both approaches: use the IoT boards to generate actual traffic, and if needed more, use the Iperf tool to generate more traffic (e.g. for other services or background traffic).

REFERENCES

[1] "Amazon EC2 pricing," 2018. [Available] https://aws.amazon.com/ec2/pricing/.

[2] "Amazon S3 pricing," 2018. [Available] https://aws.amazon.com/s3/pricing/.

[3] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, "Cloudpath: a multi-tier cloud computing framework," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, p. 20, ACM, 2017.

[4] 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.

[5] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[6] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "QoS-aware dynamic fog service provisioning," *arXiv preprint arXiv:1802.00800*, 2018.

[7] C. Witt, M. Bux, W. Gusew, and U. Leser, "Predictive performance modeling for distributed computing using black-box monitoring and machine learning," *arXiv preprint arXiv:1805.11877*, 2018.

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

[9] Z. Zhou, J. Feng, L. Tan, Y. He, and J. Gong, "An air-ground integration approach for mobile edge computing in iot," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 40–47, 2018.

[10] L. Liu, X. Guo, Z. Chang, and T. Ristaniemi, "Joint optimization of energy and delay for computation offloading in cloudlet-assisted mobile cloud computing," *Wireless Networks*, pp. 1–14, July 2018.

[11] B. McMahan, "Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 525–533, 2011.

[12] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 486–494, IEEE, 2018.