When Clones Flock Near the Fog

Sherif Abdelwahab, *Member, IEEE*, Sophia Zhang, Ashley Greenacre, Kai Ovesen, Kevin Bergman, and Bechir Hamdaoui, *Senior Member, IEEE*

***Abstract*—FogMQ is a message brokering and device cloning service in fog and edge computing. Excessive tail end-to-end latency occurs with conventional message brokers when a massive number of geographically distributed devices communicate through a message broker. Latency of broker-less messaging is highly dependent on computational resources of devices. Device-to-device messaging does not necessarily ensure low messaging latency and cannot scale well for a large number of resource-limited and geographically distributed devices. For each device, FogMQ provides a high capacity device cloning service that subscribes to device messages. The clones facilitate near-the-edge data analytics in resourceful cloud compute nodes. Clones in FogMQ apply Flock; an algorithm mimicking flocking-like behavior and allows the clones to autonomously migrate between heterogeneous cloud platforms. Flock controls and minimizes the weighted tail end-to-end latency. We have implemented FogMQ and evaluated it in a geographically distributed testbed. In our functional evaluation, we show that FogMQ is stable and achieves a bounded tail end-to-end latency that is up to 34% less than existing brokering methods.**

***Index Terms*—Game theory, messages brokering, resource management, ubiquitous computing.**

I. Introduction

MANY large-scale applications are sensitive to latency as they rely on messaging subsystems between geographically distributed devices and cloud services. Even if 10% of messages were delayed for longer than 150–300 ms, applications like remote-assisted surgery and real-time situation awareness may not be feasible [1], [2]. A bounded tail end-to-end latency is a cornerstone for the realization of large-scale Internet of Things (IoT) applications near the network edge [3], [4].

When devices communicate through a middle message broker, successive packets queuing in multihop paths becomes a major source of latency. For example, the average end-to-end latency of messages exchanged using a Redis broker in a close Amazon data-center is three times longer than deploying the same broker one-hop away from devices. Broker-less messaging using device-to-device communication does not necessarily solve the successive packets queuing problem. In IoT applications, a device communicates a large number of messages with many devices. Devices limited processing and memory capacity become another major source of latency for large-scale distributed applications. Experiments show that direct device-to-device messages can experience double the end-to-end latency compared to brokering the messages through a one-hop away broker (see Section II).

If devices are cloned in a one-hop away cloudlet [5], a device's clone can provide message brokering service so that interacting devices can communicate with minimal latency and allow devices to offload intensive computation in very large memory and processing nodes that host the clones. Of course, communicating through a one-hop away clone may still cause long *tail end-to-end latency* when the broker service relays messages to distant devices. If a clone can measure: 1) messaging demand with other devices/clones; 2) the tail latency experienced by messages; and 3) the potential latency of other cloudlets/cloud platforms, clones can self-migrate between cloud platforms to always ensure a bounded weighted tail end-to-end latency. We show how autonomous clone migration can mimic birds flocking and we capitalize over our proofs in [6] that it is stable and achieves a tight minimal latency that is $(1+\epsilon)-$far from optimal.

The use of cloudlets and dynamic service migration to solve latency problems are not new: Cloudlets [7] reduce the single-hop latency from 0.5 to 1 s to tens of milliseconds, and technologies like MobiScud and FollowMe [8], [9] migrate clones to sustain an average single-hop round trip time (RTT) at nearly 10 ms. Such schemes struggle to make optimal migration decisions despite using central control units as they: adopt too constraining migration metric (average single-hop latency) and trigger migration only if devices locations change [10]. However, applications in fog computing [4] necessitate the deployment of internetworking clones in heterogeneous platforms (cloudlet/clouds) without centralized administration. In this fog environment, clones communicate with several geo-distributed devices and other clones where the *tail weighted end-to-end latency*—that considers the 99th percentile of computation latency plus the communication latency between clones/devices—becomes the primary latency measure instead of the average RTT of a single-hop.

We design self-deploying brokering clones that discover cloud hosting platforms and autonomously migrate between them according to self-measured weighted tail end-to-end latency, ultimately allowing us to stabilize clones deployment and achieve a near minimum latency given an existing infrastructure limits.

We implement FogMQ as a simple Linux service that provides three key features.

Manuscript received September 18, 2017; revised January 23, 2018 and February 25, 2018; accepted March 2, 2018. Date of publication March 20, 2018; date of current version June 8, 2018. This work was supported by an NPRP grant from the Qatar National Research Fund (a member of Qatar Foundation) under Grant NPRP 5-319-2-121. *(Corresponding author: Sherif Abdelwahab.)*

The authors are with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: abdelwas@oregonstate.edu; zhangso@oregonstate.edu; greenaca@oregonstate.edu; ovesenk@oregonstate.edu; bergmank@oregonstate.edu; hamdaoui@oregonstate.edu).

Digital Object Identifier 10.1109/JIOT.2018.2817392

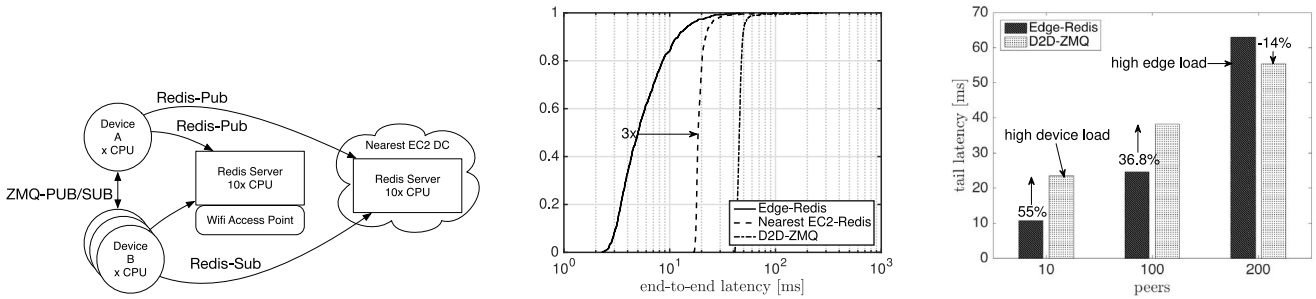

Fig. 1. Motivating experiments: the sources of latency in devices publish/subscribe communication. Using message brokers near the edge improves latency by 2× as-long-as the hosting node is not loaded. (a) Experiments setup. (b) CDF of latencies. (c) Tail latency in D2D versus edge brokers.

1) It reduces the successive queuing problem by ensuring a bounded *tail weighted end-to-end latency* and a rapid adaptation to shared hosting nodes and network variations and *without sacrificing computation offloading gain* in cloud platforms.
2) It autonomously discover and migrate to heterogeneous cloud/edge platform in an Internet-scale system *without the need of a central monitoring and control unit*.
3) It is simple and *requires no change* in existing cloud platforms controllers.

II. Motivation and Challenges

We first show that multihop queuing along Internet paths is a major source of end-to-end latency for IoT applications. We then show that devices' limited compute and memory resources standstill against latency reduction by direct device-to-device communication.

A. Sources of Latency

When we host a message broker in a multitenant cloud platform, the end-to-end latency degrades as *network interference* delays the broker's messages. Network interference occurs when the broker messages share: ingress/egress network I/O of its host, and one or more queues in the data-center network switches [11]. Hosts and network resources become spontaneously congested by traffic-demanding applications. For a single-authority cloud, an operator can control network interference of latency-sensitive applications with switching, routing, and queue management policies besides controlling contention for hosts' compute, memory, and I/O resources [12], [13].

Network interference is harder to control for IoT applications. As devices communicate using cloud-hosted brokers, messages share network resources of multihop paths with diverse and unmonitored traffic. Multiple, unfederated authorities manage network resources along these paths, which make it hard to enforce *unified* traffic shaping or queuing management policies. Adding also variations in devices traffic demand, communication pattern with other devices, and mobility it becomes particularity hard to trace devices traffic, delays, and infrastructure conditions to find optimal policies with centralized solutions. Multihop queuing along Internet paths can account for a 3× degradation in end-to-end latency on average.

TABLE I
Median and 99th End-to-End Latency of Redis and ZeroMQ Measured Under Different Loads (Number of Messages)

Benchmark	50%	$99^{th}\%$
Redis, 1000 messages	523.2	$1,276.0\mu s$
ZeroMQ, 1000 messages	314.8	$647.6\mu s$
Redis, 10,000 messages	620.1	$2,010.5\mu s$
ZeroMQ, 10,000 messages	320.3	$652.9\mu s$

Fig. 1(a) illustrates experiments to quantify this latency degradation. We install a Redis server in a virtual machine (VM)-instance in the nearest Amazon data-center (EC2-Redis) and install another Redis server in a same capacity VM in a host that is co-located with our WiFi access point (Edge-Redis). Our host runs other workloads. We emulate devices as simple processes running on another host that uses the same access point. We ensure that all VMs and hosts are time-synchronized with zero delays and jitter *during the experiment execution time*. A device emulator *A* publishes 10K messages to either Redis servers and another device emulator *B* subscribes to *A*'s messages. Fig. 1(b) shows the cumulative distribution function (CDF) of the end-to-end latency measured as the time between receiving a message at *B* and publishing it from *A*. The tail end-to-end latency for Edge-Redis is 15.6 ms, while it measured at 24.2 ms for EC2-Redis accounting for 1.5× tail end-to-end latency improvement by avoiding the multihop path to the closest EC2 instance and 3× improvement on average.

B. Why Broker-Less Is Not Always the Answer?

Direct device-to-device communication using broker-less message queues can be thought to be better than using message brokers. The obvious reasons for broker-less queues superiority are their lightweight implementation, and usage of a minimal number of shared queues, switches, routers, and access points, between communicating devices. Table I shows the median and tail end-to-end latency of ZeroMQ and Redisunder different loads, where ZeroMQ can deliver 10 000 messages three times faster than Redis.

Unfortunately, if the devices are resource limited, the latency superiority of direct device-to-device communication

is not always the case. We return to our motivating experiment in Fig. 1(a). We limit the resources used by the devices emulators using Linux cgroups such that a device emulator can use no more than 10% of the CPU time compared to EC2-Redis or Edge-Redis. Fig. 1(b) shows that the average end-to-end latency of D2D-ZeroMQ is seven times longer than Edge-Redis, and the tail end-to-end latency is four times longer. Several factors can contribute to this deteriorated performance including the wireless environment loading and implementation details of either Redis or ZeroMQ. However, the main factor that limits direct device-to-device latency is the limited compute resources of the devices emulators.

To emphasis this observation, we increase the number of the publishing device peers (i.e., number of subscribing device emulators) until the Edge-Redis server becomes loaded. Fig. 1(c) shows the tail end-to-end latency for different number of peers. As we increase the number of peers, the latency superiority of the Edge-Redis starts to diminish, until we reach the 200 peers points at which our host becomes loaded at 90% utilization and the tail end-to-end latency of broker-less D2D-ZeroMQ becomes better by 14%. Broker-less device-to-device messaging is only better if a device computational resources are sufficiently large, which is an unrealistic assumption for most IoT devices.

III. FogMQ System Design

Our motivating experiments show that multihop queuing along Internet paths is a major source of tail latency for cloud-based messaging systems and that the latency improvement promise from device-to-device communication cannot be always attained due to limited devices resources. FogMQ tackles multihop queuing by reducing the queuing of messages. Primarily, if message brokers can self-deploy and migrate across cloud platforms (from edge to cloud and including cloudlets) according to the communication pattern of the devices, then we will diminish the impact of multihop queuing delay. In the extreme case, if two resource-limited devices communicate through brokers in the same unloaded host and using the same access point, we can achieve a finite minimal bound on the latency.

In this section, we design FogMQ by which we bound the weighted end-to-end latency of devices' clones in FogMQ given an arbitrary network of heterogeneous cloud platforms. Although the stability and bounded performance of our design is intuitive, we solidify this intuition by relating the design to the theory of singleton weighted congestion games [14]–[16], where we show that self-deploying clones reach a Nash equilibrium (NE) and it tightens the price of anarchy (PoA) of the weighted end-to-end delay.

A. Social Networks of Clones

To begin, we assume that devices communicate with each other according to IoT applications requirements and form a social network of devices. Typically, the convergence of man–machine interactions in IoT will drive devices to form a social network [17]. This network can form according to existing social network structure of devices' users or according to the required communication between devices that is inherited from application design.

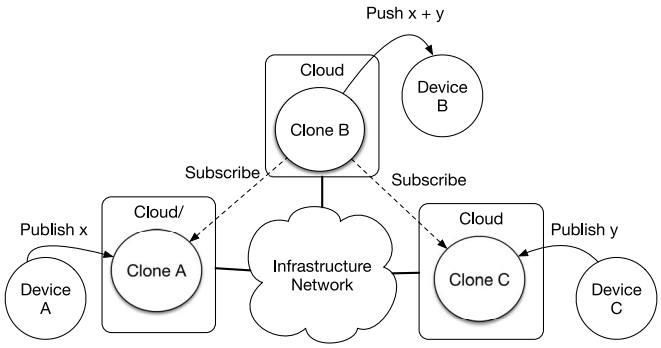

Fig. 2. Example overlay aggregation tree formed by devices clones.

The idea of modeling IoT analytics applications as social networks is simple. Applications are modeled as graphs (e.g., [18]–[20]) of internetworked microservices, nanoservices, VMs, or containers. Devices contribute to the execution of an IoT application by *publishing* their data to a brokering clone of the device. Other devices that are participating in the execution of the application also *publish* their information to their clones. On the other hand clones *subscribe* to each other according to the application modeled graph which forms an overlay network of clones that execute the application within resource-rich compute and network nodes. Upon completion of the executions clones may *push* the results back to devices and the application users. Fig. 2 illustrates a simple tree aggregation application for data retrieved from three devices. Device-A and Device-C publish their data, x and y, to their clones. Clone-B subscribes to data from clone-A and clone-C, evaluates $x + y$, and pushes the result to device-B.

The pub/sub communication pattern provides an efficient messaging middle-ware for applications modeled as large-scale graph structures. We can rely on already in-place subscription and matching languages to effectively route information between devices and clones, and interclone. Existing pub/sub systems also simplify addressing and clone authentication, hence the design of large-scale applications as overlay network between the clones.

Generally, the overlay network design of the clones is either structured or unstructured and focus mainly on reducing clones fanout to minimize the communication overhead between the clones. For example, topic-connected overlays are designed such that devices interested in the same topic are organized in a dissemination overlay [21]. Overlay network design forms the foundation for distributed pub/sub and directly impact scalability and applications performance [22]–[24]. We assume that an overlay topology of clones is given and we model it as a social network of clones. We model the network of clones as a graph $G = (V, P)$, where V denotes the set of n clones and P denotes the set of clone pairs such that $p = (i, j) \in P$ if the ith and jth clone communicate with each other.

B. FogMQ Architecture

FogMQ consists of a management-plane, control-plane, and data-plane. The management-plane comprises nanoservices

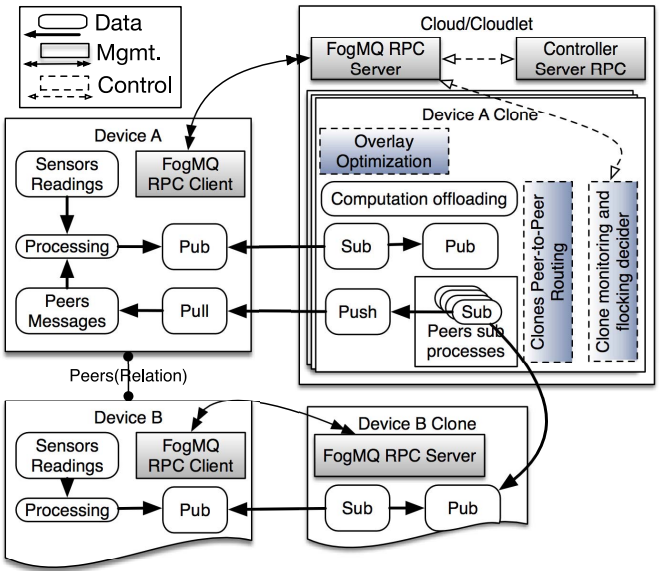

Fig. 3. FogMQ architecture.

and clients for devices onboarding, clone creation, and cloud-network tomography. The control-plane consists of clone monitoring and migration, overlay optimization, and peer-to-peer routing functions. The data-plane uses the publish/subscribe communication pattern for messaging between devices and their clones as well as interclone communications. Fig. 3 illustrates the high-level architectural elements of FogMQ and its management, control, and data modules.

FogMQ initially clone a device at the closest cloud/cloudlet from a set A of m cloud/cloudlets that are available to all devices and that can communicate over the Internet. Association with nearest brokering server is a standard approach in existing cloud-based brokers. An remote procedure call (RPC) client in the device is responsible for the device registration, and peer relationship definition with other devices by which a device participates in the execution of a distributed application. A typical approach that clients can use to initiate clones is to query a global geo-aware domain name service (DNS) load balancer to retrieve the Internet protocol (IP) address of the nearest FogMQ RPC server. With the integration of cloud computing in cellular systems [25], devices can also use native cellular network procedures to initiate clones in the nearest cellular site to the device.

The FogMQ RPC server realizes the device clone in a cloud platform as a VM, container, or native process, where processes are always a favorable design choice to avoid virtualization overhead. Recent Linpack benchmark [26] shows that containers and native processes can achieve a comparable number of floating point arithmetic per second that is at least 2.5× greater than VM. Despite that containers have a better advantage for privacy and security, as they provide a better administration, network, storage, and compute isolation, containers networking configuration can account for 30μ s latency overhead compared to a minimal network configuration of native processes [27]. As we will discuss later implementing clones as processes has an advantage over both VMs and containers as the incur minimal migration overhead in our design.

Once FogMQ creates a clone for a device, the clone subscribes to the devices published messages that contain preprocessed sensors reading. Subscribing to the devices messages eases the clone migration processes as we will detail later. If a clone migrates from one cloud to the other, any changes to the clone's IP address or network configuration become transparent to the device. Upon migration, the clone resubscribes to the devices messages, allowing the device to continue publishing its messages without the need to notify the device of its clone migration.

A device's clone can process the devices message in its computation offloading module (see Fig. 3) with high processing, memory, and storage capacities. The computation offloading module of the clone also executes distributed applications defined as overlay networks that interconnect several clones. To exchange messages between peer-clones of an overlay network, a clone creates a separate process to process its peer messages. Each process subscribes to the published messages from its corresponding peer-clone to make messages from peers available for the computation offloading module. If needed a clone pushes messages and/or computation results back to its device using a push/pull messaging pattern. Fig. 3 illustrates the messages flow between different modules for a simple example in which Device-*A* is a peer to Device-*B*.

The overlay optimization module and the peer-to-peer routing module are responsible for optimizing the fan-out of overlay networks and the routing decisions. Although the design of these mechanisms is integral to the performance of the overall system, overlay design, and routing optimization algorithms are orthogonal to the scope of this paper as we focus on autonomous migration decisions that minimize the tail end-to-end latency.

C. Cloud Network Tomography

An administrator runs FogMQ servers as application middle-wares that have network tomography functionalities. In addition to managing the clones, the network tomography functions assist the clones to measure and discover their local and potential hosting platforms, hence autonomously decide their optimal hosting clouds. The tomography function consists of measuring the average processing delay of a clone's host and the network latency between the hosting cloud and any other target cloud. It also comprises a function to discover potential cloud platforms.

1) Discovering Cloud Platforms: FogMQ servers in different clouds form a peer-to-peer network that can autonomously evolve and discover each other. A FogMQ server discovers other FogMQ servers explicitly in a bootstrapping phase and implicitly as it creates and manage clones. Once a FogMQ server starts, it register with bootstrapping nodes that also authenticate the server. If a bootstrapping node permits the server to join the network of FogMQ servers, the server becomes entitled to query any bootstrapping nodes to learn about other FogMQ servers that already exist.

Additionally, FogMQ implicitly learn about other FogMQ servers from clones communication sessions. For two clones to communicate, each clone needs to learn the IP address of the hosting node of the other clone by querying a clones registry (e.g., distributed key-value store). The server proxies such control queries, and from the query responses it can learn about new IP addresses of other hosting platforms and query the bootstrapping nodes for more information if needed.

2) Processing Delay and Network Latency: The FogMQ servers are also end-points that measure the average processing delay and network latency of the hosting cloud and of any other cloud. Processing delay of any cloud platform is a function of the utilization of the node that hosts the clone. A FogMQ server can programmatically query the utilization of VMs using existing cloud platforms Application program interfaces (APIs). Network latency can be actively measured by ICMP protocols, or passively measured by logging the RTT statistics of ongoing transmission control protocol sessions.

Each device clone self-monitors and characterizes the demands with its peers and evaluates latencies with the assistance of the FogMQ server. For clones, i and j, let $d_{ij} \in \mathbb{R}^+$ denote the traffic demand between i and j and assume that $d_{ij} = d_{ji}$. Let $x_i \in A$ denote the cloud that hosts i and let $l(x_i, x_j) > 0$ be the average latency between i and j if they are hosted at x_i and x_j, respectively (*Note:* if i and j are hosted at the same cloud $x_i = x_j$). We assume that l is reciprocal and monotonic. Therefore, $l(x_i, x_j) = l(x_j, x_i)$ and there is an entirely nondecreasing order of $A \rightarrow A'$ such that for any consecutive $x_i, x_i' \in A'$, $l(x_i, x_j) \leq l(x_i', x_j)$. The reciprocity condition ensures that measured latencies are aligned with peer-clones and imitates the alignment rule in bird flocking. FogMQ servers align the measured latencies such that x_i and x_j use the same values of $l(x_i, x_j)$. We model $l(x_i, x_j) = \tau(x_i, x_j) + \rho(x_i) + \rho(x_j)$, where $\tau(x_i, x_j)$ is the average packet latency between x_i and x_j, and $\tau(x_i, x_j) = \tau(x_j, x_i)$. The quantity $\rho(x)$ is the average processing delay of x modeled as: $\rho(x) = \delta \sum_{i \in V: x_i = x} \sum_{j \in V} d_{ij} / (\gamma(x) - \sum_{i \in V: x_i = x} \sum_{j \in V} d_{ij})$, where δ is an arbitrary delay constant and $\gamma(x)$ denote the capacity of x to handle all demanded traffic of its hosted clones. An increased value of $\rho(x_i)$ signals the clone i that it is crowding with other clones in the same cloud which is imitating the separation rule in bird flocking.

D. Clones Shall Flock

To resolve the successive queuing problem, we design a simple protocol by which a clone autonomously decides its hosting cloud using only local network tomography provided by its hosting FogMQ server. Clones greedily minimize their perceived weighted latency. In this section, we analytically show that this protocol is stable and close to optimal. In Section IV-E, we demonstrate these properties when we functionally evaluate FogMQ. A clone i evaluates its weighted latency with its peers if hosted at x as

$$u_i(x) = \sum_{j \in V} d_{ij} l\big(x, x_j\big) / \sum_{j \in V} d_{ij}. \tag{1}$$

Algorithm 1 Flock: Autonomous Clone Migration Protocol

Initialization: Each clone $i \in V$ runs at a cloud $x \in A$.
Ensure: A Nash equilibrium outcome σ.
During round t, do in parallel $\forall i \in V$
Greedy migration process imitating cohesion in flocking:
i solicits its current strategies A_i from x.
i randomly selects a target cloud $y \in A_i$.
if $u_i(y) f(w_y + u_i(y)) \leq \eta u_i(x) f(w_x - u_i(x))$ **then**
  $x \xrightarrow{i} y$.
end if

Our objective is to design an autonomous clone migration protocol that converges to an outcome $\sigma = (x_1, x_2, \ldots, x_n)$ that minimizes the sum of weighted latency $\sum_{i \in V} u_i(x_i)$. That is to say, σ maximizes the responsiveness of all clones given their peer-to-peer communication pattern (i.e., connectivity and demands).

A clone i learns a set $A_i \subseteq A$ from its FogMQ server. The set A_i is called the strategies set of i. For a cloud x, FogMQ server evaluates its current weight $w_x = \sum_{i: x_i = x} u_i(x)$ and advertises a monotonic non-negative regularization function $f(w_x) : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, such that $\alpha < f(w_x) < 1$ for $\alpha > 0$, to all the clones that are hosted at x. As any clone is hosted by a single cloud and all clones have access to the same strategies set, we model the migration problem as a singleton symmetric weighted congestion game that minimizes the social cost $C(\sigma) = \sum_{x \in A} w_x f(w_x)$. If $f(w_x) \approx 1$, this game model approximates to minimizing $\sum_{i \in V} u_i(x_i)$. Let $x \xrightarrow{i} y$ denote that a clone i migrates from cloud x to cloud y and let $\eta \leq 1$ denote a design threshold. We propose the migration protocol in Algorithm 1.

We prove that Flock converges to an NE, where each clone chooses a single cloud (strategy) and no clone has an incentive (i.e., less average latency) to migrate from its current cloud. Then, we derive an upper bound on Flock's PoA for general regularization functions, f, following a similar approach to [16]. Finally, we propose a regularization function $f(w_x) \approx 1$ that achieves a tight PoA of at most $1 + \epsilon$.

Theorem 1: Flock converges to an NE outcome. Proof outline—We show that any step of Flock reduces the social value $C(\sigma)$ and $C(\sigma)$ is bounded below (see [6] for proof).

Lemma 1: The social value of Flock has a perfect PoA at most $\lambda/(1 - \varepsilon)$, if for $\varepsilon < 1$ and $\lambda > 1 - \varepsilon$ the regularization function satisfies $w^* f(w + w^*) \leq \lambda w^* f(w^*) + \varepsilon w f(w)$, where $w \geq 0$ and $w^* > 0$. See [6] for proof.

Theorem 2: The regularization function $f(w) = \exp(-1/(w + a))$ tightens the PoA to $1 + \epsilon$ for a sufficiently large constant a and reduces the game to the original clone migration problem, i.e., minimizing $\sum_i u_i(x_i)$. See [6] for proof.

IV. FogMQ Prototyping and Evaluation

We prototyped FogMQ and evaluated it in a virtual testbed in AWS cloud. We implemented all FogMQ components as nanoservices that communicate using ZeroMQ (source code can be obtained from [28]). We deployed FogMQ and the

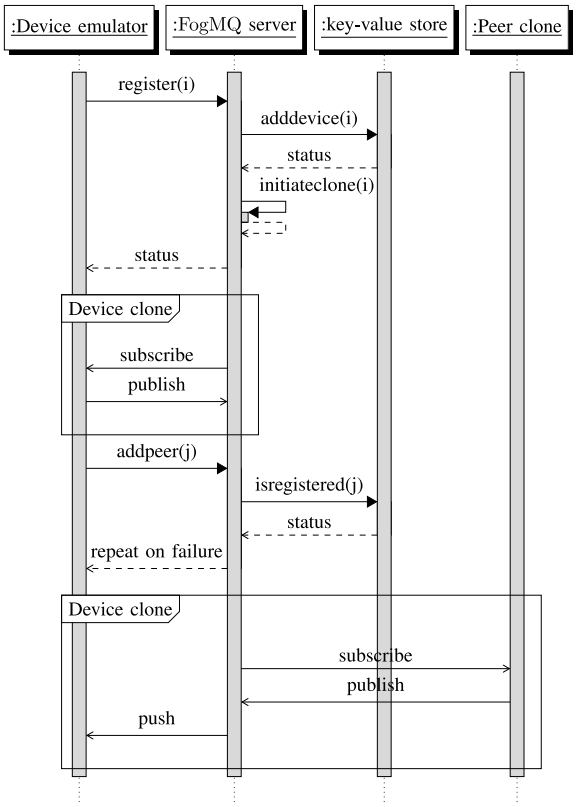

Fig. 4. Device emulator onboarding, cloning, and adding peers.

device emulators in several geographically distributed VMs in all Amazon cloud regions and availability zones. Our objective from this prototyping is to have a reference implementation of FogMQ by which we test its functionality and verify the stability and near optimality properties in a shared cloud and Internet environment.

A. Devices on-Boarding and Overlay Setup

In our prototype, we have emulated IoT devices as Linux processes that send messages according to a Poisson distribution with a random average interarrival time of 1–30 s. Each device emulator has a random unique ID. We used a Redis to register the devices emulators and to track the clones deployments across different clouds.

Fig. 4 shows the device emulator initialization sequence. A device emulator registers itself with the nearest FogMQ server. The emulator uses its ID, host IP address, and ZeroMQ publish and pull ports for the registration. The FogMQ server adds the device information to the Redis key-value store. This step can also be extended for devices authentication. Once registered, the FogMQ server starts a clone for the device and acknowledge the device that it is ready to add peers. The clone subscribes to the devices ZeroMQ publish port.

Whenever a message becomes available to the device from one of its peers, the clone pushes the message to the device pull port. The devices peer-to-peer relationships—in our evaluation—are determined based on real social network dataset from [29]. Devices emulators adds their peer at the clone side using only the peer device ID. The device clone

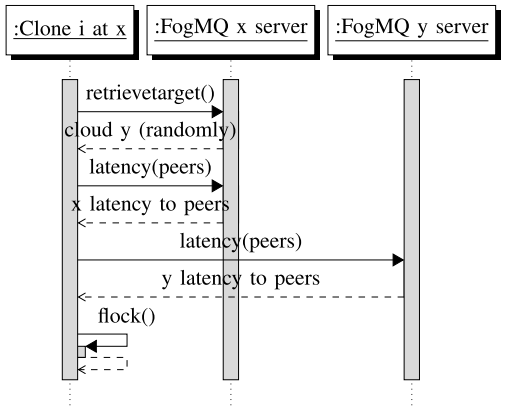

Fig. 5. Flock sequence diagram.

retrieves the peer clone host IP from the key-value store. If the peer was not registered in the key-value store, the device clone instructs the device to back-off for a randomly chosen timeout before retrying to add the peer. Otherwise, the peer sub processes subscribes to messages from the peer clone to establish the clone-to-clone overlay communication.

B. Flock and Clones Migration

We now detail how we implement Flock using nanoservices exposed by FogMQ servers and how clones autonomously and efficiently migrate between cloud platforms. Each clone periodically executes the sequence diagram that we illustrate in Fig. 5. In our implementation, we arbitrarily choose the execution period to be 30 s and it is left as a design parameter that can be configured to tune how rapid clone migration shall be.

To execute Flock , a clone i retrieves a potential target cloud y, from its FogMQ server running at its hosting cloud x. Then i executes two RPCs with both FogMQ servers at x and y to determine the current and the potential latencies with its peers. When a server receives a *latency* RPC for a peer j, it answers with previously performed latency measurements with the cloud hosting j, or perform active measurements to determine the estimated latency. Finally, the clone executes Flock to decide if a migration shall be performed or not.

If the migration condition in Flock is satisfied, the clone i sends its meta-data an RPC to x requesting to migrate to y as illustrated in Fig. 6. The server x forwards i's meta-data to y to initiate a clone for device-i at y. The server y replaces the current hosting cloud of i to y and the new clone at y subscribes to i's messages. Finally, server y instructs x to terminate its migrated clone.

C. Negligible Migration Overhead

Fig. 6 illustrates the migration sequence diagram and Fig. 7 shows the total time needed for clone migrations as Flock continues execution over time. By only transferring the meta-data, which is less than 140 bytes in size, we ensure that clone migration overhead is minimum. Alternatively, one can take a snapshot of the clone at x, transfer the entire clone memory, and restart the clone at y. This can resemble a significant overhead as we could transfer unnecessary information. As shown

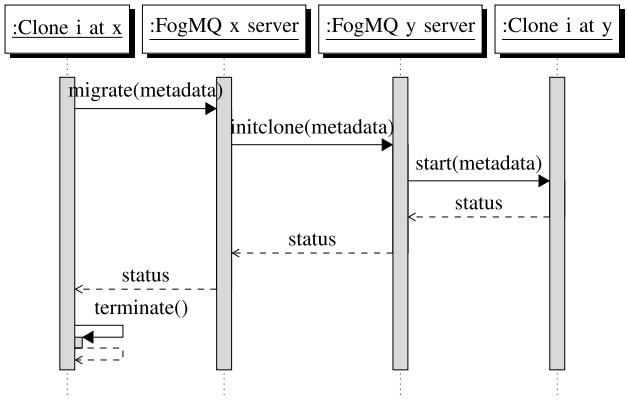

Fig. 6. Clone migration sequence diagram.

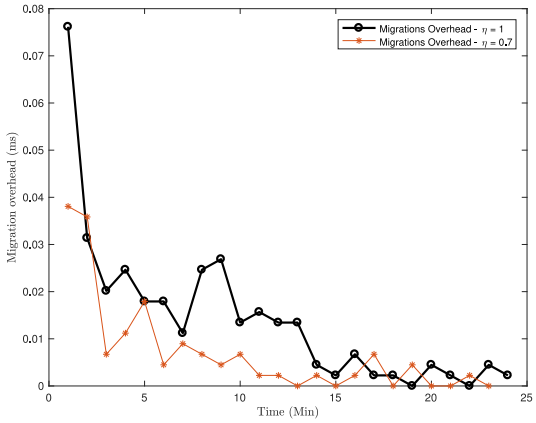

Fig. 7. FogMQ migration overhead.

in Fig. 7, the parameter η can have an impact on the migration overhead at initial execution where higher values of η eases the migration decisions. Moreover, the use of the publish/subscribe communication here makes such migration transparent to device i, and prevents potential data loss.

D. How to Prevent Data Loss During Clone Migration

Although making new clone to device subscription before breaking the old one prevents data loss from a device to FogMQ, migration can still results in data loss between peers. First, the peer processes that originally subscribed to i's clone at x needs to detect its termination and resubscribe to the new clone at y. Second, the new clone needs to enforce a history quality of service, by which the new clone retains devices data until the peer processes subscribes to the new clone and retrieve all historical data. The solution to the last problem is straight forward. FogMQ implementation allows an administrator to configure FogMQ to either keep all devices data in the new clone until delivery to peer processes, or to keep a configurable number of messages (either by number or duration).

To solve the first problem, we need two mechanism to detect the termination of the original clone and discover the new hosting cloud of the new clone. We use in-place connection-monitoring in ZeroMQ to detect clone termination. A main thread monitors clones' connections. If the monitoring thread

TABLE II
TESTBED DISTRIBUTION IN AMAZON REGIONS

Amazon Regions	Virtual Machines
ap-northeast-1	4
ap-southeast-1	8
eu-west-1	6
sa-east-1	4
us-east-1	8
us-west-1	4
us-west-2	7

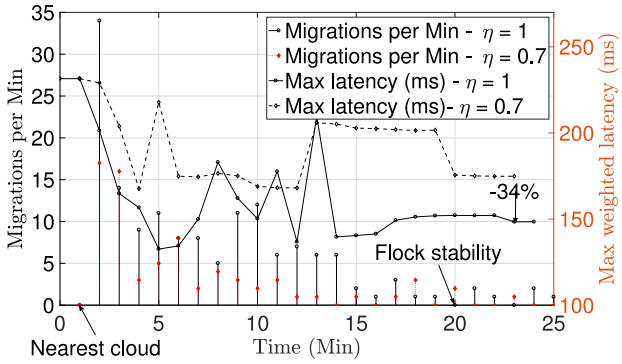

Fig. 8. FogMQ minimizes weighted average latency from 235 ms to 150 ms (34%) and maintains a stable latency at its minimal value.

sees an EVENT_DISCONNECT event, it sends a RESET signal to the peer process. As the peer process receives a RESET signal, it queries the current host from the key-value store, and attempts to resubscribe to the clone at the new hosting cloud.

E. Experimental Results

We deployed our prototype in a geographically distributed testbed comprising 41 VMs running in Amazon regions as shown in Table II. One VM is running a Redis key-value store to maintain experiments logs and a bind DNS server. Twenty VMs are running FogMQ, and each of the remaining VMs runs five device emulators. We generated the peer-to-peer relationships of the hundred device emulators as a subset of the Facebook dataset obtained from [29].

Our *functional evaluation* validates the stability of FogMQ given the flocking of clones and its capability to subdue to end-to-end weighted latency of the clones [defined in (1)]. The use of geographically distributed virtual testbed challenges FogMQ's stability and optimality given the real variability and uncertainty of the Internet and the hosting cloud conditions. We also show the advantage of using Flock in FogMQ over the widely used architecture in which devices relay messages through the nearest broker.

Fig. 8 shows a 30 min experiment. Initially, all device emulators register with the nearest FogMQ server by querying the DNS server. The Flock is activated at minute 1, where all servers verify that all device emulators are registered from the Redis server. At this minute, clone starts to autonomously migrate between the server and the secondary access show the total migrations per minute. The number of migrations decrease rapidly and by minute 20, all clones reside in a FogMQ server where they cannot improve their measured latency anymore. This is a stability point of Flock. After

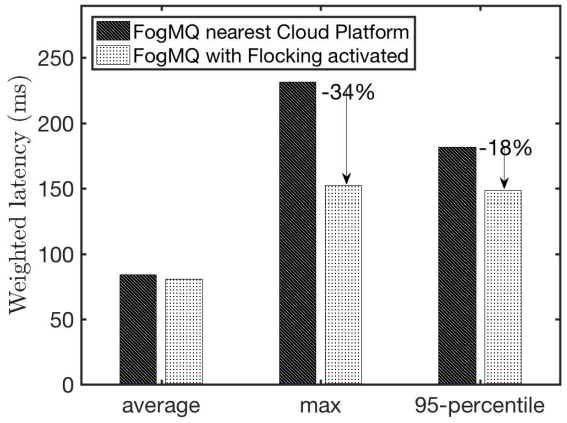

Fig. 9. FogMQ improves the tail weighted latency by 34%, while having no effect on average latency.

minute 20, we notice some clones perform migrations in which they quickly respond to variations in the system.

Fig. 9 shows the summary statistics of the weighted latency comparing the initial deployment and the stability point of FogMQ. The average latency is not significantly improved. On the other hand the tail end-to-end latency shown as the maximum latency and the 95-percentile latency has improve by 34% and 18%, respectively. On the secondary axis of Fig. 8, we show the maximum weighted latency measured by clones. As Flock progresses, also stabilizes at 150 ms compared to 231 ms initially. This stable behavior demonstrates the capability of Flock to maintain tail latency under tight control, which we theoretically proved that it is no far than $1 + \epsilon$ from an optimal.

V. Related Work and Summary of Contribution

We propose autonomous brokering clones for designing large-scale distributed pub/sub systems as a major mechanism that complements existing techniques to minimize the tail end-to-end messaging latency.

A. Distributed Message Brokers

Researches focus on three main techniques for the development of simple, scalable, and resource economic message brokers and pub/sub systems: 1) content-centric above layer-3 routing between brokers (e.g., [30]–[34]); 2) overlay brokers network topologies designs (e.g., [23] and [35]–[38]); and 3) content-centric in-network caching (e.g., [39]–[41]).

Distributed pub/sub systems organize brokers, devices, or routing functions as an overlays and suboverlays at the application layer. Upon constructing an efficient overlay network, routing protocols above layer-3 build minimum-cost message dissemination paths to deliver messages to subscribers according to specific topic-interest. Caching policies replicate clones' contents closer to devices interested in a content for faster repetitive publishing. For a given routing, overlay topology, and caching mechanisms, FogMQ ensures that these mechanisms achieve their full potentials by self-reorganizing the deployment of brokers through migrations in heterogeneous, unmanaged, and dynamic cloud environments. Unlike widely adopted centralized systems (e.g., Redis), FogMQ suits the large-scale applications and use cases of IoT and avoids the limitations of broker-less systems (e.g., ZeroMQ).

B. Algorithms for Service Migration

Existing migration solutions are limited in their applicability to minimize the weighted end-to-end latency. Several existing solutions rely on a system-wide central controller to manage the states of clones, devices, and physical resources of cloud platforms [10], [42]–[45]. For the considered fog environment, these solutions lack scalability for an Internet-sized network without relaxations that potentially compromise solutions quality and challenging in practice to enforce a consistent state in a centralized controller [46], [47]. In FogMQ the state information are local to every single clone and does not require a global shared state.

Consider Markov decision process (MDP) based solutions. MDP requires a central server to collect statistics of devices mobility, clones demands, and clouds connectivity and utilization. This server also executes the value iteration algorithm to evaluate an optimal migration policy [10], [42], [48]. It is intractable to model all possible states of clones and their hosting platforms; hence it is common to discretize states measurements to relax the complexity of the policy optimization algorithms [10], [48]. This compromises the solutions quality.

Game-theoretic approaches potentially decentralize the migration algorithms and improve their scalability. However, existing game-theoretic solutions provide an unbounded PoA [49]. We cannot use them—as is—and guarantee optimal or close to optimal weighted end-to-end latency; finally, existing migration solutions serve specialized cloud providers objectives (e.g., energy, load, and cost) to profitably manage providers' infrastructure [49], [50].

Unlike the work proposed in [51], FogMQ focuses on delay-optimum device-to-device middle-ware communication without requiring knowledge or organization of the underlaying infrastructure. FogMQ effectively allocates compute and network resources to the devices' clones by autonomous migration decisions of the clones themselves without any requirement on: federation, centralization, or decision consensus. Given the design requirement and scope of FogMQ, we adopt the delay minimization objective of Flock. However, we have shown in [6] that Flock objective can be tuned to achieve other goals such as energy consumption minimization and load-balancing similar to [52] and [53].

This paper provides a generalized clone-to-clone as well as clone-to-device model compared to [43], [52], and [54]. On another hand, the system models in [43]–[45] and [52] are based on a one-to-one relationship between the compute resource and the device which does not suit applications that requires communication between the compute resources themselves. Moreover, generic stochastic algorithms (such as ant-colony or particle swarm optimization [44], [45]) do not provide the numerical properties that allow us to simplify the modeling of the delay, dynamics, and structure of the network of clones. We find it impractical to adopt a generic stochastic algorithm in the design of a proof-of-concept that requires

simple management of state information at scale and provides latency and stability guarantees, which is a primary motivation for us to develop a fully distributed algorithm like Flock which we used in our FogMQ implementation.

The existing models do not capture network latency between clones that are executing distributed IoT applications. Unlike existing solutions FogMQ adopts a simple autonomous migration protocol that is stable and bounds the tail end-to-end latency $(1-\epsilon)$ far from optimal.

VI. Conclusion

There are many message brokers, cloud offloading, and communication middle-wares that support IoT analytics. We propose FogMQ, a message broker and cloning system for scalable and geographically distributed analytics near and at the network edge. FogMQ uses Flock to enable device clones to autonomously migrate across heterogeneous cloud/edge platforms. FogMQ servers expose tomography functionalities that enables devices clones to take migration decisions without complete knowledge about the hosting platform. This feature suites the deployability and scalability challenges across wide area networks and the Internet. We discuss the prototyping of FogMQ and its evaluation of a virtual and geographically distributed testbed using public cloud resources. In our functional evaluation, we demonstrated the stability of Flock, and the ability of FogMQ to improve the tail latency by one third of that attained by a widely used architecture in which device communicate through the nearest message broker.

References

[1] M. Anvari *et al.*, "The impact of latency on surgical precision and task completion during robotic-assisted remote telepresence surgery," *Comput.-Aided Surgery*, vol. 10, no. 2, pp. 93–99, 2005.

[2] U. Ramachandran *et al.*, "Large-scale situation awareness with camera networks and multimodal sensing," *Proc. IEEE*, vol. 100, no. 4, pp. 878–892, Apr. 2012.

[3] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, "It's time for low latency," in *Proc. HotOS*, vol. 13. Napa, CA, USA, 2011, p. 11.

[4] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[6] S. Abdelwahab and B. Hamdaoui, "Flocking virtual machines in quest for responsive IoT cloud services," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, 2017, pp. 1–6.

[7] M. Satyanarayanan *et al.*, "Cloudlets: At the leading edge of mobile-cloud convergence," in *Proc. IEEE 6th Int. Conf. Mobile Comput. Appl. Services (MobiCASE)*, Austin, TX, USA, 2014, pp. 1–9.

[8] K. Wang *et al.*, "MobiScud: A fast moving personal cloud in the mobile network," in *Proc. 5th Workshop All Things Cell. Oper. Appl. Challenges*, London, U.K., 2015, pp. 19–24.

[9] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep./Oct. 2013.

[10] R. Urgaonkar *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, pp. 205–228, Sep. 2015.

[11] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proc. 1st Annu. ACM SIGMM Conf. Multimedia Syst.*, Phoenix, AZ, USA, 2010, pp. 35–46.

[12] X. Pu *et al.*, "Who is your neighbor: Net I/O performance interference in virtualized clouds," *IEEE Trans. Services Comput.*, vol. 6, no. 3, pp. 314–329, Jul./Sep. 2013.

[13] R. C. Chiang and H. H. Huang, "Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. Int. Conf. High Perform. Comput. Netw. Stor. Anal.*, Seatle, WA, USA, 2011, p. 47.

[14] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis, "The structure and complexity of nash equilibria for a selfish routing game," in *Automata, Languages and Programming*. Heidelberg, Germany: Springer, 2002, pp. 123–134.

[15] D. Fotakis, S. Kontogiannis, and P. Spirakis, "Selfish unsplittable flows," *Theor. Comput. Sci.*, vol. 348, nos. 2–3, pp. 226–239, 2005.

[16] K. Bhawalkar, M. Gairing, and T. Roughgarden, "Weighted congestion games: The price of anarchy, universal worst-case examples, and tightness," *ACM Trans. Econ. Comput.*, vol. 2, no. 4, p. 14, 2014.

[17] L. Atzori, A. Iera, and G. Morabito, "SIoT: Giving a social structure to the Internet of Things," *IEEE Commun. Lett.*, vol. 15, no. 11, pp. 1193–1195, Nov. 2011.

[18] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the Internet of Things," in *Proc. 2nd ACM SIGCOMM Workshop Mobile Cloud Comput.*, Hong Kong, 2013, pp. 15–20.

[19] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella, "Graph sample and hold: A framework for big-graph analytics," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, New York, NY, USA, 2014, pp. 1446–1455.

[20] N. Satish *et al.*, "Navigating the maze of graph analytics frameworks using massive graph datasets," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2014, pp. 979–990.

[21] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "A generalized algorithm for publish/subscribe overlay design and its fast implementation," in *Distributed Computing*. Heidelberg, Germany: Springer, 2012, pp. 76–90.

[22] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, "Efficient publish/subscribe through a self-organizing broker overlay and its application to siena," *Comput. J.*, vol. 50, no. 4, pp. 444–459, 2007.

[23] C. Chen, H.-A. Jacobsen, and R. Vitenberg, "Algorithms based on divide and conquer for topic-based publish/subscribe overlay design," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 422–436, Feb. 2016.

[24] Y. Sun, X. Qiao, B. Cheng, and J. Chen, "A low-delay, lightweight publish/subscribe architecture for delay-sensitive IoT services," in *Proc. IEEE 20th Int. Conf. Web Services (ICWS)*, Santa Clara, CA, USA, 2013, pp. 179–186.

[25] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud," *IEEE Internet Things J.*, vol. 3, no. 3, pp. 327–338, Jun. 2016.

[26] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, present and future," *Concurrency Comput. Pract. Exp.*, vol. 15, no. 9, pp. 803–820, 2003.

[27] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Philadelphia, PA, USA, 2015, pp. 171–172.

[28] S. Abdelwahab. (2017). *FogMQ IoT Device Middleware in Edge Computing Software*. Accessed: Feb. 24, 2016. [Online]. Available: https://github.com/abdelwas/fogmq

[29] J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2012, pp. 539–547.

[30] C. Cañas, E. Pacheco, B. Kemme, J. Kienzle, and H.-A. Jacobsen, "Graps: A graph publish/subscribe middleware," in *Proc. 16th Annu. Middleware Conf.*, Vancouver, BC, Canada, 2015, pp. 1–12.

[31] N. Carvalho, F. Araujo, and L. Rodrigues, "Scalable QoS-based event routing in publish-subscribe systems," in *Proc. 4th IEEE Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, USA, 2005, pp. 101–108.

[32] K. An, A. Gokhale, S. Tambe, and T. Kuroda, "Wide area network-scale discovery and data dissemination in data-centric publish/subscribe systems," in *Proc. Posters Demos Session 16th Int. Middleware Conf. (Middleware Posters Demos)*, Vancouver, BC, Canada, 2015, p. 6.

[33] C. Chen and Y. Tock, "Design of routing protocols and overlay topologies for topic-based publish/subscribe on small-world networks," in *Proc. Ind. Track 16th Int. Middleware Conf.*, Vancouver, BC, Canada, 2015, p. 2.

[34] W. Rao, K. Zhao, Y. Zhang, P. Hui, and S. Tarkoma, "Towards maximizing timely content delivery in delay tolerant networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 4, pp. 755–769, Apr. 2015.

[35] G. Siegemund, V. Turau, and K. Maamra, "A self-stabilizing publish/subscribe middleware for wireless sensor networks," in *Proc. Int. Conf. Workshops Netw. Syst. (NetSys)*, Columbus, OH, USA, 2015, pp. 1–8.
[36] C. Chen, Y. Tock, H.-A. Jacobsen, and R. Vitenberg, "Weighted overlay design for topic-based publish/subscribe systems on geo-distributed data centers," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Columbus, OH, USA, 2015, pp. 474–485.
[37] Y. Teranishi, R. Banno, and T. Akiyama, "Scalable and locality-aware distributed topic-based pub/sub messaging for IoT," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, San Diego, CA, USA, 2015, pp. 1–7.
[38] M. Onus and A. W. Richa, "Minimum maximum-degree publish–subscribe overlay network design," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1331–1343, Oct. 2011.
[39] M. Diallo, S. Fdida, V. Sourlas, P. Flegkas, and L. Tassiulas, "Leveraging caching for Internet-scale content-based publish/subscribe networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kyoto, Japan, 2011, pp. 1–5.
[40] K. Cho *et al.*, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Orlando, FL, USA, 2012, pp. 316–321.
[41] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "Stan: Exploiting shared interests without disclosing them in gossip-based publish/subscribe," in *Proc. IPTPS*, 2010, p. 9.
[42] V. Eramo, E. Miucci, and M. Ammar, "Study of migration policies in energy-aware virtual router networks," *IEEE Commun. Lett.*, vol. 18, no. 11, pp. 1919–1922, Nov. 2014.
[43] X. Sun and N. Ansari, "Avaptive avatar handoff in the cloudlet network," *IEEE Trans. Cloud Comput.*, to be published. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7920303/
[44] L. Wang and J. Shen, "Multi-phase ant colony system for multi-party data-intensive service provision," *IEEE Trans. Services Comput.*, vol. 9, no. 2, pp. 264–276, Mar./Apr. 2016.
[45] Y. Gao, G. Zhang, J. Lu, and H.-M. Wee, "Particle swarm optimization for bi-level pricing problems in supply chains," *J. Glob. Optim.*, vol. 51, no. 2, pp. 245–254, 2011.
[46] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.
[47] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Chicago, IL, USA, 2014, pp. 1–6.
[48] L. Chen, H. Shen, and K. Sapra, "Distributed autonomous virtual resource management in datacenters using finite-Markov decision process," in *Proc. ACM Symp. Cloud Comput.*, Seattle, WA, USA, 2014, pp. 1–13.
[49] Z. Xiao *et al.*, "A solution of dynamic VMs placement problem for energy consumption optimization based on evolutionary game theory," *J. Syst. Softw.*, vol. 101, pp. 260–272, Mar. 2015.
[50] T. Duong-Ba, T. Nguyen, B. Bose, and T. Tran, "Joint virtual machine placement and migration scheme for datacenters," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Austin, TX, USA, 2014, pp. 2320–2325.
[51] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2082–2091, Dec. 2017.
[52] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," *IEEE Access*, vol. 5, pp. 21355–21367, 2017.
[53] W. Wang, Q. Wang, and K. Sohraby, "Multimedia sensing as a service (MSaaS): Exploring resource saving potentials of at cloud-edge IoT and fogs," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 487–495, Apr. 2017.
[54] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.

Sherif Abdelwahab (S'07–M'11) received the B.S. and M.S. degrees in electronics and communications engineering from Cairo University, Giza, Egypt, in 2004 and 2010, respectively, and the Ph.D. degree in electrical and computer engineering from the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA.

He was a technical lead in the mobile networks industry with Alcatel-Lucent, Cairo, Egypt, from 2004 to 2007 and Etisalat, Cairo, from 2008 to 2013. He is currently a Software Engineer with Amazon Web Services, Seattle, WA, USA. His current research interests include design, analysis, and optimization of networked systems enabling Internet of Things.

Sophia Zhang is currently pursuing the undergraduation degree in electrical and computer engineering at the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA.

Ashley Greenacre is currently pursuing the undergraduation degree in electrical and computer engineering at the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA.

Kai Ovesen is currently pursuing the undergraduation degree in electrical and computer engineering at the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA.

Kevin Bergman is currently pursuing the undergraduation degree in electrical and computer engineering at the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA.

Bechir Hamdaoui (S'02–M'05–SM'12) received the Diploma of Graduate Engineer degree from the National Engineering School of Tunis, Tunis, Tunisia, in 1997, and the M.S. degrees in both electrical and computer engineering and computer science and Ph.D. degree in electrical and computer engineering from the University of Wisconsin–Madison, Madison, WI, USA, in 2002, 2004, and 2005, respectively.

He is currently an Associate Professor with the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA. His current research interests include computer networking, wireless communications, and mobile computing, distributed optimization, cognitive and dynamic spectrum networking, cloud computing, and Internet of Things.

Dr. Hamdaoui serves/served on the Editorial Boards of several journals, and as the Program Chair/Co-Chair of many conferences. He has been selected as a Distinguished Lecturer of the IEEE Communication Society in 2016 and 2017.