Fog Operating System for User-oriented IoT Services: Challenges and Research Directions

Nakjung Choi, Daewoo Kim, Sung-Ju Lee and Yung Yi

***Abstract*—As the proliferation of mobile devices ignited cloud computing, it is expected that increasing development and deployment of IoT services will expedite the era of fog computing. Fog computing brings computing, storage, and networking even closer to end users and devices for services with better QoS. We introduce Fog Operating System (FogOS), a fog computing architecture for IoT services. We take the perspective of designing an operating system, practicing the architectural lessons from the long history of operating systems. We focus on addressing the challenges raised by (i) diversity and heterogeneity of IoT services and (ii) edge devices that are owned by individuals and different owners, and presenting how FogOS is designed to effectively and efficiently provide and manage such IoT services. We provide a city-scale surveillance use case to demonstrate FogOS in action.**

I. Introduction

Internet of Things (IoT) is no longer a vision, but a reality. We are already witnessing and experiencing many interesting IoT applications. Gartner predicts about 21 billion "things" across industry sections will be connected to the network by 2020 [1]. These devices generate and transmit data that have diverse requirements in terms of not only volume, but also variety and velocity.

With the ever increasing number of devices and data generated from the edge, classical cloud-based computing paradigm is faced with challenges, as IDC estimates that the amount of data analyzed on the IoT that are physically at or near the devices is approaching 40 percent [2]. To address these networking and computing trends, *fog computing* brings the cloud closer to the "things" that produce and act on IoT data. It is an architectural paradigm that is more appropriate for the fast-growing IoT as it brings computing, storage, and networking closer and faster to the edge devices.

We propose Fog Operating System (FogOS), a fog computing architecture for IoT services. We view the whole IoT ecosystem as a computer, and take the perspective of an operating system for our FogOS architecture. The role of traditional operating systems in computers is managing computer hardware and software resources and providing common services for computer programs. FogOS on the other hand, regards IoT applications (that correspond to programs in OS) as *X*-aaS (*X*-as-a-Service, e.g., "lightning"-as-a-Service, "temperature-sensing"-as-a-Service, etc) for which common interfaces are provided. The set of resources managed by FogOS include all fog and cloud (e.g., nano/edge cloud) and edge devices (e.g., sensors/actuators). They can also be connected to any level of cloud (e.g., central/regional/metro) when FogOS has an appropriate peering contract.

N. Choi is with Bell Labs, Nokia, Murray Hill, USA, e-mail: nakjung.choi@nokia-bell-labs.com.

D. Kim and Y. Yi are with the School of Electrical Engineering, KAIST, Daejeon, South Korea, e-mail: {daewookim, yiyung}@kaist.ac.kr.

S. Lee is with the School of Computing, KAIST, Daejeon, South Korea, e-mail: profsj@kaist.ac.kr.

Y. Yi is a corresponding author.

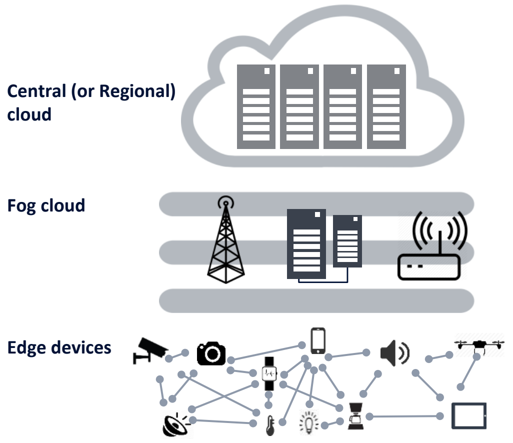

Fig. 1. Fog cloud and edge devices.

We particularly consider the case where edge devices are possibly owned by different individuals and providers. Many current IoT devices are deployed by infrastructure providers, but we argue that much more sensors/actuators will be increasingly individually-owned and shared in the future, when they are appropriately incentivised. In that scenario, there exists an economic, complex inter-play between different players, e.g., IoT users, IoT application providers, infrastructure providers, and edge device owners. FogOS hence can function as (i) a distributed operating system that manages the cloud and the resources at the *edge*, and (ii) a platform of incentivizing and connecting individually-owned edge devices [1].

OS for network resource management such as ONOS (Open Network Operating System) [5] is a seemingly similar concept to FogOS. For example, ONOS has been proposed as a control platform of SDN (Software Defined Networks) for carrier and cloud provider networks, with scalability, availability and performance in mind. However, there exists two main differences between FogOS and ONOS

First, ONOS operators directly own and control all network devices that are fixed with relatively stable operating conditions, while FogOS needs to control significantly diverse edge devices that are under "high dynamics" and owned by different

[1] 'Fog' typically means both clouds at edge and user devices in literature, but throughout this article, we use the terms of 'fog cloud' and 'edge device'.

TABLE I
CHALLENGES AND SOLUTIONS:FOG COMPUTING

Components (FogOS)	Task	Example func. in general OS	Challenges	Existing solutions
Service and device abstraction	Providing a device data model	File as a universal resource identification	Diversity in devices	Device data model in [3], [4]
	Providing service APIs	System calls & standard libraries	Diversity in services	IoT service APIs (e.g., IoBridge, Evrythng)
Resource Management	Pooling resources	Distributed system (e.g., Hadoop distributed file system)	Spatially separated resources, heterogeneous devices	Network controller for SDN (e.g., ONOS [5])
	Slicing resources	Virtual memory	Lightweight slicing	Hypervisor for compute resource
Application Management	Matching services with resources	CPU scheduling	Adaptation to dynamics environments (e.g., diverse service lifetime, devices mobility)	Virtual network embedding and adaptation
Edge Resource control	Registration & identification	Device manager	Diversity and large scale	AllJoyn and IoTvity's administration system [3], [4]
	Control interface	System bus	Heterogeneous network interfaces	Openflow, CoAP, MQTT

parties. Thus, FogOS needs to play the role of a broker of pooling/slicing edge devices' resources and coordinating all the players with self interest. Second, ONOS participates in standardizing device interfacing, e.g., OpenFlow [6]. However, in IoT, diversity in devices, services and protocols is inevitable, resulting in a more challenging environment. Cloud computing OS, e.g., OpenStack [7], also aims to orchestrate the large-scale computing resources in a shared infrastructure built on top of standard and commodity hardware under the same administrative domain. Hence, FogOS has the similar major differences, compared to traditional cloud computing OS.

There are three groups actively designing architectures for fog computing: Open Fog Consortium [8], ETSI MEC [9] and Cloudlets [10], each with slightly different visions and emphasis (see [11] for comparison). We believe that FogOS can be applied to or even merged with any of these architectures, as our focus is on handling the diversity and heterogeneity of user-oriented IoT services and edge devices that are owned by individuals and different owners using fog computing.

II. FOG OPERATING SYSTEM: ARCHITECTURE

A. Key Challenges

We describe the challenges of fog computing architecture for highly diverse IoT applications with heterogeneous edge devices owned by different individuals and providers (see Table I for the summary and the existing solutions).

- ***Scalability.*** Being at the exponential growth, there would be a significant number of IoT devices, which in turn run various IoT applications and generate a sheer amount of data.
- ***Complex inter-networking.*** Due to the large scale and diversity, IoT devices will be physically connected in various forms and under diverse conditions, e.g., wireless multi-hop connectivity using heterogeneous radio access technologies, often with mobility.
- ***Dynamics and adaptation.*** With wireless connectivity and mobility, IoT devices would experience frequent environmental changes in topology and communication conditions. In addition, IoT applications may have diverse lifetimes and QoS requirements, requiring prompt allocation of edge resources and re-embedding of IoT applications.
- ***Diversity and heterogeneity.*** Edge devices have various capabilities in communication radios, sensors, computing powers, storage, etc. This requires seamless interfacing and inter-operability, often incurring non-negligible overhead and yielding implementation/operation complexity.

In FogOS, we tackle the above challenges, using a reference architecture as depicted in Fig. 2, consisting of the following four main components: (i) service and device abstraction, (ii) resource management, (iii) application management, and (iv) edge resource: registration, ID/addressing, and control interface. The challenges due to diversity and heterogeneity is resolved by an abstraction layer for services and devices (see Section II-B). The application and resource managers closely work together to provide complex inter-networking services and adaptively allocate edge/fog resources to accommodate the dynamics of applications and resources (see Sections II-C and II-D). In Section II-E, we describe the ways of improving network and service scalability.

B. Service and Device Abstraction

In the fog computing environment where FogOS operates, there exists a common property in IoT applications and edge devices: *diversity.* This diversity complicates the process of developing an IoT application and the control of edge devices. It is therefore necessary to provide flexible, yet consistent abstraction as APIs, both from FogOS to applications (service abstraction or service API) and from FogOS to edge devices (device API). These APIs are designed and categorized by the degree of (i) *generalization* and (ii) *exposure,* where generalization refers to how concrete abstraction should be and exposure deals with how controllable we should make service and device through the designed APIs.

Service abstraction

(a) Generalization: In operating systems, users can directly access OS resources by invoking low-level system calls or using a high-level programming language dependent standard library. FogOS defines the following three hierarchical service APIs from low to high level:

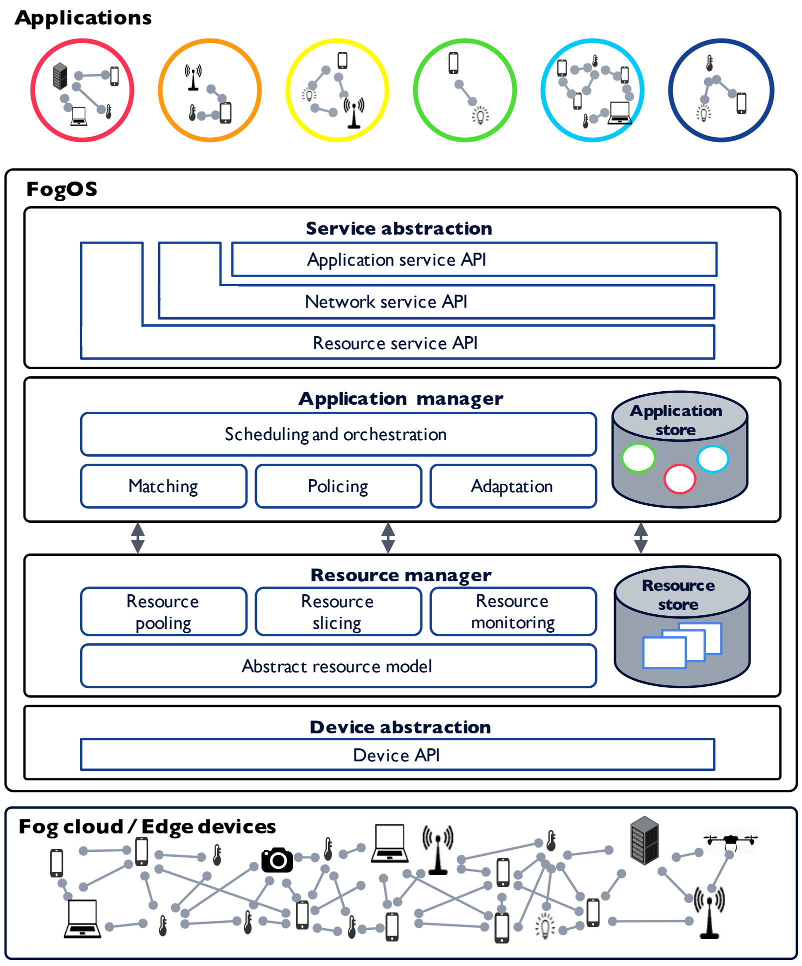

Fig. 2. FogOS (Fog Operating System) reference model.

- ◦ ***Level 1: Resource service API.*** This API resides at the lowest-level, providing services that can control each individual edge device resource such as compute, storage, sensing, actuating, radio access (or link) to applications. For example, a service call of "read the temperature from sensor X" can be invoked by an application.
- ◦ ***Level 2: Network service API:*** Using a collection of resource service APIs, this API provides the service of creating a networked slice that consists of some set of edge device resources. For example, to create a video surveillance service, a service call of "form a wireless video sensor network with video sensors X, Y, Z and an edge cloud C, where all wireless sensor nodes are connected to C with statistical bandwidth guarantee of 1 Mbps." Note that there can be a multi-hop path from X to C, where the resource matching module of the service manager determines the "optimal" path (see Section II-D).
- ◦ ***Level 3: Application service API:*** This API is at the highest abstraction that allows application developers to easily create a service that is defined as a typical service apriori. An example could be a call of "create a video surveillance service at hot spots with high-definition TV quality."

Note that there could exist more levels in this hierarchy. Application developers are allowed to utilize any level of service APIs with different controllability and programming proficiency.

(b) Exposure: An FogOS designer may choose different exposure degrees even in each service APIs, based on programming-friendliness and security level. For example, a video surveillance service can be created with the service requirement description {3 cameras} or {1 camera near *gps-1*, 2 cameras near *gps-2*} at the level-3 resource API. Also, everything cannot be open to application developers. For example, network link resources in the resource API at level 1 cannot be accessible because arbitrary change of link resources may negatively affect other applications, or even entire resource API at level 1 can be blocked to allow only high-level access.

Device abstraction

(a) Generalization: Unix-like operating systems treat everything as a file, e.g., /dev/sda1 for a hard disk, /tmp/mongodb-27017.sock for a socket, or /proc filesystem (procfs) for a process or other system information. FogOS could enjoy a similar level of generalization, but has more diversity to interface with various existing and emerging edge devices, which are manufactured possibly by different vendors. To this

end, multiple device data models are defined and exposed to FogOS, e.g., "sensor device data model," which is different from a single device data model in operating systems, i.e., a file. Note that device data models might have inheritance relation as in the object oriented programming language, e.g., temperature sensor device data model inherits sensor device data model, or some device data models might be grouped. This generalized abstraction of edge devices enables emerging IoT devices to be easily incorporated without affecting existing applications.

(b) Exposure: Typical operating systems provide different control granularity to each managed resource. For example, a default Wi-Fi device driver provides the interface to configure Wi-Fi behaviors. However, a vendor-provided device driver can be activated to expose vendor-specific features with finer control granularity. Similarly, edge devices that are in the same device category might have their own specific features that can be differentiated from other edge devices. Hence, FogOS still requires vendor-specific/owner-specific device drivers to control devices' detail or new features, in addition to general and abstract IoT device data model-based control.

C. Resource Management

FogOS manages the resources of edge devices and fog clouds that are spatially separated and often needs to be controlled in a distributed manner. We assume that the list of available resources are registered at the resource management module (see Section II-E) for the process of edge resource registration. As in traditional OS, FogOS pools or slices the available resources, whenever needed, but there exist many challenges to be handled, as elaborated next.

Resource pooling. The concept of resource pooling is used in a variety of context across different domains. In this article, we define resource pooling as a mechanism to collect the resources of the same "class." A good example in general OS is the notion of virtual memory in the hierarchical memory system, which enables the main memory and hard disk to be pooled, transparently seen as a run-time storage by running processes.

In fog computing, similar resource pooling would be useful in furnishing IoT applications with larger service options and freedom. The unique challenges of resource pooling in fog computing is: (i) pooling occurs among edge devices that might be placed in spatially different locations, (ii) the resources to be pooled are highly heterogeneous, and (iii) limited resources of edge devices often requires large-scale pooling. These unique features act as technical challenges that should be tackled by FogOS. A candidate list of resource pooling is as follows:

- *Compute/storage pooling.* The processing power of an edge device is likely limited. For intense data processing that requires fast response, we can use multiple edge devices in a distributed manner. Similarly, limited storage can be compensated by a distributed collection of storage of other devices.
- *Sensor/actuator pooling.* Many IoT applications relying on data from sensors might increase the information accuracy by exploiting similar data from multiple sensors. Also, different kinds of sensors lead to more complete information on the monitoring status, e.g., pooling and fusion of the data from a combination of gyroscopes, magnetometers, and accelerometers. In actuator pooling, a good example is multiple drones flying in a group, performing environmental sensing in a collaborative manner.
- *Network link pooling.* IoT applications that generate large data or require low latency need high-speed access links. To that end, an edge device with multiple communication radios can pool them to create a thick communication pipe. Also, a device that are not directly connected to a fog cloud could use other edge nodes as relays.

Resource slicing. As opposed to resource pooling, resource slicing corresponds to a mechanism that enables sharing of physical resources by multiple IoT applications. For example in OS, storage can be sliced through the concept of virtual memory space so that multiple processes can regard the entire (virtual) memory as if it is exclusively allocated to each single process. Multicore CPU allocating each core independently to each process is another example.

Slicing of the resources of edge devices can provide differential granularity and help to use the resources efficiently. Sensor/actuator and processing resources can be sliced temporally, and storage resource can be sliced spatially. The network link resource also can be sliced temporally as well as spatially, e.g., subdivision of communication channel. Synchronization and scheduling among the distributed resources is a key challenge, as poor execution would result in serious resource waste.

D. Application Management

As the resource manger manages all the edge device resources, the application manager manages everything on the running IoT applications by (i) matching the service requests to the edge resources, (ii) monitoring the running application's resource usage status and enforcing SLAs (Service-Level-Agreements), (iii) orchestrating the registered and available edge resources among multiple (e.g., prioritization), concurrent applications, and (iv) adapting to the changes of edge resource and application status.

Application-Edge resource matching. Different IoT applications need different types and amount of edge resources, depending on their QoS requirements. One of key functions of the application manager is to compute a matching solution from such requirements to the edge resources, where the available resources are obtained by querying to the resource manager. In typical OS, such a matching is trivial as the device resources are directly controllable and in small scale. However in IoT, there are many and diverse devices, often placed in spatially different locations, requiring networked control. Diversity requires the matching module to match the required resource "optimally" from many candidates. Depending on how effective this matching is, the number of IoT applications that can be accepted and run is determined, which has large impact on the revenue of IoT service providers and other economic players. Theoretical understanding of this matching problem must be made, and practically implementable algorithms with low complexity is of significant importance, which in turn depends on the type of applications provided as IoT application service API (see Section II-B).

Policing, scheduling, and orchestration. Once edge resources are appropriately allocated to incoming IoT applications, the application manager keeps track of their resource usage and monitors whether SLAs are violated. SLA violation of an application might degrade QoS of other applications, for which a certain level of resource partitioning often becomes of some value. The key challenge comes from the large scale of edge devices, where monitoring and policing for each would incur significant overhead even for a small edge device.

Dynamic creation and termination of IoT applications fluctuates available edge resources over time, often leading to the resource competition among them. In addition, each application would be assigned a different priority based on, for instance, security and pricing. All these motivates FogOS to employ a smart scheduler of running applications, similar to the job scheduler of OS.

Adaptation to resource and service changes. After the applications are matched to a set of edge resources, this matching result might not hold valid due to the change of application status and the change/fault of edge resources. Whenever applications with higher priority arrives or existing applications terminate, application manager might need to re-match the resource among the running applications for better resource management. This adaptation is also necessary when the edge resource topology changes as edge devices move or experience fault (e.g., battery shortage). In this case, the application manager must support continuous reconfiguration of application and the edge resources in collaboration with the resource manger. However, we must consider the trade-off between operation cost efficiency of reconfiguration and performance.

E. Edge Resource: Registration, Identification, and Control Interface

Identification and addressing. In fog computing, high dynamics and diversity of edge networks force FogOS to interact with edge resources frequently to keep up-to-date snapshot of Resource Store of the resource manager (see Fig. 2), and pool/negotiate a proper set of resources to embed various IoT applications, where an efficient identification of edge resources via IDing and addressing is essential. We propose to use both syntactic and semantic IDs in FogOS. Syntactic IDs refer to the ones that directly identify the edge resource (e.g., a 5th sensor of room 2 of building 2 of KAIST), whereas semantic IDs support the context of what a service want to utilize (e.g., any temperature sensor sensing 10 Celsius of room 2 of building 2). Each ID might use a different binding with its network address, e.g., static binding for syntactic IDs and dynamic binding for semantic IDs.

One can take reference from IoTivity's identification specification [3], AllJoyn [4], geocasting [12] or NDN (Named Data Networking) for IoT [13]. For example, AllJoyn requires each IoT device to know the minimum information (i.e., name) of destinations, and each device can find the destination device with this information by using mDNS (multicast). However, while AllJoyn is suited for a small-scale IoT network, FogOS targets networks with a large number of edge devices with high dynamics and diversity, and hence requires scalable solutions.

Resource discovery, registration and management. FogOS must discover edge devices and their resources, manage the list and monitor their status. Two schemes are possible: (i) proactive and (ii) reactive. In a proactive scheme, when an edge device enters our FogOS-administered network, it notifies FogOS of its intention to join with its list of available resources. FogOS then updates its resource store database to keep track of this new edge resources. In order to keep the up-to-date information, the available resource status must be periodically reported to the resource manager of FogOS. In a reactive scheme, edge resources are queried on demand, whenever new edge resources are needed as new applications are about to be created. Proactive schemes provide faster response to resource look-up and matching for new applications, but at the cost of larger overhead stemming from keeping track of the resource-related information. On the other hand, reactive schemes provide fresher information but with slower response time. AllJoyn follows this reactive scheme, mainly because it is designed for home-scale one-hop IoT applications. We envision FogOS to operate at a larger scale, thus we believe that a certain degree of proactive scheme is necessary for a possible hybrid approach.

Heterogeneous control and network protocols. As discussed in Section II-B, FogOS uses device APIs to control each individual edge device and fog clouds. We argue that to control fog clouds, the current approach SDN, i.e., OpenFlow is a good option. However, to control diverse resource-constrained edge devices, the classical SDN approach might be too heavy and inflexible. Thus, a lightweight version of SDN could be a candidate solution for separating data and control planes. Much of the challenges are due to high heterogeneity in control, communication and networking protocols of edge devices. The control plane should leverage existing IoT control protocols such as CoAP and MQTT, and also emerging architectures such as information-centric network (ICN). Similarly, the data plane should support diverse wireless technologies, e.g., Wi-Fi, LTE, LPWAN (Low Power WAN) [14], ZigBee, which is necessary to deliver data as well as control information. There exist different proposals for this, where one is to employ a gateway that can understand such heterogeneity, but such an approach of only a single-hop at the last mile might restrict the service coverage, and thus limiting the scope of possible IoT applications that FogOS supports. To extend the reach of FogOS, seamless multi-hop communication over a large-scale wireless nodes would be highly valuable.

III. FogOS-driven IoT Ecosystem

In the fog computing market, there are four key economic players that compete and cooperate to increase their revenues. This is depicted in Fig. 3. We do not claim that the eco-system

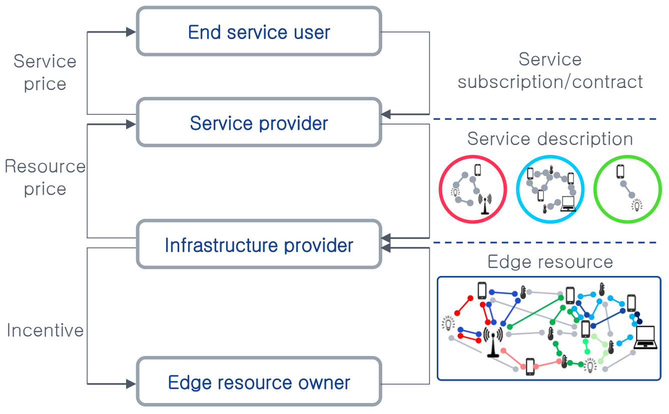

Fig. 3. Major economic players in fog computing and their interaction patterns.

mentioned in this article is the only one that would emerge. Rather, we believe that it might be one of the most basic and intuitive pattern where players interact.

(a) End service[2] users (SU). These are end users who are ready to enjoy IoT applications. They pay the application service fee to service providers under a variety of tariffs.

(b) Edge resource owners (ERO). These are individuals or large companies (such as mobile network operators that have large-scale communication and sensor infrastructures) who own edge resources or fog clouds. Especially, individual edge resource owners share their resource and partially or entirely sell the resource to an infrastructure provider. They act similarly to Uber drivers. They need to be appropriately incentivised to share the resource, where the incentive mechanism would be given by infrastructure providers.

(c) Service providers (SP). Service providers create diverse IoT applications that attract end service users as OTT (Over-The-Top) providers. Logically, they do not own the resources of fog clouds or edge devices, but rent them. Thus, they make a contract with infrastructure providers that manage the edge resources. Note that it is possible that service providers and infrastructure providers are run by a single company. Application development is made based on the service APIs opened by FogOS (see Section II-B).

(d) Infrastructure providers (InP). They are the ones who run FogOS. Infrastructure providers have infrastructure of edge devices and fog clouds, and might rely on individual edge resource owners for a large portion of IoT infrastructure. They are required to develop a nice incentive mechanism to attract as many edge resource owners as possible with low cost. Their resources are interfacing with FogOS via device APIs (see Section II-B), and sell their owned and leased resources to service providers. They make profit through the business with service providers and edge resource owners.

Note that this market is open to many diverse competition and cooperation scenarios. Edge device owners may act selfishly to maximize their individual revenue, or cooperate with infrastructure providers under fair revenue sharing mechanisms. As mentioned earlier, some big player might behave as multiple players. For example, an infrastructure provider, e.g., mobile network operator that already has a large-scale cellular and Wi-Fi infrastructure, minimally rely on edge device owners by deploying city-scale or even nation-scale sensor/actuator platforms; in such cases, big players are highly likely to provide IoT applications as well to make a large profit from the IoT industry. Non-cooperative and cooperate game theory helps in understanding these complex inter-play in this market and predicting the business landscape.

[2]In earlier sections, we used the term ‘IoT applications’ rather than ‘IoT service’ as ‘service’ is also used as the service provided by FogOS to applications. However, in this section, we use service to mean IoT application service, unless otherwise noted.

IV. Use Case and Demonstrator: City-Scale Surveillance Service

We now present the use case of FogOS, a large-scale surveillance service, where Fig. 4 shows our preliminary, proof-of-concept implementation of FogOS for a drone-based surveillance service.

Scenario overview. This is an example of Sensing-as-a-Service that deploys a city-scale surveillance, originally staring with a set of sensing of some target regions, and extends to the service coverage change with drone-driven moving sensors. In this case, a service provider can be an IoT sensor service provider, and SUs may include a public safety agency.

Original service

1) An SP requests a surveillance service to an InP (running FogOS) with a service requirement description $\{K$ regions, M videos, N audios, P sensors$\}$ through a Level 3 Application Service API.
2) FogOS searches available edge resources owned by itself as well as those owned by ERO (possibly with different priorities), and let the application manager allocate the required resources to embed this application (i.e., matching). These resources might be ready in advance through a proactive resource discovery mechanism, or be searched through a reactive mechanism (see Section II-E).
3) The application manager’s matching algorithm (i) produces an embedding solution, (ii) allocates the computed resource by communicating with the resource manager to retrieve existing/available fog/edge resource information, and (iii) commands the resource manager to perform the allocation action.
4) SUs enjoy this surveillance service.
5) The resource manager periodically monitors the resource usage at associated edge devices, and collaborates with the service manager whenever there exists any change or fault of the existing edge resources.

Service extension

1) The SP intends to observe more details of a specific region, say R due to an expected crime for instance. It requests to add a drone-based video sensing of region R with the modified service description $\{$region R, 1 video sensing with drone, `AVAILABLE` sensors$\}$ through a Level 1 & 2 resource and network service APIs. This service

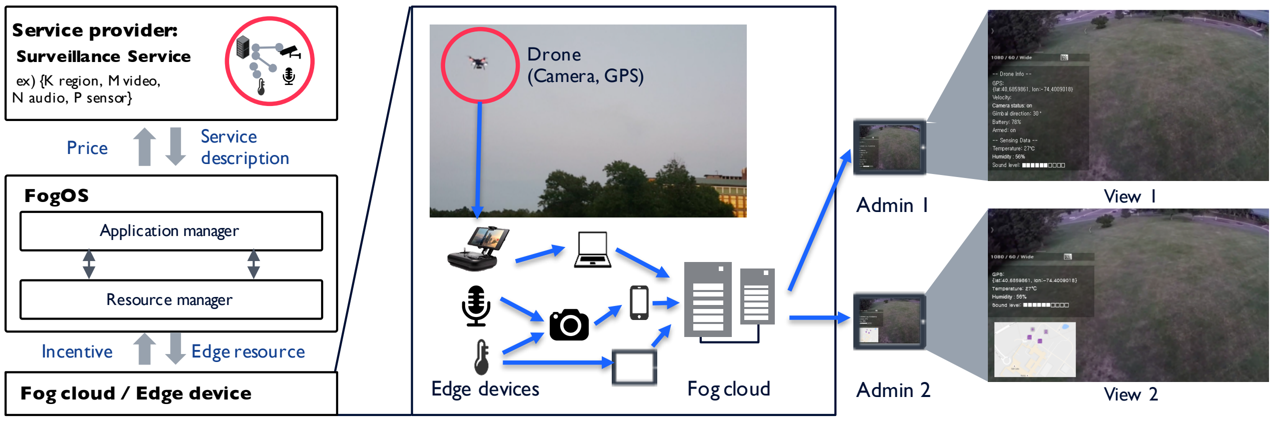

Fig. 4. Example implementation of surveillance service on FogOS: Extension of drone-based moving video sensing.

corresponds to live video streaming at edge cloud in region R to a single or multiple SUs.

2) FogOS searches its resource pool, and finds a fog cloud as well as a group of Wi-Fi APs by the InP, but failed to find a drone. It broadcasts a request of finding a drone with video sensor to the EROs in R.
3) The video scenes captured by drones and sensing data from the original service reach the allocated fog cloud in region R, which performs AR (Augmented Reality) functions to generate a richer content of the scene view. This post-processed video stream is delivered to multiple SUs.

Proof-of-concept implementation of FogOS

To see the feasibility of FogOS, we implement a prototype, where FogOS plays the following two roles: (i) controller and (ii) platform for IoT ecosystem. In our implementation, the economic interaction between key players is simplified, i.e., when EROs register their resources to FogOS, EROs' resources are shared through InP. In this article, we mainly focus on the control function of FogOS, as follows: (a) Drones and sensors are controlled by an application running on the FogOS through service and device abstraction layers. Thus, we are able to control flying drones and SDN IoT sensors through FogOS, (b) Computing, sensing and networking resources are pooled together and matched to this service by the resource and service managers of FogOS, and (c) a video from drones and sensing data are processed/merged by the allocated computing resources, and then multiple views for different SUs are created as shown in Fig. 4.

V. Conclusion

We introduced a fog computing and networking architecture for IoT services, termed FogOS, practicing architectural lessons from operating systems. FogOS is composed of four major components: service/resource abstraction, resource manager, application manager, and edge resource identification/registration, whose challenges and the main research directions are discussed. We hope that our vision in FogOS will be shared by other groups in academia and industry working on IoT and fog computing, and more constructive discussions will continue to follow, inspired by FogOS. These future directions include the extension of FogOS to support the key scenarios in 5G, i.e., eMBB (enhanced Mobile BroadBand), URLLC (Ultra-Reliable and Low Latency Communications), and mMTC (massive Machine Type Communications).

Acknowledgement

This work was partially supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0717-17-0034,Versatile Network System Architecture for Multi-dimensional Diversity).

References

[1] Gartner. “Gartner says 6.4 billion connected things will be in use in 2016, up 30 percent from 2015”, 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317 (accessed on 5/4/2017)
[2] V. Turner, C. MacGillivray, J. Gaw, R. Y. Clarke, M. Morales, and B. Kraus, “IDC FutureScape: Worldwide Internet of Things 2015 Predictions,” *International Data Corporation*, 2014.
[3] Open Connectivity Foundation (OCF), “OIC core candidate specification,” 2016.
[4] A. Alliance. “AllJoyn Framework”, 2016. [Online]. Available: https://allseenalliance.org/framework (accessed on 5/4/2017)
[5] ON.LAB. “ONOS - a new carrier-grade SDN network operating system designed for high availability, performance, scale-out”. [Online]. Available: http://onosproject.org (accessed on 5/4/2017)
[6] Open Networking Foundation, “Openflow switch specification - version 1.5.1,” 2015.
[7] “OpenStack: Open source software for creating private and public clouds”. [Online]. Available: https://www.openstack.org/ (accessed on 5/4/2017)
[8] “Open Fog Consortium”. [Online]. Available: https://www.openfog consortium.org/ (accessed on 5/4/2017)
[9] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, “Mobile-edge computing introductory technical white paper,” *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.
[10] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, “Cloudlets: at the leading edge of mobile-cloud convergence,” in *Proceedings of IEEE Mobile Computing, Applications and Services (MobiCASE)*, 2014.
[11] G. I. Klas, “Fog computing and mobile edge cloud gain momentum Open Fog Consortium, ETSI MEC and Cloudlets,” 2015.
[12] Y.-B. Ko and N. H. Vaidya, “Geotora: A protocol for geocasting in mobile ad hoc networks,” in *Proceedings of IEEE International Conference on Network Protocols*, 2000.
[13] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, “Information centric networking in the iot: Experiments with ndn in the wild,” in *Proceedings of ACM Information-Centric Networking*, 2014.
[14] “LTE-M - optimizing LTE for the internet of things,” White Paper, Nokia Networks, August 2015.

Nakjung Choi is Member of Technical Staff in Nokia Bell Labs, at Murray Hill, NJ, since April 2010. He received his B.S. (magna cum laude) and Ph.D. at School of Computer Science and Engineering, Seoul National University in 2002 and 2009, respectively. Also, he has received several awards such as Best Paper Awards and Awards of Excellence. His research focused on SDN/NFV/Cloud, 4G/5G/IoT and future converged services.

Daewoo Kim received his B.S. in the Department of Electrical Engineering from Yonsei University, South Korea in 2013. He is a doctoral student at the school of Electrical Engineering, KAIST since 2013. His research interests include sensor networks, network economics, fog computing, and machine learning in networking.

Sung-Ju Lee is Associate Professor and KAIST Endowed Chair Professor at KAIST. He received his PhD in Computer Science from UCLA in 2000, and spent 15 years in the industry in the Silicon Valley before joining KAIST. His research interests include computer networks, mobile computing, network security, and HCI. He is the winner of the HP CEO Innovation Award, the Best Paper Award at IEEE ICDCS 2016, and the Test-of-Time Paper Award at ACM WINTECH 2016.

Yung Yi is Associate Professor at the Department of Electrical Engineering at KAIST. He received his Ph.D. in the Department of Electrical and Computer Engineering at the University of Texas at Austin in 2006. His research interests include computer networks and machine learning. He received the two best awards at IEEE SECON and ACM Mobihoc in 2013, and he was the winner of IEEE William R. Bennet Prize in 2016.