Towards Distributed SDN: Mobility Management and Flow Scheduling in Software Defined Urban IoT

Di Wu, *Member, IEEE*, Xiang Nie, Eskindir Asmare, *Member, IEEE*, Dmitri I. Arkhipov, Zhijing Qin, Renfa Li, *Senior Member, IEEE*, Julie A. McCann, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—The growth of Internet of Things (IoT) devices with multiple radio interfaces has resulted in a number of urban-scale deployments of IoT multinetworks, where heterogeneous wireless communication solutions coexist (e.g., WiFi, Bluetooth, Cellular). Managing the multinetworks for seamless IoT access and handover, especially in mobile environments, is a key challenge. Software-defined networking (SDN) is emerging as a promising paradigm for quick and easy configuration of network devices, but its application in urban-scale multinetworks requiring heterogeneous and frequent IoT access is not well studied. In this paper we present UbiFlow, the first software-defined IoT system for combined ubiquitous flow control and mobility management in urban heterogeneous networks. UbiFlow adopts multiple controllers to divide urban-scale SDN into different geographic partitions (assigning one controller per partition) and achieve distributed control of IoT flows. A distributed hashing based overlay structure is proposed to maintain network scalability and consistency. Based on this UbiFlow overlay structure, the relevant issues pertaining to mobility management such as scalable control, fault tolerance, and load balancing have been carefully examined and studied. The UbiFlow controller differentiates flow scheduling based on per-device requirements and whole-partition capabilities. Therefore, it can present a network status view and optimized selection of access points in multinetworks to satisfy IoT flow requests, while guaranteeing network performance for each partition. Simulation and realistic testbed experiments confirm that UbiFlow can successfully achieve scalable mobility management and robust flow scheduling in IoT multinetworks; e.g., 67.21 percent throughput improvement, 72.99 percent reduced delay, and 69.59 percent jitter improvements, compared with alternative SDN systems.

Index Terms—Distributed control, flow scheduling, Internet of Things, mobility management, software defined networking

1 Introduction

Recent developments in wireless communications and embedded systems have resulted in consumer devices becoming highly ubiquitous creating a strong interest in the Internet of Things (IoT) [1], [2] as part of smart city solutions. Real world urban IoT applications are expected to be heterogeneous, due to various access networks and connectivity capabilities [3], [4], resulting in geographically wide-scale *multinetworks* [5] where there is a coexistence of multiple wireless communication solutions (e.g., WiFi, Bluetooth, Cellular). Given the heterogeneity of IoT multinetworks, it is challenging to coordinate and optimize the use of the heterogeneous resources in mobile and multi-access edge computing environments [6], [7].

1.1 Motivations

Large-scale mobile IoT network in urban scenario generates highly dynamic traffic flows [8], [9]. Any solution which manages flows in the networks adaptively, stores and updates non-trivial amounts of information regarding flow's devices, access points, and allows dynamic modifications in the strategies according to which the management is performed will require significant processing power not available on commodity networking devices [10]. Software Defined Networking (SDN) [11] presents a feasible solution which provides all of these features.

SDN is a relatively new paradigm for communication networks which separates the control plane (that makes decisions about how traffic is managed) from the data plane (actual mechanisms for forwarding traffic to the desired destinations); where control is handled by the SDN *controller*. This decoupling abstracts low-level network functionalities into higher level services, therefore allowing quick and flexible configuration for flow-based routing and enabling rescheduling over the network components. SDN is

- D. Wu is with the Department of Computer Engineering, Hunan University, Changsha 410082, China, the Department of Computer Science, University of California, Irvine, CA 92617, and also with the Department of Computing, Imperial College LondonSW7 2AZ, London, United Kingdom. E-mail: dwu@hnu.edu.cn.
- X. Nie and R. Li are with the Department of Computer Engineering, Hunan University, Changsha 410082, China. E-mail: {neo, lirenfa}@hnu.edu.cn.
- E. Asmare and J.A. McCann are with the Department of Computing, Imperial College LondonSW7 2AZ, London, United Kingdom. E-mail: {e.asmare, j.mccann}@imperial.ac.uk.
- D.I. Arkhipov and Z. Qin are with the Department of Computer Science, University of California, Irvine, CA 92617. E-mail: {darkhipo, zhijingq}@ics.uci.edu.
- K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561, and also with the Department of Computer Science, Hunan University, Changsha 410082, China. E-mail: lik@newpaltz.edu.

Manuscript received 21 Sept. 2018; revised 10 Sept. 2019; accepted 5 Oct. 2019. Date of publication 28 Nov. 2018; date of current version 31 Jan. 2020. (Corresponding author: Di Wu.)
Digital Object Identifier no. 10.1109/TPDS.2018.2883438

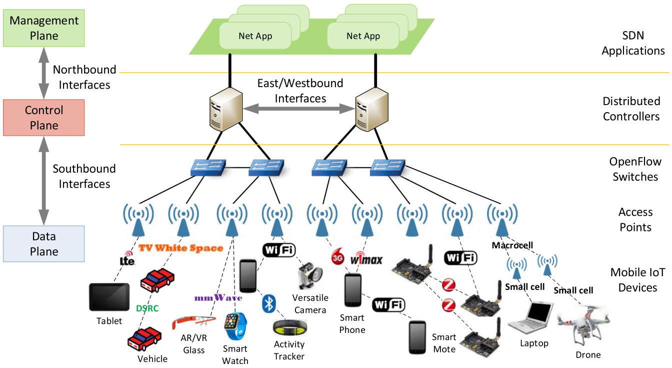


Fig. 1. Software defined IoT.

particularly useful when networks have to be adapted to ever changing traffic volumes with different demands. It is for this reason that we believe that SDN is a good approach to solving the resource management and access control issues in wide-scale IoT multinetworks.

OpenFlow [11] is the most prominent approach which implements the SDN concept, where the controller takes charge of all the functions in control plane, while the OpenFlow switch retains only the basic data forwarding functions. In the OpenFlow centralized control model, all routes are determined by the controller taking a global *view* of the network status. However, the request processing capability of a single controller is limited; for example NOX [12] can process about 30K requests per second. In fact, large-scale network environments (e.g., IoT applications in smart cities) have the potential to provide vast amounts of data flows; according to the report from Cisco, by 2021, there will be over 11.6 billion mobile-connected IoT devices and the monthly global mobile data traffic will surpass 10 exabytes [13]. With the increasing scale of IoT deployments, centralized controllers will have serious implications for scalability and reliability. Hence the next logical step is to build a distributed control plane with multiple physical controllers, which can provide the scalability and reliability of a distributed architecture and yet preserves the simplicity of the control function.

Fig. 1 presents such a software defined IoT system with the support of distributed controllers and partially connected OpenFlow switches in multinetworks that have heterogeneous *access points*. In this architecture built on the distributed SDN framework, different IoT devices are associated with the heterogeneous access points, dependent on their various needs of flow types [14], [15]. The assignment of access points to IoT devices is determined by the coordination of distributed controllers and OpenFlow switches. Specifically, as shown in the figure, the IoT scenario in our paper is illustrated as an urban-scale IoT multinetwork, where IoT traffic flows are mainly from heterogeneous IoT devices composed of various consumer devices (e.g., Smart Phone, Smart Watch, Smart Mote, Tablet, Laptop, Drone, AR/VR Glass, Activity Tracker, Versatile Camera, Robot, Vehicle) that are configured with multiple radio access capabilities (e.g., WiFi, 3G, 4G, Bluetooth, Zigbee, WiMax, DSRC, TV White Space, mmWave) [3], [16], [17].

In urban environment, these heterogeneous IoT devices carried by human or vehicle keep roaming from one urban partition to another, and the IoT traffic flows requested by these devices in mobile scenario can be support by their multiple radio access capabilities [18], [19], [20]. Therefore we need a city-wide mobility management to dynamically assign proper access point to these mobile IoT devices. Also, most mobile traffic requested by these IoT devices are driven by surrounding events (spatial domain) and personal interests (temporal domain), therefore the urban-scale flow scheduling across heterogeneous access points requires analysis of spatialtemporal status of the IoT multinetworks. Different from existing works that mainly discuss the cognitive radio capability within the IoT devices for being co-existent [21], in software defined IoT system, the cognitive radio capability should be mainly executed by the SDN controller and the controller can enable proper radio interface in the IoT devices to access corresponding spectrum.

However, the current implementations of SDN technologies are still far from addressing the heterogeneous and dynamic needs of ubiquitous IoT applications, especially in mobile environments [22], [23], [24]. The popular use of SDN technologies today is in Data Center Networks (DCNs) [25], [26], [27], where the focus is on the optimisation of network behaviours (e.g., bandwidth consumption) where nodes are linked via fast interconnections within a data center. Even though Huawei have recently launched the world's first SDN-based Agile IoT Solution [28] as we envisioned, their applications are only restricted to use centralized controller to manage static IoT devices (mostly sensors) deployed in buildings, home appliances, gymnasiums, etc., which is far from large-scale IoT multinetworks and has not addressed the mobile IoT scenario that we have illustrated in Fig. 1. In contrast to these SDN applications, state information in the urban-scale IoT multinetwork setting is gathered from mobile IoT devices distributed over a more loosely coupled ubiquitous network. Therefore, the main issues related to the application of software defined IoT in urban mobile environments are:

1) The operation of a distributed control plane requires scalable control combined with consistent management to coordinate multiple controllers and switches for message exchange, while providing data replication and maintaining flow scheduling. This is especially challenging given IoT devices roam frequently in urban environments and each controller needs a network view about the mobility of these IoT devices to manage their spatio-temporal access requests and collaborate with other controllers for adaptive handover and dynamic flow scheduling over multinetworks. Where component failure or traffic congestion occurs, distributed controllers are required to be fault tolerant and able to load balance.
2) The last hop links in urban IoT architecture are more heterogeneous than existing Wi-Fi/LTE scenario. As shown in Fig. 1, the last-hop radio access technologies (RATs) in urban IoT multinetworks not only incorporate the well-known WiFi/3G/4G technologies but also potential 5G communication solutions, such as TV White Space, mmWave and different types of small cells, due to the emergent rich consumer IoT devices that are configured with various

interface capabilities and designed for different application services. In addition, there might be several access points providing homogeneous RAT for client to connect at the same time. Coordination and communication in this urban IoT multinetworks is challenging because IoT device in the highly heterogeneous networks faces additional concerns on link heterogeneity and redundancy; it must solve this added complication and choose not only the proper interface used in the connection but also the best-available access point through which a device establishes a connection, so that the basic flow QoS can be guaranteed for its running application.

3) Unlike the DCN situation, link and node capabilities in IoT multinetworks are highly heterogeneous and application requirements are correspondingly different. This implies that single objective optimization techniques of typical DCN flow scheduling are not directly applicable in IoT multinetworks. In this sense, controllers should schedule the access point to transmit IoT flows based on specific per-device service requirements, while providing network traffic balance through the interactions between controllers and controlled devices.

4) The performance metrics of interest in urban IoT multinetworks go beyond bandwidth consumption; with more heterogeneous and time-sensitive traffic flows from consumer IoT devices; unlike DCNs, whose network requirements primarily focus on utilization and bandwidth, IoT multinetworks metrics are delay, jitter, packet loss, and throughput.

1.2 Summary of Prior Work

Two popular approaches have been used in scalable SDN management. One is to design a distributed SDN architecture, such as Hyperflow [29] and Onix [30]. In Hyperflow, the controllers are flatly organized where every controller has a global view of the network. Onix controllers represent a hierarchical structure, where the lower tier controller and its managed network are aggregated as a logical node in upper tiers. The network view is distributed among multiple controller instances in Onix. The alternative approach is to offload the partial workload of controllers to switches, as DevoFlow [26]. This approach can improve scalability to some extent, however the switch hardware is required to be modified. Nevertheless, all of the above scalable techniques are designed specifically for DCN, not designed for IoT multinetworks.

Our work contributes to the dynamic SDN management for IoT communications. In previous multi-contrioller systems, ElastiCon [31] was proposed as an elastic distributed controller architecture designed to dynamically reassign switches to controllers and grow or shrink the pool of controllers assigned to an SDN as demand grows or shrinks. Schmid *et al.* [32] present a locality centered view of distributed SDN computing to partition distributed system by symmetry into problem aspects, where the symmetric aspects must be solved globally, and asymmetric aspects can be solved locally. However, none of existing work has presented fine-grained solution to guarantee both local and global performance by adapting to the variance of spatial-temporal demands. As for consistent maintenance of distributed system, Cassandra [33] is a distributed storage system for managing very large amounts of structured data spread out across different data centers, while providing highly available service with no single point of failure. Auspice [34] produces a scalable alternative to DNS which is able to resolve resource identity more quickly than competing alternatives under conditions where devices are highly mobile. These literature worked on a related but not identical problem in comparison with our research. We concentrate on consistent scheme of distributed controllers to exchange mobility information of IoT devices roaming across different urban partitions deployed with heterogeneous wireless communication infrastructure.

More recently, SDN techniques are being applied to heterogeneous wireless networks, differently from traditional flow and access scheduling schemes in specific networks [35], [36], [37]. OpenRadio [38] suggests the idea of decoupling the control plane from the data plane to support ease of migration for users from one type of network to another, in the PHY and MAC layers. The flow scheduling between WiFi and WiMAX/Bluetooth networks when video data is streamed has been prototyped in OpenRoads [39] and MINA [40], using centralized controller. OpenFlow based vertical handover is also discussed and implemented in the GENI testbed [41]. These wireless SDN solutions provide the necessary building blocks for managing IoT multinetworks, but they are not sufficient. Two important functions absent in these wireless SDN solutions are mobility management and distributed control. Mobile IP [42] uses a tunnel between a mobile device and a home agent to record the new IP address of the mobile device, but its triangle routing problem adds delay and extra network costs. SoftCell [43] and SoftMoW [44] have mechanisms in handling mobility and handovers, however these architectures are designed specifically for cellular networks and do not address the device/flow heterogeneity problems. Therefore the existing schemes cannot support heterogeneous IoT devices and dynamic flow requirements in urban mobile scenario.

1.3 Our Approaches and Contributions

In this paper, we present UbiFlow, the first software-defined IoT system for ubiquitous flow control and mobility management in urban heterogeneous networks. To achieve lightweight processing in IoT devices, in UbiFlow all jobs related to mobility management, handover optimization, access point selection, and flow scheduling are executed by the coordination of distributed controllers. Specifically, UbiFlow adopts multiple controllers to divide an urban-scale SDN into different geographic *partitions* to achieve distributed control of IoT flows. A distributed hashing based overlay structure is proposed to maintain network scalability and consistency. Based on this UbiFlow overlay structure, relevant issues in mobility management such as scalable control, fault tolerance, and load balancing have been carefully examined and studied. The UbiFlow controller differentiates flow scheduling based on the requirements per-device as well as whole-partition capabilities. Therefore, it can present a network status view for the optimized selection of access points in multinetworks to satisfy IoT flow requests, while guaranteeing the network performance in each partition. In general, the key contributions of UbiFlow are as follows:

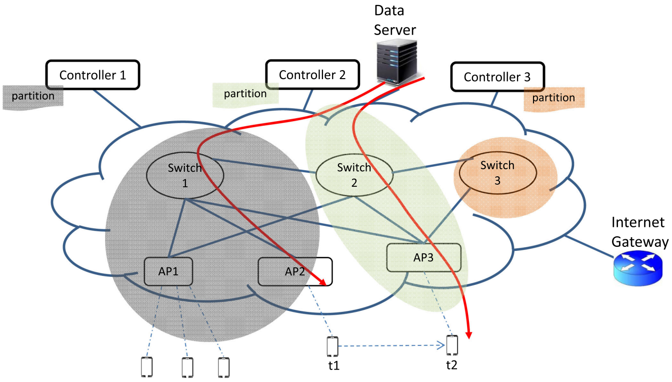


Fig. 2. UbiFlow system architecture.

1) A novel overlay structure to achieve mobility management and fault tolerance in software-defined IoT. The consistency and scalability of distributed controllers are maintained by their individual roles under a framework using modified distributed hashing in ubiquitous environments.
2) A network calculus model based on discrete packet aggregation is used in the analysis of network requirements. Both node-level analysis and multi-hop path analysis by association operations present a partition view for the controller to evaluate current network status, and then make handover decisions for IoT flow scheduling.
3) Distributed controllers are able to match the best available access points to IoT devices, by running an assignment optimization algorithm with current network status analysis and incoming IoT flow requests as inputs. An adaptive window scheme is designed for the algorithm to serve mobile IoT devices with better output.
4) Instead of static mapping between switches and controllers, UbiFlow periodically load balances the controllers by analyzing the variations in both temporal and spatial traffic characteristics. The dynamical adaptation of the switch to controller mapping is accomplished via elastic double-hashing on the fly.

Note that UbiFlow is specifically addressed to regulate city-wide mobility management and flow scheduling of IoT consumer devices in urban environments, however it also can be applied in many different kinds of IoT networks for example inventory management across multiple warehouses and retail location, and asset monitoring for heterogeneous construction equipment, where flow scheduling is always needed between mobile IoT devices and surrounding access points [45].

The rest of this paper is organized as follows. Section 2 gives a system overview of UbiFlow architecture. Section 3 describes the UbiFlow overlay structure for mobility management. Section 4 addresses the ubiquitous flow control and related issues in UbiFlow. Section 5 evaluates UbiFlow performance using simulations and real testbed experiments. Section 6 concludes our paper.

2 System Overview

The UbiFlow system is designed for ubiquitous access to multinetworks and the mobility management of IoT devices in the distributed SDN context. The system architecture is illustrated in Fig. 2, where the data server, controllers, switches, access points and IoT devices act as its core components.

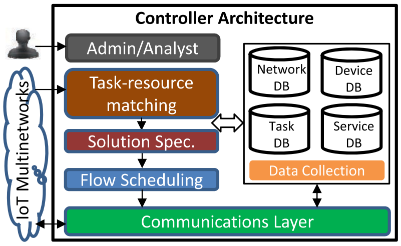
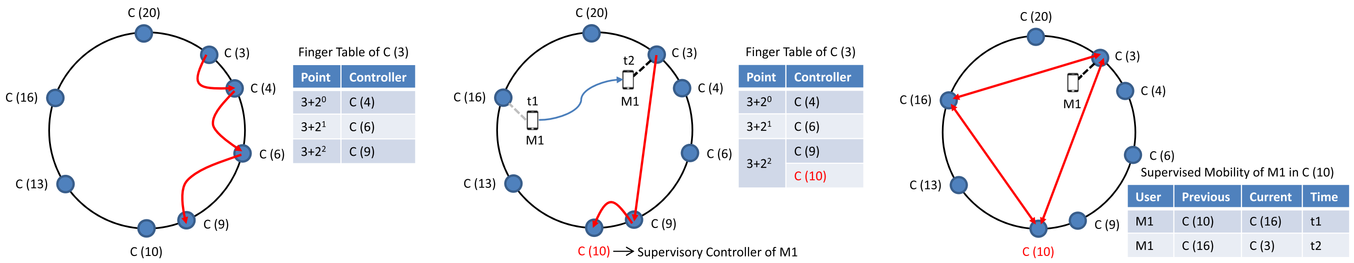


Fig. 3. UbiFlow controller architecture.

Multiple controllers have been deployed to divide the network into several partitions, which represent different geographical areas in our paper. All IoT devices in a single partition associate with different types of access points (e.g., WiFi, WiMAX, Cellular), which are connected to local switches to request various types of data flow (e.g., text, audio, video) from the corresponding data server. Information pertaining to service requests and flow transmissions can be analyzed and administrated by the partition-dependent controller. Additionally, for urban-scale SDN, mobile IoT devices roam across different partitions at different times. Newly joining and leaving IoT devices are also recorded in the local controller to indicate user density and resource usage. Therefore, each controller has a partitioned view of its local network status.

The architecture of UbiFlow controller is illustrated in Fig. 3. The data collection component collects network/device information from the IoT multinetwork environment and stores it in databases. This information is then utilized by the layered components in the controller. The task-resource matching component maps the task request (e.g., flow requirements) onto the existing resources (e.g., available access points) in the multinetwork. Once candidate resources are selected, the solution specification component adds more network characteristics and constraints (e.g., partition view) to filter resources. Finally, the flow scheduling component takes these requirements and schedules flows that satisfy them. For example, given a minimum throughput request from an IoT device, the controller first lists some candidate access points in its partition that can provide the throughput, and then adds more constraints (e.g., delay, energy, fairness) to find the best available access point that can both satisfy the flow request of the IoT device and guarantee optimal network performance of whole partition. To better perform network monitoring, the controller is also extended by the Admin/Analyst APIs, which enable the control processes to be governed not only by the controller itself but also by humans or external programs.

In UbiFlow architecture, as shown in Fig. 2, switches from different partitions are partially interconnected, so that the network information recorded in different controllers can be exchanged through these connected switches to achieve network consistency and robust maintenance. In addition, connected switches can also facilitate the inter-controller flow migration over the IoT multinetwork for load balancing purpose. In general, there are two types of IoT flows in the context of a distributed SDN as shown in Fig. 2. The first



Finger Table of C (3)

Point	Controller
$3+2^0$	C (4)
$3+2^1$	C (6)
$3+2^2$	C (9)

(a) Overlay

Finger Table of C (3)

Point	Controller
$3+2^0$	C (4)
$3+2^1$	C (6)
$3+2^2$	C (9)
	C (10)

(b) Mobility

Supervised Mobility of M1 in C (10)

User	Previous	Current	Time
M1	C (10)	C (16)	t1
M1	C (16)	C (3)	t2

(c) Handover

Fig. 4. Mobility management in UbiFlow overlay network.

one is the IoT flow between the data server and the IoT device. This is scheduled through intra-partition communication, with the assistance of a local access point, switch and controller. The second one is the IoT flow between IoT devices located within different partitions. This type of IoT flow needs to be scheduled through inter-partition communication. Utilizing the connected switches, controllers can coordinate to direct the flow initiated from one partition to a different access point in another partition. Note that IoT device to IoT device multi-hop wireless communication (e.g., ZigBee, WiFi Direct) also exists in the UbiFlow system. If this happens in the same partition and the last hop is directed to an access point to connect remote data server, then it can be classified as the first type of IoT flow. Otherwise, if the last hop is an IoT device (not an access point) in the same partition as the origin device, then the IoT flow is transmitted by purely multi-hop wireless communication without SDN support [46], [47]; it does not belong to the discussion of this paper, since we focus on using SDN to improve the IoT multinetwork performance.

Given the UbiFlow architecture, we will discuss its mobility management and flow scheduling in the following sections.

3 Mobility Management in UbiFlow

When IoT devices roam from one partition to another and request efficient handover, a consistent scheme to coordinate controllers is required for the mobility management of IoT devices. Assume a large-scale IoT networks is composed of different regions. These regions can correspond to a prior geographic division, for example Zip Codes; regions are distinct and different from the UbiFlow partitions in our scheme; a region could have multiple partitions. For a region-scale network UbiFlow uses an overlay structure based mobility solution to present it, as shown in Fig. 4. We will illustrate its key functions in the following sections.

3.1 Overlay Structure

Two types of IDs are used in the mobility management, which are:

- *Mobile ID*: the identifier of a mobile IoT device (e.g., IP v6 address or MAC address);
- *Controller ID*: the identifier of a controller in distributed SDN.

To provide scalable and efficient mobility management, UbiFlow maintains a controller network based on structured overlays (e.g., Chord DHT [48]), where a *consistent hashing* [49] is maintained based on an ordered ring overlay, as shown in Fig. 4a. The purpose of using a ring structure similar to Chord in our UbiFlow system is to locate resources which can be mapped into the hash ring in a network rapidly and efficiently [50]. In the consistent hashing framework, distributed controllers are configured as overlay nodes with unique integer identifiers in the range of $[0, 2^{m-1}]$. Each controller ID can be represented by m bits. The consistent hashing also matches each mobile ID with an m-bit integer as a "key" using a base hash function h, such as SHA-1 [51]; therefore key = h (Mobile ID). The key can be later used for the lookup of controllers, as explained in Section 3.3.

Each controller $C(n)$ with ID n maintains a routing table, namely the "finger table", to achieve scalable key lookup in this overlay structure. Each finger table has up to e entries. The ith entry in the table indicates the closest controller to the corresponding point, where the controller ID $\geq (n + 2^{i-1})$. A query for a given key is forwarded to the nearest node that most immediately precedes the key, among the e entries at the controller. Finger tables are used for the case where there is no controller with the exact ID as the key value. In that case, we designate the closest successor of the key as the expected controller. For example, in Fig. 4a, we represent the controller with ID n as $C(n)$, and there are 3 entries in the finger table of $C(3)$. The 3rd entry of the finger table points the successor of the key $(3 + 2^2)$, which is $C(9)$ in reality.

Theorem 1. *In an N-controller overlay network based on consistent hashing, the lookup cost to find a successor is bounded by $O(\log N)$.*

Proof. The lookup cost in an N-controller overlay network indicates the number of nodes that must be contacted to find a successor. The above theorem has been proved in the paper [48] that introduced Chord overlay structure, also based on consistent hashing. □

For a region-scale network, UbiFlow divides a region into multiple partitions and each partition is regulated by a controller. The region maintains its own overly structure as shown in Fig. 4a to lookup controllers in different partitions. The propagation latency between controllers within a region is not big enough to impact the lookup delay, due to the fact that the distance between partitions is no very differed.

3.2 Mobility Structure

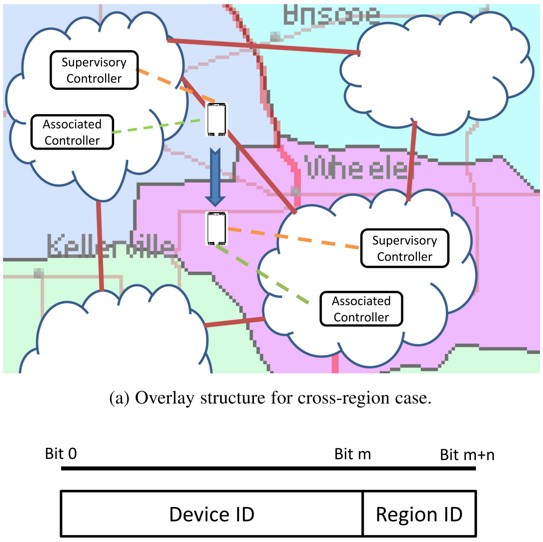
In our region-scale SDN based controller overlay architecture, we achieve efficient mobility management through coordination between controllers. Specifically, in an SDN with multiple controllers, we have two classes of controllers:

- *Associated Controller*: the current controller that the mobile IoT device is associated with;
- *Supervisory Controller*: the controller that is assigned to a newly joined IoT device as its initially associated controller. Note that each supervisory controller also functions as an associated controller but with additional information to record the mobility behaviour of its supervised IoT devices. The updated mobile information of an IoT device could be collected through information exchange with its current associated controller, following our UbiFlow architecture as described in Section 2

A geographical region, e.g., a city, is divided into several network partitions; each partition has a single controller. An IoT device that has never previously connected to the network after entering a region assigns itself a default access point (it is chosen arbitrarily from those available to the IoT device to connect to the network). The default access point is used to register the IoT device to its supervisor controller, which is determined by the hash value of the mobile ID of the IoT device, and then connect the IoT device to current partition's controller as its associated controller. After analyzing the device's flow requests and the communication capability of its default access point, the associated controller will assign an optimal access point in the partition for the device to transmit its requested flow (the optimization for making this choice is described in Section 4.2).

Each network request will be sent to the device's associated controller because it is at the gateway between the Internet and the device. The associated controller will log that it is the controller for the mobile device in its area at the device's supervisory controller. Then when the device moves to another partition, its associated controller assigned in the new partition can recover the flow from the device's previous associated controller (information stored at the supervisory controller) without disrupting the flow. After the device's new associated controller captures the flow state from the device's previous associate controller the previous associated controller is free to remove this data from its memory. Finally, once the state information is stored at the new associated controller, the IoT device is notified and it gracefully transitions from the access point used to communicate with the previous associated controller to the access point used to communicate with the new associated controller.

Note that in overlay networks covering geographically distant areas propagation delays become an issue and path latency can be increased because the algorithms operating on the overlay network do not take into account these implicit latencies between overlay nodes. However by virtue of a relatively simple change in our architecture and protocol in Fig. 4a, cases where the network covers a wide area can be accounted for. The UbiFlow architecture presented in this paper assumes that a large-scale network can be broken into multiple regions and each region maintains an independent overlay structure. These regions are geographically localized to the extent that cross-region propagation delay between their controllers in the network is within the same order of magnitude. To account for this new complication each IoT device will maintain a geofence [52] detecting its migration from region to region. Each device will store both the origin region and destination region when a region boundary is crossed. We amend the description of a mobile ID as introduced in Section 3.1 to include not only a unique device identifier, but also a unique regional identifier specifying the coordinate of the centroid of the region the device is in. The mobile ID is a concatenation of these two identifier fields.



(a) Overlay structure for cross-region case.

(b) Mobile ID for cross-region case.

Fig. 5. Mobility management in large-scale IoT network.

As an IoT device transitions from region to region and simultaneously from partition to partition (as we assume that no partition spans more than one region) it will migrate its supervisor controller from a controller mapped to with the hash of an identifier representing its origin region to a hash entry representing its destination region. In addition, it is necessary to ensure that the $\log(N)$ communications necessary to recover the supervisor controller do not obscure asymmetric latencies; to do so we employ the family of locality preserving hash functions discovered by Indyk *et al.* [53]. As our hash function for UbiFlow overlay, we define the distance metric in the domain of our function to be the spherical distance between the regional centroid components of the identifiers. With these modifications we can ensure first that the supervisor controller for any device will always be identified by an identifier encoding the centroid of the same region as the location in which the device is located. Second, we ensure that hash-lookups of the device's identifier (including the identifier of the residing region) will map into the neighborhood of controllers located geographically near it, thus the $\log(N)$ communications will take place within a region where latency between nodes is similar.

Fig. 5 shows five regions from the US state of Texas, the regional division coincides with the actual division of the land in the state into separate zip codes. Suppose that each such region is administered by a network implementing UbiFlow, that is internally each cloud in Fig. 5a which represents a UbiFlow deployment as described in Fig. 2. Then if an IoT device is moving across the zip-code boundary between the top-left and bottom-right regions as described above, the geofence at the mobile device will send a request for the supervisory controller and associated controller managing the device in the UbiFlow deployment at the top-left region to transfer mobility and flow information to its new supervisory and associated controllers at the bottom-right region. The bottom-right region will assign these controllers based on a new mobile ID that the mobile device will assign itself at the time it enters the bottom-right region. We show the changes to the mobile ID needed for this transitioning to work in Fig. 5b. When the device transitions to the bottom-right region and associates with the new mobile ID, its previous supervisory controller (located as before by hashing previous mobile ID) will transfer previous mobility and flow information to its new supervisory controller that can be hashed by the new mobile ID.

For consistency in following sections, we will mainly describe UbiFlow for the region-scale overlay network. However, by the above modifications, UbiFlow can be easily extended to support large-scale IoT networks that are composed of multiple regions.

3.3 Mobile Handover

In the urban mobile scenario, when an IoT device is assigned to one controller, the controller will store its flow information. However, the IoT device may frequently change its associated controller. To achieve seamless mobile handover, when an IoT device enters a new partition, the corresponding controller requires a fast lookup of the device's previous associated controller to fetch the uncompleted session data so that it can quickly reroute flows to the newly joined IoT devices via the APs in its partition. Therefore the mobile IoT device can obtain a continuous data service without data loss or handover delay. In the SDN based mobility management, we achieve efficient handover through the coordination between controllers.

When a new IoT device joins the distributed SDN network, as a bootstrapping step, it will be assigned to a supervisory controller as its initially associated controller, based on the hash result of its mobile ID. In the mobility scenario, for each IoT device with its mobile ID as the original value, the UbiFlow overlay structure can hash the mobile ID to get an integer key, and use this to localize its supervisory controller. Both the controller ID and the hashed key of the mobile user are required to be placed in the same ID space ranging $[0, 2^{m-1}]$. Specifically, to localize the supervisory controller, we follow the rule to assign a hashed key to the controller that has the closest ID, namely the immediate successor of the key.

Since every controller can use consistent hashing to localize an IoT device's supervisory controller, the supervisory controller is used in UbiFlow to record the previous and current associated controllers of the mobile IoT device. Using this scheme for distributed SDN, the new associated controller can localize the previous associated controller of the IoT device by fetching this information from the supervisory controller assigned to the IoT device.

As shown in Fig. 4b, mobile IoT device 1, denoted as $M1$, was previously associated with controller $C(16)$ at time $t1$, and its supervisory controller is $C(10)$. When $M1$ moves to the geographical partition of $C(3)$ at time $t2$, $C(3)$ needs to localize its previous associated controller and reroute flows to its current partition. To do this, $C(3)$ first tries to localize the supervisory controller according to the hashed key of $M1$. Based on the finger table, $C(3)$ can forward the lookup request to the furthest controller $C(9)$ that is closer to the supervisory controller. Then $C(9)$ can help to localize $C(10)$ as the expected supervisory controller. As shown in Fig. 4c, once $C(3)$ localizes $C(10)$ from the traceback route, as $C(10) \rightarrow C(9) \rightarrow C(3)$, it can later directly communicate with $C(10)$ to learn that the previous associated controller of $M1$ is $C(16)$ at $t1$. After this, $C(3)$ can directly communicate with $C(16)$ to fetch the previous session between $M1$ and $C(16)$ and reroute flows to current partitions. As for $C(10)$, it will also update the current associated controller of $M1$ to be $C(3)$ at $t2$, and notify $C(16)$ to end the previous session for $M1$.

Note that as a mobile IoT device in the urban scenario, $M1$ may leave the partition of $C(3)$ and re-enter again with unpredictable mobility, therefore in the UbiFlow overlay structure, there is an extra entry in the finger table of each controller to label all the supervisory controllers of its current and previously associated IoT devices with a TTL (time-to-live). Hence, when previously associated IoT device enters its partition again, the controller does not need to initiate another multi-hop request to localize the supervisory controller. Instead, the controller can localize the supervisory controller using fast lookup in its finger table, which will further save the communication cost and improve the efficiency of handover.

Theorem 2. *The mobile lookup cost to find the previous associated controller for an IoT device in UbiFlow could be either* $O(\log N)$ *or* $O(2)$.

Proof. Mobility management of UbiFlow is organized by a consistent hashing based overlay structure. Theorem 1 has proven that the usually lookup cost in this kind of structure is bounded to $O(\log N)$. Since supervisory controller records previous association of supervised devices, the normal mobile lookup cost to find the previous associated controller for an IoT device is $O(\log N) + 1$, by localizing supervisory controller first and then requesting local lookup in the supervisory controller, which is still bounded to $O(\log N)$. If the supervisory controller was found before and has been recorded in the local finger table as the additional information, the lookup cost for the corresponding mobile device is then just a local lookup as $O(2)$, with one step to reach the supervisory controller and one step to request local lookup in the supervisory controller. □

3.4 Scalable Control

To achieve scalable mobility management by multiple controllers in distributed SDN, we focus on the Join and Leave operations of controllers, as follows:

1) *Join*: When a new controller with ID n joins an existing SDN with multiple controllers, it first identifies its successor by performing a lookup in the SDN according to its ID. Once it localizes the successor, it selects the successor's keys that the new controller is responsible for. After this, the new controller sets its predecessor to its successor's former predecessor, and sets its successor's predecessor to itself.

Meanwhile, an initial finger table will be built in the joined controller by performing lookup points $(n + 2^i - 1)$, for $i = 1, 2, \ldots, e$, where e is the number of finger print entries.

2) *Leave*: When a controller with ID n wants to leave an existing SDN, it first moves all keys that the controller is responsible for to its successor. After this, it sets its successor's predecessor to its predecessor, and sets its predecessor's successor to its successor. For consistency purposes, before the controller leaves the distributed network, the SDN related control information (e.g., network status and flow status) in the controller will be copied to its successor by default, and other controllers can later update their finger tables by replacing controller n with its successor in the corresponding entry. If the controller also performs as the supervisory controller for some IoT devices, its successor will be also designated as the new supervisory controller for these IoT devices, and it records the existing mobility information from the leaving controller.

3.5 Fault Tolerance

To handle failure in the distributed SDN, we tackle failures of different components in UbiFlow. As for controller level failure, we adopt data replication to achieve robust control. That is, we copy the finger tables and data-bases from local controller n to its r live successors in the UbiFlow overlay structure, by searching key $(n + 2^{i-1})$, for $i = 1, 2, \ldots, r$. These r successors also update these replications periodically. Therefore, if the local controller fails, we can find a new successor that still can provide the control service.

As for finger-table level failure, we adaptively choose alternate paths while routing. That is, if a finger does not respond, we take the previous fingers in the local table, or the finger-table replicas from one of the r successors. In addition, the local finger table also performs self-check to refresh all fingers by periodically looking-up the key $(n + 2^{i-1})$ for a random finger entry i. The periodic cost is $O(\log N)$ per controller due to the finger refresh.

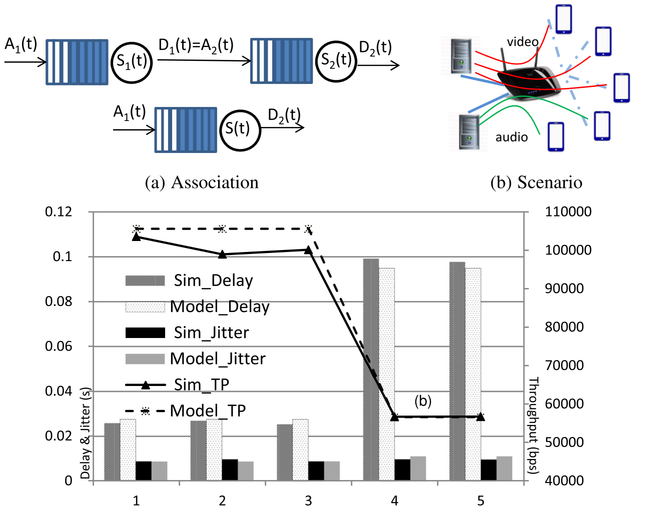
As for access-point failure, we designate an associated controller to detect the failure and redirect flows going through failed access points to others in its partition. We address the assignment of the best available access point to the IoT device in Section 4.

4 Flow Scheduling in UbiFlsow

Given flow requirements (e.g., delay, throughput, jitter) from an IoT device, the UbiFlow controller needs to find the best available access point that can both satisfy the flow request of the IoT device and guarantee optimal network performance of the whole partition. The relevant UbiFlow design to achieve robust flow scheduling is described in this section.

4.1 Network Calculus based Partition View

The UbiFlow controller of each partition needs a partition view to obtain current network status for flow scheduling. To guarantee the performance of software defined IoT with various flow requirements, UbiFlow controller uses Network Calculus [54] to model the arrival traffic $A(t)$, served traffic $S(t)$, and departure traffic $D(t)$ on a network node during the time interval [0,t) as the partition view in its partition. We assume that each node has a constant capacity R and can provide a service curve $S(t) = R[t - T]^+$, where R is the capacity (transmission rate), $[x]^+ = \max\{0, x\}$, and T is the transmission delay, which is the time between the first bit of the packet entering a queue and the last bit leaving the transmitter. T depends on R, the length of this packet, and the amount of data currently in the queue. We can use min-plus convolution on arrival and service curves, to generate a departure curve: as $D(t) = A(t) \otimes S(t)$, which means: $D(t) \geq \inf_{s \leq t}(A(s) + S(t - s))$.



(a) Association

(b) Scenario

(c) Result

Fig. 6. Network calculus based partition view in UbiFlow.

If there is more than one flow going through a node, all flows share the same transmission service. Here we assume each intermediate node has a FIFO scheduler, in which packets are served in the sequence as they arrived. Flow i will have a leftover service curve:

$$S_i = \frac{\theta^i}{\sum_{j \neq i} \theta^j} R[t - T]^+, \tag{1}$$

where R is the capacity of the downlink of this node (transmission rate), and θ is the weight of each flow; In a multihop path, the departure curve of the current hop is the arrival curve of the next hop as shown in Fig. 6a, and a combination service curve along the path $S(t)$ can be obtained by iteratively adding each node's service curve using the associative operation in min-plus convolution, as follow:

$$S(t) = S_1 \otimes S_2 \otimes \ldots \otimes S_n. \tag{2}$$

In order to provide a fine grained partition view of the traffic, UbiFlow models the traffic as a set of discrete points (each point represents a packet) in Network Calculus. It assumes that the profile of each flow (e.g., packet length and sending time) is known at each sender, and each packet is served by the service curve $S(t)$ with a constant capacity R and a delay T. At packet arrival time, we examine the current queue state in terms of how many packets are in the queue and their lengths. The delay T is the transmission time of all packets that are already in the queue. Hence the total delay of a packet consists of two parts: one is T and the other is the transmission (service) time of the packet itself. In this way, we can get an approximate end-to-end delay for each packet. To verify this model, we examine three QoS parameters: delay, throughput, and jitter. For each flow, we profile it with points at the sender side to plot the curve. Once we get the arrival curve $D(t)$ of flow i at the destination node by the modified Network Calculus model, we compare it with flow i's initial arrival curve $A(t)$. Each point (packet) will suffer from a delay and the final arrival time is recorded. The average delay, average jitter, and total throughput for each flow can be calculated by UbiFlow controller accordingly. As shown by the test in Fig. 6b for a two-hop network consisting of one video server and one audio server, one router and 5 clients, each server connects to the router via a 100 Mbps Ethernet link while each client connects to the router via a 2 Mbps 802.11b wireless link. Each server provides either a video streaming service [55] or a Skype voice service [56] to one of the clients. The corresponding test results in Fig. 6c show consistent performance with our Network Calculus based model: the average error rate of the delay, jitter, and throughput (i.e., TP) are 5, 8, and 3 percent respectively. Therefore, the fine grained model can be used by UbiFlow controller to obtain the partition view.

4.2 IoT Multinetworks Matching

After obtaining the partition view of the current network status from the network calculus model, the UbiFlow controller can manage handover between heterogeneous access networks by assigning newly joined mobile IoT devices to the best access point, based on the current multinetwork capacity in the controlled partition, the supported radio access technologies and the types of services the mobile devices are requesting.

We formulate the assignment of a set of newly joined mobile IoT devices $\mathbb{MD}$ to a set of access points $\mathbb{AP}$ as a generalized assignment problem (GAP). Each access point j is characterized by a residual bandwidth capacity function $B(j)$, and each mobile device i is characterized by a bandwidth demand function $d(i,j)$ that describes the bandwidth demand of device i when assigned to access point j. A utility function $u(i,j)$ measures the benefit obtained by the system as a result of assigning a mobile device i to access point j. The assignment problem is formulated as:

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in\mathbb{AP}}\sum_{i\in\mathbb{MD}} u(i,j)x(i,j) \\ \text{subject to} \quad & \sum_{i\in\mathbb{MD}} d(i,j)x(i,j) \leq B(j), \forall j \in \mathbb{AP} \\ & \sum_{j\in\mathbb{AP}} x(i,j) \leq 1, \forall i \in \mathbb{MD} \\ & x(i,j) = 0 \; or \; 1, \forall i \in \mathbb{MD}, \forall j \in \mathbb{AP}, \end{aligned} \tag{3}$$

where $x(i,j) = 1$ if device i is assigned to access point j or 0 otherwise.

Note that the optimization takes place per partition and no global variables are shared between partitioned optimizations, the optimizations at each partitions are thus independent and performance of the optimization per partition does not degrade as the number of partitions grows larger. As the set of access points and especially the set of mobile devices changes dynamically, the assignment is done in an adaptive time-window based manner. The assignment is performed at the end of each window using (1) the capacity, demand and utility functions evaluated at that time, (2) the set of newly joined mobile devices within the window of time, and (3) the set of active access points at that time. Algorithm 1 is an adaptation of the GAP approximation in [57] combined with a greedy heuristics for the Knapsack problem that sorts items based on their utility-to-demand ratio and tries to pack as much high-ratio items as possible. It takes the sets of access points and devices, a matrix of utilities and demands, and a vector of capacities as an input. It starts by checking the residual capacity (capacity at the end of the time window) of the set of access points ($\mathbb{AP}$) against the demand vector ($D_*[ap]$) of mobile devices with respect to that access point type, and creating a feasible set of access points ($\mathbb{AP}_f$) by selecting those that can at least satisfy the minimum demand. It then initializes the assignment vector (X) and iteratively computes the assignment as follows. For each access point, it creates a utility vector from the utility matrix using either the original utility value or the difference in utility depending on whether the mobile device is assigned to an access point in the previous iteration. The utility-to-demand ratio is then computed using the utility vector, and the set of mobile devices ($\mathbb{MD}$) is sorted in non-decreasing order based on this ratio. Using this ratio and a greedy Knapsack packing scheme, the mobile devices are assigned to the current access point. This is repeated until all access points are exhausted. The vector X is the result of the assignment where $X[md]$ indicates that mobile device md is assigned to access point $X[md]$ if $X[md]$ is not -1.

The UbiFlow controller determines each mobile device's compatibility (i.e., support for the radio access technology used by the access point) with access points and requirement with respect to quality of service such as bandwidth demand (d) and the maximum tolerable latency (l_t) based on the types of services the device is trying to access. The demands of IoT devices can be obtained during their request processes, and the partition status can be derived from the network calculus model, as described in Section 4.1. If there is compatibility between a device and an access point, the degree of satisfaction of a mobile device, if assigned to the access point, with respect to these requirements is modeled by utility functions namely u_d and u_l respectively. In addition, a utility function u_a that measures the load (i.e., the number of mobile devices) on an access point is used in order to take the degree of distribution of load into consideration so that one capable access point will not be overloaded. Given an access point with latency l and N number of mobile devices already assigned to it, the utility functions are as follows:

$$\begin{aligned} u_d(i,j) &= \log\left(1 + \frac{B(j)}{d(i,j)}\right), d(i,j) > 0 \\ u_l(i,j) &= \log\left(1 + \frac{l_t}{l}\right), l > 0 \\ u_a(j) &= \log\left(1 + \frac{|\mathbb{MD}|}{N + |\mathbb{MD}|}\right), \mathbb{MD} \neq \emptyset. \end{aligned} \tag{4}$$

Using these utility functions, the controller computes the utilities for each potential assignment, normalizes them and then computes the total system utility using predefined positive weights that capture the significance of each type of utility as:

$$u(i,j) = \begin{cases} w_d\hat{u}_d(i,j) + w_l\hat{u}_l(i,j) + w_a u_a(j) & \text{compatible} \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where $\hat{u}_d$ and $\hat{u}_l$ are the normalized utilities and $w_d + w_l + w_a = 1$. It then performs assignments that would maximize the overall system utility.

Algorithm 1. Mobile Device to Access Point Assignment

```
Input: $\mathbb{AP}$, $\mathbb{MD}$, U, D, $C$
Output: $X$
for $ap \in \mathbb{AP}$ do
    if $C[ap] \geq min\{D_*[ap]\}$ then
        Add $ap$ to $\mathbb{AP}_f$
    end if
end for
for $r = 1$ to $|\mathbb{MD}|$ do
    $X[r] \leftarrow -1$
end for
for $ap \in \mathbb{AP}_f$ do
    for $md \in \mathbb{MD}$ do
        if $X[md] == -1$ then
            $U_{ap}[md] \leftarrow U[md][ap]$
        else
            $U_{ap}[md] \leftarrow U[md][ap] - U[md][X[md]]$
        end if
        Compute utility-to-demand ratio vector: $R_{ap}[md] \leftarrow \frac{U_{ap}[md]}{D[md][ap]}$
    end for
    Sort $\mathbb{MD}$ such that $R_{ap}[md]$ is in non-increasing order
    $b \leftarrow min\{q \in \{1, ldots, |\mathbb{MD}|\} : \sum_{r=1}^{q} D_{ap}[r] > C[ap]\}$
    for $q = 1$ to $b - 1$ do
        $X[q] \leftarrow ap$
    end for
end for
```

Theorem 3. *The time complexity for the assignment of* $|\mathbb{MD}|$ *mobile devices to* $|\mathbb{AP}|$ *access points using Algorithm 1 is bounded by* $O(|\mathbb{AP}||\mathbb{MD}|\log(|\mathbb{MD}|))$ *when an* $O(|\mathbb{MD}|\log(|\mathbb{MD}|))$ *algorithm is used to sort the utility-to-demand ratio vector.*

Proof. The running times of the first and second loops are proportional to $|\mathbb{MD}||\mathbb{AP}|$ and $|\mathbb{MD}|$ respectively. Suppose the running time of the sorting algorithm used to sort the utility-to-demand ratio vector is $f(|\mathbb{MD}|)$. Consequently, the running time of the third loop is proportional to $|\mathbb{AP}|(|\mathbb{MD}| + f(|\mathbb{MD}|) + |\mathbb{MD}| + |\mathbb{MD}|)$. Hence, the time complexity of the algorithm is $O(|\mathbb{AP}|f(|\mathbb{MD}|) + |\mathbb{AP}||\mathbb{MD}|)$. Thus, when an $O(|\mathbb{MD}|\log(|\mathbb{MD}|))$ algorithm is used to sort the utility-to-demand ratio vector the time complexity of Algorithm 1 is $O(|\mathbb{AP}||\mathbb{MD}|\log(|\mathbb{MD}|))$. □

Note that above optimized IoT multinetwork matching scheme also works for mobile handover in the same partition (but with a different access point). When a IoT device moves out of the communication range of its associated access point or the quality of flow service is degrading obviously with the access point, these mobile information can be collected by our Network Calculus model as partition view. Once the controller in the partition obtains the new partition view after at the end of a time window, it will perform above optimization to choose a new access point for the IoT device in the partition.

4.3 Adaptive Matching Window

The window of time the controller waits before performing the next assignment plays a significant role in achieving optimal assignments. If the controller performs instantaneous assignments, devices that would maximize the system's profit would be assigned to a less suitable access point or left unassigned if there is not enough capacity left. Conversely, a longer window would result in discovering more devices and thereby facilitating better assignments provided that the residual capacity of the access points is not exceeded by the demand of the newly discovered devices. However, the choice of the exact length requires consideration of the residual capacity of the access points; the rate of arrival of mobile devices to the network and their associated demand; and the waiting period mobile devices can tolerate before they are assigned to an access point. If the arrival of devices to the network is characterized by a mean rate λ and the expected demand of the devices by a mean demand μ, the upper bound for the best (with respect to achieving optimal assignment) window length can be estimated as $\frac{\sum_{j \in \mathbb{AP}} B(j)/\mu}{\lambda}$. This bound could be dynamically estimated by choosing appropriate models for the arrival rate of mobile devices to the network and their demands, and learning the parameters of these models using a maximum likelihood estimation. The actual window length can then be decided by taking this bound and the waiting period mobile devices can tolerate before they are assigned to an access point into consideration (e.g., the minimum of the two values).

We propose that in a real-world situation the length of the time window $t + 1$ is calculated after the time window t elapses, and the calculation will take into account flow data calculated during the window t. Data on the arrival of devices within the window t is used to calculate the expected mean device arrival rate λ. Demand data within the window t is used to calculate the expected demand of devices (μ) within the next window. μ and λ are then used to calculate the length of the optimal window according to above upper bound formula. The length of window $t + 1$ is then set to be the minimum value between the optimal time window length calculated by the controller, and the tolerant waiting period of mobile devices to associate with an access point in the corresponding partition. Above operations for the controller to select an adaptive matching window of time are described in Algorithm 2.

4.4 Load Balancing

One key limitation of existing SDN systems is that the mapping between a switch and a controller is statically configured, making it difficult for the control plane to adapt to temporal and spatial traffic load variations. If the switch to controller mapping is static, a controller may become overloaded if the switches mapped to this controller suddenly observe a large number of flows, while other controllers remain underutilized. Furthermore, the load may shift across controllers over time, depending on the temporal and spatial variations in traffic conditions. As load imbalance occurs, it is desirable to migrate a switch from a heavily-loaded controller to a lightly-loaded one. However, such a migration operation is not supported natively in current de faco SDN OpenFlow standards. Following our architecture as illustrated in Fig. 2, UbiFlow consists of a cluster of autonomous controllers that coordinate amongst themselves to provide a consistent control logic for the entire network. We can design a robust load balancing scheme based on the UbiFlow architecture to dynamically shift the load across switches and controllers.

Algorithm 2. Adaptive Matching Window

```
Input: The arrival rate sample set $\mathbb{R}$ and demand sample set $\mathbb{D}$ in
    the time window $t-1$; The set of active access points $\mathbb{AP}$
Output: The length $\mathbf{T}$ of time window $t$
for sample $r_i$ in $\mathbb{R}$ do
    $l(\lambda)* = p(r_i|\lambda)$
end for
for sample $d_i$ in $\mathbb{D}$ do
    $l(\mu)* = p(d_i|\mu)$
end for
Solving maximum likelihood function $l(\lambda) = \prod_{i\in\mathbb{R}} p(r_i|\lambda)$ as
    $\hat{\lambda} = \arg\max_{\lambda} \sum_{i\in\mathbb{R}} \ln p(r_i|\lambda)$
Solving maximum likelihood function $l(\mu) = \prod_{i\in\mathbb{D}} p(d_i|\mu)$ as
    $\hat{\mu} = \arg\max_{\mu} \sum_{i\in\mathbb{D}} \ln p(d_i|\mu)$
if $\sum_{j\in\mathbb{AP}} B(j)$is not exceeded by the demand of a device then
    $t = \frac{\sum_{j\in\mathbb{AP}} B(j)}{\hat{\lambda}\hat{\mu}}$
    Minimum tolerate time of devices is $t'$
    return $\mathbf{T} = min\{t, t'\}$
else
    The capacity of access points is not enough
end if
```

Given a controller n, if new flow requests, collected from local IoT devices, cause traffic imbalance (e.g., over maximum capacity, longer process delay) controller n needs to switch the flow to a lightly-loaded controller. However, the usual linear balancing scheme that relays the flow request to one of its r successors is not robust enough in the mobile SDN scenario, because the r successors have locally loaded flows and these may be heavily-loaded as well. Furthermore, the fault tolerant scheme presented in Section 3.5 will generate redundant data in the r successors, so additional flow requests from other partitions tend to cluster the requests of the flows into contiguous runs, which may even overlap in our circular overlay structure. In addition, because of the importance of the supervisory controller, if the supervisory controller is heavily-loaded and cannot accept other newly joined IoT devices, we also need a scheme to mitigate the traffic flow on this supervisory controller by directing the flows for new IoT devices to other controllers as a backup supervisory controller. Meanwhile, we need a consistent scheme for other controllers to be able to localize these backup supervisory controllers.

To avoid the linear clustering of heavily-loaded controllers and guarantee system consistency in the UbiFlow overlay, we use double hashing to balance a large number of flow requests and distribute them fairly in the overlay structure. The load balancing using our double hashing solution is realized in Algorithm 3. Specifically, different from the hash function h used in the finger key search, we choose a secondary hash function, h' for collision handling. If h maps some finger key k to a controller $C[i]$, with $k = h$ (Mobile ID), that is already heavily-loaded, then we iteratively try the controllers $C[(i + f(j)) \bmod P]$ next, for $j = 1, 2, 3, \ldots$, where $f(j) = jh'(k)$. In this scheme, the secondary hash function is not allowed to evaluate to zero; a common choice is $h'(k) = q - (k \bmod q)$, for some prime number $q < P$. Also, P should be a prime number.

Algorithm 3. Load Balancing Using Double Hashing

```
Input: The Mobile ID set of IoT devices $\mathbb{MD}$
Output: The controller ID
for Mobile ID in $\mathbb{MD}$ do
    $k = h$(Mobile ID)
    $k$ is mapped to $C[i]$ which is a supervisory controller
    if $C[i]$ is already heavily-loaded then
        for $j = 1$ to $n$ do
            $f(j) = j(q - (k \bmod q))$
            The backup supervisory controller of $k$ is $C[(i + f(j))$
                mod $P]$
            if $C[(i + f(j))$ mod $P]$ is not heavily-loaded then
                The $C[i]$ records that $k$ is mapped to controller
                    $C[(i + f(j))$ mod $P]$
                return The ID of controller $C[(i + f(j))$ mod $P]$
            end if
        end for
    end if
    return The ID of controller $C[i]$
end for
```

Theorem 4. *The time complexity of using double hashing to avoid linear of heavily-loaded controllers is $O(1)$ or $O(n)$ in UbiFlow.*

Proof. The bound of a successful search for a finger key k to a controller follows a probe sequences formed by the secondary hash function when it was first inserted. so if k is the $(i+1)$th key to insert into the UbiFlow overlay network, the average cost of a successful search is: $\frac{1}{S}\sum_{i=0}^{S-1} \frac{1}{1-i/T} = \frac{T}{S}\sum_{i=0}^{S-1} \frac{1}{1-i/T}\frac{1}{T} \approx \frac{T}{S}\sum_{i=0}^{S} \frac{1}{1-i/T}\frac{1}{T}$, where T is the size of the overlay network, and S is the number of keys that have been inserted in the overlay network, and $\alpha = S/T$ is the load factor meaning a measure on the load of finger keys in the overlay network. We can convert the summation process to following calculus process: $\frac{T}{S}\sum_{i=0}^{S} \frac{1}{1-i/T}\frac{1}{T} = \frac{1}{\alpha}\int_0^{\alpha} \frac{1}{1-x} dx = \frac{1}{\alpha}\ln(\frac{1}{1-\alpha})$. According to this, the average search cost is independent of T when α is less than 1, and its time complexity follows the constant order $O(1)$. However, the average search cost is bounded only by T when

α is close to 1, and its time complexity follows the linear order as $O(n)$. □

Note that, for supervisory controller, the heavy load status may cause local failure, but its mobility records for IoT devices are important for the mobility management. When UbiFlow observes the load imbalance in a supervisory controller, it also uses the double-hashing scheme to copy the mobility information to other controllers as a backup. By this way, UbiFlow can effectively protect the mobility information, in case consecutive failures happen and the redundancy scheme in Section 3.5 fails.

5 Performance Evaluations

We have implemented a prototype of UbiFlow, and evaluated its performance on flow scheduling and mobility management by both simulation and real testbed experiments. The simulation is performed by the OMNeT++ network simulator [58], and the real testbed is built based on the Orbit wireless testbed [59], both with OpenFlow support.

5.1 Implementation Methodology

The specific implementation methods and details of our UbiFlow prototype are explained as follows.

5.1.1 Compatibility With Heterogeneous Access Technologies

In traditional SDN, the SDN controller or OpenFlow switch itself does not support heterogeneous networks with different radio access technologies. In our UbiFlow system, we use encapsulation to mark the category of radio access on packet level, and make SDN work in compatibility with heterogeneous networks after decapsulation. Specifically, when an AP receives new packets, the corresponding AP attributes, such as its access category and spectrum information will be encapsulated into these packets and then forwarded to the OpenFlow switch. Since the SDN controller can obtain a copy of these packets from the switch through the OpenFlow protocol, it then parses the radio access information from these packets by decapsulation process. On one hand, the information can assist the controller to recognize current category of AP used by a specific IoT device, and direct control message or service flow to the corresponding AP. On the other hand, the information can be used by the controller to run statistical analysis on the usage status of heterogeneous APs through our Network Calculus approach as explained in Section 4.1, and update the partition view to obtain whole-partition capabilities and perform optimal assignment of heterogeneous APs to IoT devices.

5.1.2 Co-Existence of Heterogeneous Spectrum Access

The cognitive radio capability is mainly executed by the SDN controller and the controller can enable proper radio interface in the IoT devices to access corresponding spectrum. Specifically, a newly joined IoT device will first choose a default radio access to connect an AP by customized setting and send out its flow requests. As explained above, once the AP receives and encapsulates packets from the IoT device, the controller can decapsulate these packets for statistical analysis on the residual capability of corresponding AP, and summarize current spectrum usage of heterogeneous APs in its partition. Then, the controller can differentiate flow scheduling based on the requirements per-device as well as whole-partition capabilities, and later obtain the optimized selection of access points in multinetworks to satisfy IoT flow requests, while guaranteeing the network performance in each partition. If the initial AP connected by the IoT device is overloaded, the controller will assign another AP with specific spectrum in its partition to the IoT device. Once the IoT device receives the AP update message from the controller, it will disconnect the initial AP by closing current radio interface, and switch to connect the assigned AP in designated spectrum by opening a new radio interface.

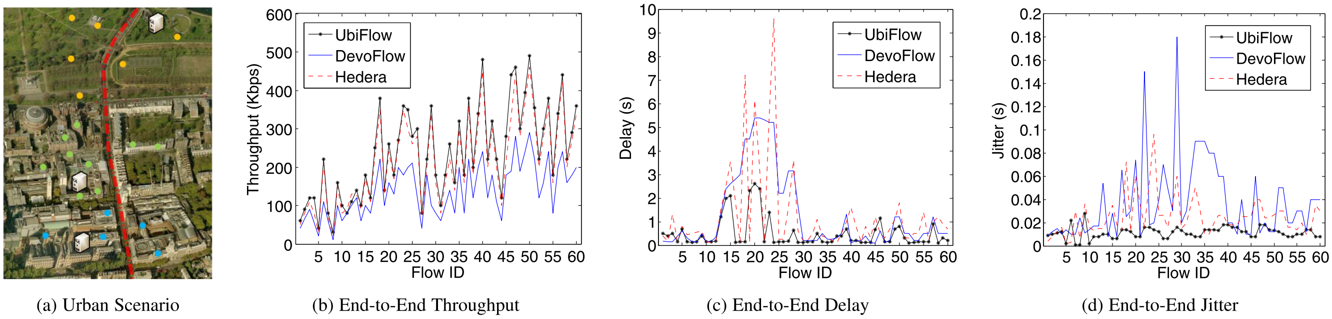
5.1.3 Communications Between SDN Controller and IoT Devices

Several control traffic have been specifically added between SDN controller and IoT devices for distributed control, which are mainly composed of following types of control messages. The first one is the AP request message initiated by the IoT device to request AP assignment from the controller. The second one is the AP response message from the controller to assigns optimal AP to the IoT device. The third one is the AP update message when IoT devices roaming in different SDN partitions, so that the SDN controller can detect the mobility behavior and update the connections of IoT devices with APs. The fourth one is the flow request message sent from the IoT device to the SDN controller to request different types of data flow, so that the controller can localize proper server with corresponding data service and provide flow path to the server, and later the server directly transmits the data flow to the IoT device.

The detailed process of managing control traffic varies in each transmission phase between SDN controller and IoT devices. During the communication phase from SDN controller to OpenFlow switch, all types of control messages are loaded into the Packet-Out message which is a message type of the default southbound OpenFlow protocol, and the control traffic share the same channel with the data traffic. During the phase from OpenFlow switch to APs, these control messages are forwarded to different APs through the corresponding AP identifier encapsulated in each control message, and these messages share the same channel too. After the control messages arrive at an AP, the AP uses multiple control channels to transmit the control messages to IoT devices through corresponding types of communication spectrums according to the spectrum information encapsulated in the control message.

5.1.4 Interactions Between SDN Controller and APs

As for the SDN controller to interact with the AP, the SDN controller employs the Network Calculus to obtain the partition view, and execute AP assignment optimization algorithm. After this, it sends the AP update message to the AP through the OpenFlow switch. As for the AP to interact with the SDN controller, the AP encapsulates radio access and spectrum information into the packet received from the



(a) Urban Scenario (b) End-to-End Throughput (c) End-to-End Delay (d) End-to-End Jitter

Fig. 7. Mobile flow scheduling in UbiFlow.

connected IoT devices, and forward these packets to the SDN controller through the OpenFlow switch for the statistical analysis on partition view. At the same time, the AP waits the AP update message come from the SDN controller. If the optimal assignment is the AP itself, it continues to maintain connection with the IoT devices. Otherwise, the AP will disconnect the IoT devices and update the IoT devices to connect the newly assigned AP. Therefore, the SDN controller interacts with the APs through OpenFlow switch, and these interactions are mainly driven by the access events and flow requests from IoT devices.

5.2 Simulation Results

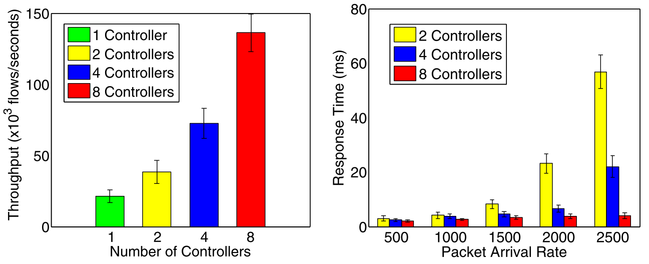
Recently, OMNeT++ has incorporated an OpenFlow extension implemented in the INET framework for SDN simulation [60]. However, its controller only supports the wired data center networks and lacks mobility management. We have changed its data plane to incorporate the inherent advantages of OMNeT++ on setting heterogeneous wireless networks, and extended its control plane to support multiple SDN controllers. The UbiFlow framework has been implemented in these controllers to support mobility management, flow processing and flow forwarding under distributed environments.

To verify the performance of UbiFlow in urban scenario, our simulation is based on a popular area in the city of London, which consists of several parks, universities, and museums, as shown in Fig. 7a. This area is usually crowded by high density of tourists, students and workers, with large number of IoT devices and various types of flow requests. Therefore, in our first set of evaluation, three controllers have been deployed in park partition, university partition and museum partition, respectively, for flow scheduling and mobility management. The backbone topology consists of 3 data servers (each of the three data servers provides either file sharing, audio, or video streaming services), 3 switches (each switch has a 1 Gbps Ethernet link to one server; each controller directly controls one switch), and 20 access points (each access point has one 100 Mbps Ethernet link to every switch). There are three types of access points: WiMAX, WiFi and Femtocell, with data rates 30 Mbps, 10 Mbps, and 2 Mbps respectively. Each IoT device has three network interfaces to directly connect with corresponding access points, and at each time instance only one interface can be used.

5.2.1 Handover in UbiFlow

In our first set of simulation, as shown in Fig. 7a, there are 5 access points (orange dots) in the park partition, 9 access points (green dots) in the university partition, and 6 access points (blue dots) in the museum partition. Some of these access points are already under heavy traffic load, and others still have enough capacity. Assume there are 60 IoT devices sending new flow requests at a time, and they are moving along the red path. 10 of them request file sharing services, 20 of them request audio services, and 30 of them request video streaming services. In our evaluation, file sharing flows are modeled by sending Constant Bit Rate with packet length uniformly distributed in [100, 1000] bytes with period T, the latter uniformly distributed in [0.01, 0.1] seconds. Audio and video streaming flows are from real traffic traces [55], [56]. For practical applications, the file sharing service requires large throughput, the audio service requires low delay, while the video streaming service requires low jitter. We evaluate our UbiFlow scheduling and compare it with other two common scheduling algorithms used in SDN world: DevoFlow [26] and Hedera [25]. The former tries to accommodate as many flows as possible into a single link to maximize the link utilization. Instead, the latter assigns flows into a link so that the total amount of the flows are proportional to the capacity of the link.

As shown in Fig. 7, we have totally 60 flows (each of 60 end devices has one flow): flows 1-10 are file sharing, flows 11-30 are audio, and flows 31-60 are video streaming. Fig. 7b shows the comparison of flow throughput. For file sharing flows, UbiFlow outperforms DevoFlow by an average of 67.21 percent, while it has an average of 15.91 percent throughput increase if compared with Hedera. The reason is that in wireless links when link utilization exceeds a threshold, the packet drop rate increases dramatically. The load balancing scheme in UbiFlow uses the controller to schedule flows according to the utilization status of each access point; therefore it can achieve comparably fair allocation of flow traffic to decrease packet drop rate. Fig. 7c shows that for audio flows, our proposed algorithm can improve the end-to-end delay performance by 72.99 and 66.79 percent, compared to DevoFlow and Hedera respectively. Audio flows have bursty traffic patterns; it might not have big data volume, but if two flows are scheduled with similar bursty patterns in the same link, a large delay occurs. Due to the traffic-aware dynamic flow scheduling scheme, UbiFlow can schedule flows both by the consideration of partition load and device requirement; therefore it can reduce the impact of flow interference. Fig. 7d shows that video streaming flows have an average 69.59 and 49.72 percent less jitter with UbiFlow than DevoFlow and Hedera. Because of the holistic solution in flow scheduling



(a) Flow Throughput (b) Flow Delay

Fig. 8. Scalability in UbiFlow.

and mobility management, distributed controllers in UbiFlow can provide more stable video flow for IoT devices.

5.2.2 Scalability in UbiFlow

In the implementation of OpenFlow, the Packet-In message is a way for the OpenFlow switch to send a captured packet to the controller. A flow arrival resulting in sending a Packet-In message to the controller. In the second set of simulation, we use Packet-In message to evaluate the scalability of flow scheduling by UbiFlow. For better scalability evaluation, we add more controllers in the above urban scenario. In addition, for every controller in its partition, the controller is directly connected with 3 to 5 switches, and controls 20 to 50 access points with various heterogeneous interfaces. For each controller, we send 10000 consecutive Packet-In messages to it and plot the throughput of UbiFlow with varying number of controllers, as shown in Fig. 8a. We observe that adding controller nodes increases the throughput almost linearly. This is because in the architecture of UbiFlow, as shown in Fig. 2, each controller mainly controls the traffic flows in its own partition. However, if there is an imbalance in one controller, other controllers with lightweight traffic also can help to migrate the flows to their partitions by physically partial connected switch and the UbiFlow overlay structure. To further illustrate the scalability of UbiFlow, we also plot the response time behaviour for Packet-In messages with changing flow arrival rate, as shown in Fig. 8b. We repeat the experiment while changing the number of controller nodes. As expected, we observe that response time increases marginally up to a certain point. Once the packet generation rate exceeds the capacity of the processor, queuing causes response time to shoot up. This point is reached at a higher packet-generation rate when UbiFlow has more number of nodes.

5.2.3 Fault Tolerance in UbiFlow

When the number of controllers increase, we also care about the performance of UbiFlow on fault tolerance, especially in mobility management. In the third set of simulation, we evaluate the ability of UbiFlow on lookup of mobile nodes after a large percentage of controllers fail simultaneously. We consider a 10^3 controller network that stores 10^5 keys, and randomly select a fraction p of controllers that fail. Note that in mobility management, UbiFlow classifies controllers into supervisory controllers and associated controllers, where supervisory controllers record the updated mobility information of IoT devices. To obtain this information, associated controllers need to first localize the supervisory controllers for mobile lookup. In our setting, the failed controllers could be supervisory controllers and associated controllers. A correct mobile lookup of a key is one that finds the supervisory controller that is responsible for the key. Fig. 9 compares UbiFlow with Chord, by the mean lookup failure rate and the confidence interval as a function of p. In Chord, the lookup failure rate is almost exactly p. Since this is just the fraction of keys expected to be lost due to the failure of the responsible nodes. UbiFlow can further improve the performance of Chord on mobile lookup both in mean lookup failure rate and the confidence interval, because of its consistent overlay scheme on mobility management. UbiFlow uses redundancy to resist failure, by coping the mobility information from local controller to its live successors in the overlay structure. Meanwhile, when a supervisory controller fails because of load imbalance, UbiFlow can use double-hashing scheme to localize the backup supervisory controller for effective lookup of mobility information.

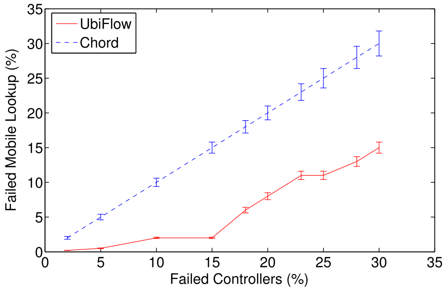
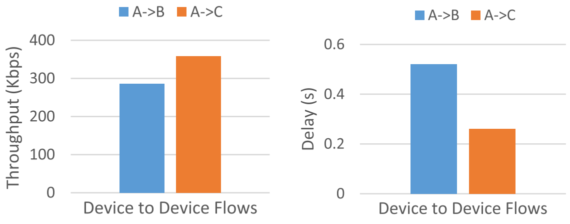


Fig. 9. Fault tolerance in UbiFlow.

5.2.4 Flow Scheduling Between IoT Devices

Though we consider device to server flows primarily in this paper since it is the more common case of SDN-based communications, as discussed in Section 2 another type of IoT flow potentially existed in software-defined IoT is the data flow between IoT devices located within different partitions. This type of IoT flow needs to be scheduled through inter-partition communication. Utilizing the connected switches, controllers can coordinate to direct the flow initiated from one partition to a different access point in another partition. Following the same settings of our evaluation in Fig. 7, we choose one device A in the university partition as sender, one device B in the university partition as receiver, and one device C in the park partition as another receiver. All the three IoT devices are assigned with WiFi access points and the data flows sent from A to B, and A to C are the same video streaming. The average performance collected in one minute is presented in Fig. 10. Fig. 10a compares the end-to-end flow throughput between $A \rightarrow B$ and $A \rightarrow C$



(a) Flow Throughput (b) Flow Delay

Fig. 10. Device to device flow scheduling.

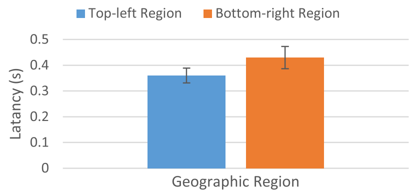


Fig. 11. Flow latency of cross-region scheduling in UbiFlow.

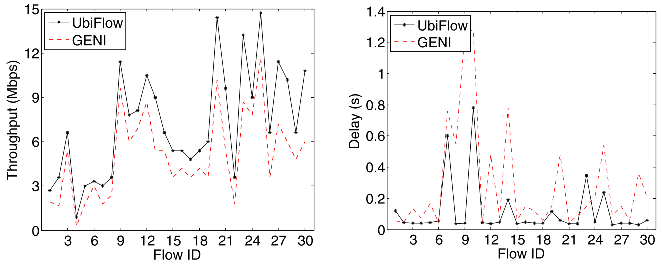
transmissions from the receiver aspect. Fig. 10b shows the difference of end-to-end delay performance between the two device to device data flows. It is interesting that in this set of evaluation intra-partition ($A \rightarrow B$) has worse performance than inter-partition ($A \rightarrow C$) flow scheduling both on throughput and delay metrics, even though the flow path from $A \rightarrow C$ is longer than $A \rightarrow B$. This is due to the fact that there are more IoT devices crowded in the university partition than the park partition in our settings. The density of IoT devices in the university partition results in higher demand of communication resources and heavier processing status in its corresponding controller, therefore generating lower throughput and higher delay along the path from controller to receiver in the same partition, in comparison with the path from controller to receiver in the park partition that has more communication and controller resources available to use.

5.2.5 *Cross-Region Flow Scheduling*

We have also verified the impact of UbiFlow overlay network on large-scale urban network scheduling across different geographic regions. Specifically we evaluate how the overlay network of the controller system would impose latency in the mobile IoT device to server communication. Our simulation is based on the mobile scenario, as shown in Fig. 5, where the top-left region has 400 IoT devices and 100 access points, and the bottom-right region has 200 access points and 600 IoT devices (the limitation of OMNeT++ is that it supports up to nearly 2000 nodes in simulations). A mobile IoT device associated with video streaming service is frequently roaming across the two regions and their partition areas. The average flow latency and its standard deviation in each region is presented in Fig. 11. We observe that the scheduled flow latency for this device in the bottom-right region does not change a lot in comparison with that in the top-left region. The stable performance is achieved by our specific design for large-scale mobility in UbiFlow, where controllers is geographically localized to the extent that propagation delay between them is within the same order of magnitude. In addition, when an IoT device enters a new region, a new supervisory controller will be assigned in the region to manage its mobility, so that the device can avoid too much cross-region long-distance information exchanges with old supervisory controller. Therefore, in case that a mobile IoT device frequently roams inside a region (with multiple partitions) and across different regions (separated by long distance), the latency from distributed UbiFlow overlay can still maintain within the same order of magnitude.

5.3 Testbed Experiment Results

In our real testbed experiments, we use ORBIT as the wireless network testbed to evaluate UbiFlow. ORBIT is composed of 400 radio nodes, where a number of experimental "sandboxes" can be accessed via its management framework. Available sandboxes include WiFi, WiMAX, USRP2, etc. ORBIT supports Floodlight [61] based OpenFlow controller to switch access between the WiFi and WiMAX interfaces, and uses Open vSwitch (OVS) [62] to allow a network interface to be OpenFlow-enabled.

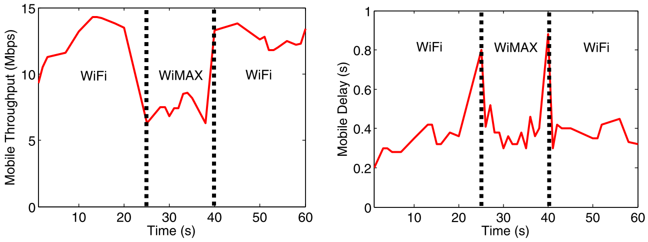


(a) End-to-End Throughput (b) End-to-End Delay

Fig. 12. Mobile flow scheduling in real testbed.

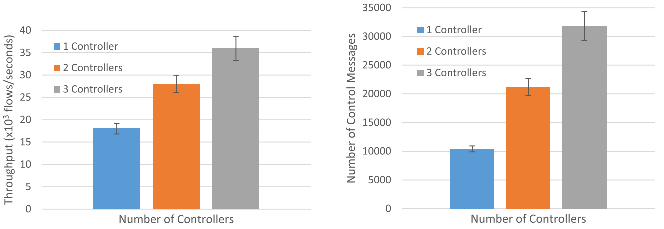
The deployment of ORBIT testbed follows a grid topology, and we choose an ORBIT sandbox with 1 WiMAX node and 7 WiFi nodes in our experiments. UDP is used as our transmission protocol. The delay is measured per packet and its performance is averaged using the fine grained network calculus model. We are aware that real mobile access pattern of IoT devices in urban scenario does not follow the random waypoint model. Actually, the urban-scale access of multi-networks is more like event or motivation driven behaviour. To better evaluate UbiFlow in this kind of mobile scenario, we collected a campus-wide mobile trace driven by class events, and use it in our evaluation. Specifically, the trace is collected during a period (10 minutes) between two consecutive classes around a lecture building by three types of IoT devices: smart phone, tablet and laptop. During that period, some students leave the building after previous class, some students come to the building for incoming class, and some students still stay in the building. Therefore, the wireless access of their IoT devices can be classified as "leaving", "joining", and "staying". We use the trace file to generate the mobile scenario in our ORBIT testbed, and verify the performance of IoT devices supporting two types of ORBIT access points: WiFi and WiMAX under the mobile scenario. That is, we match the 8 OpenFlow-enabled ORBIT nodes as corresponding access points in the building, and use two Floodlight based OpenFlow controller to scheduling different service requests from around 300 IoT devices during that period, according to the mobile trace file.

We compare UbiFlow with an OpenFlow-based handover scheme proposed by GENI [63] (namely GENI). The GENI handover [41] is a vanilla implementation of SDN in wireless environment, without ubiquitous flow scheduling and mobility management. As shown in Fig. 12, we select 30 flows from the hundreds of active IoT devices, where flows 1-5 are file sharing, flows 6-15 are audio, and flows 16-30 are video streaming. The performance shows the similar results as previous simulation results with various flow types. Generally, UbiFlow outperforms GENI handover both on end-to-end throughput and delay evaluation. For the 30 flows, UbiFlow can achieve an average throughput as 7.24 Mbps, while GENI only can provide 5.09 Mbps; UbiFlow improves the average throughput performance by



(a) Mobile Throughput (b) Mobile Delay

Fig. 13. Mobility management in real testbed.



(a) Throughput (b) Overhead

Fig. 14. Scalability in real testbed.

42.24 percent. The average delay in UbiFlow is around 0.11 s, while the delay in GENI is 0.29 s; UbiFlow reduces the average delay by 62.07 percent. In comparison with GENI, UbiFlow adopts dynamic flow scheduling scheme from the views of partition and device aspects, therefore can achieve better assignment of access points to satisify different flow requirements of IoT devices. In addition, the overlay structure based load balancing can effectively allocate flows in UbiFlow, by the coordination of controllers and switches. It also can help to improve the throughput and reduce the delay.

To test the mobility management of UbiFlow in real testbed, we choose one mobile device and evaluate the change of its multinetwork access in a period of one minute, while associating with different types of access points. The performance of throughput and delay of its access is shown in Figs. 13a and 13b respectively. As we can see, since there is only one WiMAX node in our testbed, and it is crowded by other mobile users, the throughput provided by WiMAX is much lower than WiFi nodes. According to this situation, the SDN controller only assigns the mobile device to access WiMAX when there is no available WiFi access points providing higher data rate. Once the controller finds a WiFi access point with better capacity and the mobile device sends flow request in its range, it will assign the mobile device to access the WiFi node. In mobile scenario, we notice that the average flow transmission delay for this mobile device is below 0.4s, which presents stable performance of our mobility management, considering there are hundreds of active IoT devices and only 8 working access points. Mobile delay only increases obviously when UbiFlow runs handover steps to assign new access point to the mobile device, which happens at the 25th second and 40th second of this period. Usually, when a mobile device requests an access point, it will initially send the request to the controller, and then controller sends the assignment decision back. This process results in the extra delay for message exchange and computation, which cannot be avoided if we use the controller to match access points with mobile devices. However, in these special cases, UbiFlow still can achieve a handover delay less than 0.9 seconds, therefore shows satisfactory results.

We also verify the scalability of UbiFlow in the ORBIT testbed, where we choose three ORBIT sandboxes (each has 1 WiMAX node and 7 WiFi) and assign each sandboxes with a controller. Similar to our scalability simulation in Section 5.2, Packet-In message is used to evaluate the scalability of flow scheduling in UbiFlow. On average, for each controller, we send 10000 consecutive Packet-In messages to reflect the flow requests generated by around 100 IoT devices in 10 minutes to it. We plot the throughput of UbiFlow with varying number of controllers, as shown in Fig. 14a. We observe that adding controllers increases the throughput almost linearly due to the fact that each controller in UbiFlow mainly controls the traffic flows in its own partition. In addition, if there is an imbalance in one controller, other controllers with light-weight traffic also can help to migrate the flows to their partitions by the UbiFlow overlay structure.

Note that the Packet-In messages in the scenario of Fig. 14a is a type of control message generated by the OpenFlow switches and sent to the SDN controller to request flow services for ubiquitous IoT devices. In our UbiFlow implementation, these messages are triggered by the flow request messages initiated from the IoT device. In addition, there are other overhead imposed by the distributed coordination of SDN controllers under different networking and traffic conditions. We present the total number of overhead generated by different types of control messages during mobility management and flow scheduling with varying number of controllers in Fig. 14b. In general, these control message based overhead can be classified as Packet-In messages and Packet-Out messages, respectively, for the SDN controller.

As for the interaction between the SDN controller and the IoT devices, the AP request messages and flow request messages generated by the IoT devices all can trigger Packet-In messages from the switch to the controller. From the controller aspect, the AP response messages and AP update message can trigger Packet-Out messages from the controller to the switch. As for the interaction between the controller and the switch, if new flow arrives or flow changes, the controller sends Packet-Out messages to the switch to update its flow table. The controller also sends Packet-Out messages to the switch to obtain traffic information for statistical analysis of network status and derive the partition view. As for the interaction between different controllers, the join or leave operation of a controller node on our overlay network makes its neighboring controllers to send Packet-Out message to maintain the network consistency. Also, the mobile handover of an IoT device across different geographical partitions triggers its current associated controller to localize its previous associate controller through its supervisory controller. This process results in Packet-Out message transmitted between the different types of controllers for information inquiry, mobility update and flow migration purposes. In addition, when controller level failure happens, our fault tolerance mechanism requires the failed controller to send Packet-Out message carrying data replica to its successor controllers.

Above interactions happened during mobility management and flow scheduling result in the overhead cost in

Fig. 14b. For one controller case, in addition to the 10000 consecutive Packet-In messages for flow requests from 100 IoT devices, extra overhead are caused by the control messages of interaction between the controller and the IoT devices, and the control messages of interaction between the controller and the switch. For two and three controllers, we observe that adding controllers increases the overhead almost linearly due to the fact that each controller mainly controls the traffic flows in its own partition, and on average there are 100 IoT devices in each partition generating 10000 flow request message in 10 minutes. Meanwhile, the control messages of interaction happened between different controllers generate additional overhead in comparison to the one controller case. The overhead cost of SDN controller is induced in the necessary interaction procedures, and these interactions present the dynamic capability of UbiFlow to re-adapt scheduling and effective flow bandwidth when varying the network conditions.

Overall, both simulation and real testbed results have shown that, in mobile environments, UbiFlow can adaptively match various traffic flows to wireless links; therefore can provide better service to satisfy the requirements of IoT devices and guarantee the partition performance at the same time.

6 Conclusion

In this paper, we have presented a software-defined IoT system, namely UbiFlow, for efficient flow control and mobility management in urban heterogeneous networks. In addition to flow scheduling, the approach shifts mobility management, handover optimization, and access point selection functions from the relatively resource constrained IoT devices to more capable distributed controllers. The distributed controllers are organized in a scalable and fault tolerant manner. The system was evaluated through simulation and on a testbed.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments, and sincerely acknowledge Mr. Ivan Seskar from Rutgers University and Mr. Ryan Izard from Clemson University for their help on setting up the Orbit testbed. This work was partially supported by the National Natural Science Foundation of China under Grant No. 61602168 and No. 61972145, the Intel Collaborative Research Institute for Sustainable Connected Cities (ICRI Cities), the University of California Center on Economic Competitiveness in Transportation (UCCONNECT), the Hu-Xiang Youth Talent Program under Grant No. 2018RS3040, and the research project OrganiCity under Grant No. 645198 of the European Unions Horizon 2020 research and innovation program.

References

[1] V. Gutiérrez *et al.*, “Co-creating the cities of the future,” *Sensors*, vol. 16, no. 11, 2016, Art. no. 1971.

[2] D. Wu *et al.*, “ADDSEN: Adaptive data processing and dissemination for drone swarms in urban sensing,” *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 183–198, Feb. 2017.

[3] Z. Qin, D. Wu, Z. Xiao, B. Fu, and Z. Qin, “Modeling and analysis of data aggregation from convergecast in mobile sensor networks for industrial IoT,” *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4457–4467, Oct. 2018.

[4] D. Wu, Q. Liu, Y. Zhang, J. A. McCann, A. C. Regan, and N. Venkatasubramanian, “CrowdWiFi: Efficient crowdsensing of roadside WiFi networks,” in *Proc. ACM/IFIP/USENIX Middleware Conf.*, 2014, pp. 229–240.

[5] Z. Qin, L. Iannario, C. Giannelli, P. Bellavista, G. Denker, and N. Venkatasubramanian, “Mina: A reflective middleware for managing dynamic multinetwork environment,” in *Proc. IEEE Netw. Operations Manage. Symp.*, 2014, pp. 1–9.

[6] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, Jul.-Sep. 2017.

[7] D. Wu, D. I. Arkhipov, T. Przepiorka, Y. Li, B. Guo, and Q. Liu, “From intermittent to ubiquitous: Enhancing mobile access to online social networks with opportunistic optimization,” *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 114:1–114:32, 2017.

[8] F. Xu *et al.*, “Big data driven mobile traffic understanding and forecasting: A time series approach,” *IEEE Trans. Services Comput.*, vol. 9, no. 5, pp. 796–805, Sep./Oct. 2016.

[9] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, “UbiFlow: Mobility management in urban-scale software defined IoT,” in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 208–216.

[10] D. G. Zhang, G. Li, K. Zheng, X. C. Ming, and Z. H. Pan, “An energy-balanced routing method based on forward-aware factor for wireless sensor networks,” *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 766–773, Feb. 2014.

[11] N. McKeown *et al.*, “OpenFlow: Enabling innovation in campus networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 2, pp. 69–74, 2008.

[12] N. Gude *et al.*, “NOX: Towards an operating system for networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[13] Cisco, “Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021,” 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html

[14] D. Wu, L. Bao, A. C. Regan, and C. L. Talcott, “Large-scale access scheduling in wireless mesh networks using social centrality,” *J. Parallel Distrib. Comput.*, vol. 73, no. 8, pp. 1049–1065, 2013.

[15] D. Wu, Q. Liu, Y. Li, J. A. McCann, A. C. Regan, and N. Venkatasubramanian, “Adaptive lookup of open WiFi using crowdsensing,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3634–3647, Dec. 2016.

[16] D. Wu, D. I. Arkhipov, Y. Zhang, C. H. Liu, and A. C. Regan, “Online war-driving by compressive sensing,” *IEEE Trans. Mobile Comput.*, vol. 14, no. 11, pp. 2349–2362, Nov. 2015.

[17] F. Shi, Z. Qin, D. Wu, and J. A. McCann, “Effective truth discovery and fair reward distribution for mobile crowdsensing,” *Pervasive Mobile Comput.*, vol. 51, pp. 88–103, 2018.

[18] D. G. Zhang, H. Ge, T. Zhang, Y. Y. Cui, X. H. Liu, and G. Q. Mao, “New multi-hop clustering algorithm for vehicular ad hoc networks,” *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 4, pp. 1517–1530, Apr. 2019.

[19] D. Wu, D. I. Arkhipov, T. Przepiorka, Q. Liu, J. A. McCann, and A. C. Regan, “DeepOpp: Context-aware mobile access to social media content on underground metro systems,” in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1219–1229.

[20] K. Liu, D. Wu, and X. Li, “Enhancing smartphone indoor localization via opportunistic sensing,” in *Proc. IEEE Int. Conf. Sensing Commun. Netw.*, 2016, pp. 1–9.

[21] A. A. Khan, M. H. Rehmani, and A. Rachedi, “Cognitive-radio-based internet of things: Applications, architectures, spectrum related functionalities, and future research directions,” *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 17–25, Jun. 2017.

[22] D. Wu, Y. Zhang, J. Luo, and R. F. Li, “Efficient data dissemination by crowdsensing in vehicular networks,” in *Proc. IEEE/ACM Int. Symp. Quality Service*, 2014, pp. 314–319.

[23] J. Q. Chen, G. Q. Mao, C. L. Li, W. F. Liang, and D. G. Zhang, “Capacity of cooperative vehicular networks with infrastructure support: Multi-user case,” *IEEE Trans. Veh. Technol.*, vol. 67, pp. 1546–1560, Feb. 2018.

[24] D. Wu, L. Bao, and R. Li, “Robust localization protocols and algorithms in wireless sensor networks using UWB,” *Ad Hoc Sensor Wireless Netw.*, vol. 11, no. 3–4, pp. 219–243, 2011.

[25] M. Al-Fares, S. Radhakrishnanl, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in *Proc. 7th USENIX Conf. Netw. Syst. Design Implementation*, 2010, pp. 281–296.

[26] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, 2011, pp. 254–265.

[27] D. Li, J. Zhu, J. Wu, J. Guan, and Y. Zhang, "Guaranteeing heterogeneous bandwidth demand in multi-tenant data center networks," *IEEE/ACM Trans. Network.*, vol. vol. 23, no. 5, pp. 1648–1660, Oct. 2015.

[28] Huawei Agile IoT Solution, Accessed: 2018. [Online]. Available: http://e.huawei.com/en/news/global/2015/201505301129

[29] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, pp. 1–6.

[30] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Operating Syst. Design Implementation*, 2010, pp. 351–364.

[31] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed SDN controller," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst.*, 2014, pp. 17–28.

[32] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 121–126.

[33] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[34] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A global name service for a highly mobile internetwork," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 247–258, 2014.

[35] D. Wu, L. Bao, and C. H. Liu, "Scalable channel allocation and access scheduling for Wireless internet-of-things," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3596–3604, Oct. 2013.

[36] M. Hegde *et al.*, "Experiences with a centralized scheduling approach for performance management of IEEE 802.11 Wireless LANs," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 648–662, Apr. 2013.

[37] X. Wang, X. Lin, Q. Wang, and W. Luan, "Mobility increases the connectivity of Wireless networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 440–454, Apr. 2013.

[38] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: A programmable wireless dataplane," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 109–114.

[39] K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, 2010.

[40] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2014, pp. 1–9.

[41] R. Izard, A. Hodges, J. Liu, J. Martin, K. Wang, and K. Xu, "An OpenFlowTestbed for the evaluation of vertical handover decision algorithms in heterogeneous wireless networks," in *Proc. 9th Int. Conf. Testbeds Res. Infrastructures Develop. Netw. Communities*, 2014, pp. 174–183.

[42] S. Wang, Y. Cui, S. K. Das, W. Li, and J. Wu, "Mobility in IPv6: Whether and how to hierarchize the network?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1722–1729, Oct. 2011.

[43] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and flexible cellular core network architecture," in *Proc. 9th ACM Conf. Emerging Netw. Experiments Technol.*, 2013, pp. 163–174.

[44] M. Moradi, W. Wu, L. E. Li, and Z. M. Mao, "SoftMoW: Recursive and reconfigurable cellular WAN architecture," in *Proc. 10th ACM Int. Conf. Emerging Netw. Experiments Technol.*, 2014, pp. 377–390.

[45] X. Qiu, H. Luo, G. Xu, R. Zhong, and G. Q. Huang, "Physical assets and service sharing for IoT-enabled supply Hub in industrial park (SHIP)," *Int. J. Production Economics*, vol. 159, pp. 4–15, 2015.

[46] Y. Zhao, Y. Li, D. Wu, and N. Ge, "Overlapping coalition formation game for resource allocation in network coding aided D2D communications," *IEEE Trans. Mobile Comput.*, vol. 16, no. 12, pp. 3459–3472, Dec. 2017.

[47] F. Shi, Z. Qin, D. Wu, and J. A. McCann, "MPCSToken: Smart contract enabled fault-tolerant incentivisation for mobile P2P crowd services," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 961–971.

[48] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, Aug. 2001, pp. 149–160.

[49] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proc. ACM Symp. Theory Comput.*, 1997, pp. 654–663.

[50] S. A. Abid, M. Othman, and N. Shah, "A survey on DHT-based routing for large scale mobile ad hoc networks," *ACM Comput. Surveys*, vol. 47, no. 2, pp. 20:1–20:46, 2015.

[51] National Institute of Standard and Technology (NIST), "Secure hash standard," Federal Information Processing Standards Publication (FIPS 180-1), 1995.

[52] D. Namiot and M. Sneps-Sneppe, "Geofence and network proximity," in *Internet of Things, Smart Spaces, and Next Generation Networking*. Berlin, Germany: Springer, 2013, pp. 117–127.

[53] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, "Locality-preserving hashing in multidimensional spaces," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 618–625.

[54] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Germany: Springer, 2001.

[55] Video streaming trace files, Accessed: 2018. [Online]. Available: http://trace.eas.asu.edu/TRACE/ltvt.html

[56] Skype tele audio trace files, Accessed: 2018. [Online]. Available: http://tstat.polito.it/traces-skype.shtml

[57] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.

[58] OMNeT++, Accessed: 2018. [Online]. Available: http://www.omnetpp.org

[59] ORBIT, Accessed: 2018. [Online]. Available: https://www.orbit-lab.org

[60] An OpenFlow Extension for the OMNeT++ INET Framework, Accessed: 2018. [Online]. Available: http://www3.informatik.uni-wuerzburg.de/research/ngn/ofomnet/of_omnet.shtml

[61] Project Floodlight, Accessed: 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight

[62] Open vSwitch, Accessed: 2018. [Online]. Available: http://openvswitch.org

[63] GENI, Accessed: 2018. [Online]. Available: http://www.geni.net

Di Wu received the MS and PhD degrees in computer science from the University of California, Irvine. He was a researcher at the Intel Collaborative Research Institute for Sustainable Connected Cites, a research associate at Imperial College London, a staff research associate at the University of California, Irvine, a visiting researcher at IBM research, and a student research associate at the SRI International. He is currently an associate professor with the Department of Computer Engineering, Hunan University, China, and an adjunct researcher with the University of California Transportation Center. His research interests include wireless networks and mobile computing, Internet-of-things and cyber-physical systems, smart cities and big data. He has actively served on many conference committees and is currently associate editor for the *IEEE Transactions on Intelligent Transportation Systems*. He is a member of the IEEE and the ACM.

Xiang Nie received the BS degree in network engineering from the Wuhan Institute of Technology, in 2011. He is currently working toward the MS degree in computer technology at Hunan University, China. His research interests include software defined networking, distributed systems, and Internet of Things.

Eskindir Asmare received the PhD degrees in computer science from Imperial College London, United Kingdom, in 2011. He was a research fellow with the school of informatics at the University of Sussex, United Kingdom. He is currently a research associate with the Intel Collaborative Research Institute for Sustainable Cities at Imperial College London. His research interests include autonomic management of distributed systems, mobile systems and pervasive computing. He is a member of the IEEE.

Dmitri I. Arkhipov received the BS degree in information and computer science, the MS degree in computer science and the PhD degree from the University of California, Irvine, in 2009, 2012, and 2016 respectively. He is currently a postdoctoral researcher with the Department of Computer Science, University of California, Irvine. His research interests include parallel and distributed systems, large scale combinatorial optimization, and cyber-physical systems.

Zhijing Qin received the BS degree in software engineering from Beihang University, in 2007, the MS degree in software engineering from Peking University, in 2009, and the PhD degree in networked systems from the University of California, Irvine, in 2015. He is currently a software engineer at Google Inc. His research interests include multinetwork management, information collection, formal method based network analysis and cyber physical system.

Renfa Li is a professor with the College of Computer Science and Electronic Engineering, Hunan University, China, and the director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of National Supercomputing Center in Changsha, China. His major research includes embedded systems, distributed systems, and cyber-physical systems. He is a senior member of the IEEE and the ACM.

Julie A. McCann is a professor in computer systems at Imperial College London. Her research centers on highly decentralized and self-organizing scalable algorithms for spatial computing systems e.g., wireless sensing networks. She leads both the Adaptive Embedded Systems Engineering Research Group and the Intel Collaborative Research Institute for Sustainable Cities, and is currently working with NEC and others on substantive smart city projects. She has received significant funding through bodies such as the UK's EPSRC, TSB and NERC as well as various international funds, and is an elected peer for the EPSRC. She has actively served on, and chaired, many conference committees and is currently associate editor for the *ACM Transactions on Autonomous and Adaptive Systems*. She is a member of the IEEE and the ACM as well as a chartered engineer, and was elected as a fellow of the BCS in 2013.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 600 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is an IEEE fellow.