Follow Me Fog: Toward Seamless Handover Timing Schemes in a Fog Computing Environment

Wei Bao, Dong Yuan, Zhengjie Yang, Shen Wang, Wei Li, Bing Bing Zhou, and Albert Y. Zomaya

The authors propose Follow Me Fog (FMF), a framework supporting a new seamless handover timing scheme among different computation access points. Intrinsically, FMF supports a job pre-migration mechanism, which pre-migrates computation jobs when the handover is expected to happen. Such expectations can be indicated by constantly monitoring received signal strengths.

ABSTRACT

Equipped with easy-to-access micro computation access points, the fog computing architecture provides low-latency and ubiquitously available computation offloading services to many simple and cheap Internet of Things devices with limited computing and energy resources. One obstacle, however, is how to seamlessly hand over *mobile* IoT devices among different computation access points when computation offloading is in action so that the offloading service is not interrupted — especially for time-sensitive applications. In this article, we propose Follow Me Fog (FMF), a framework supporting a new seamless handover timing scheme among different computation access points. Intrinsically, FMF supports a job pre-migration mechanism, which pre-migrates computation jobs when the handover is expected to happen. Such expectations can be indicated by constantly monitoring received signal strengths. Then we present the design and a prototype implementation of FMF. Our evaluation results demonstrate that FMF can achieve a substantial latency reduction (36.5 percent in our experiment). In conclusion, the FMF design clears a core obstacle, allowing fog computing to provide interruption-resistant services to mobile IoT devices.

INTRODUCTION

The rapid evolution of the Internet of Things (IoT) will soon have a substantial impact on our daily lives through reforming many Internet applications in a variety of domains. IoT devices, however, create a growing need to store and process significant quantities of data, but they themselves have limited computing and energy resources, and hence are not able to perform sophisticated data processing and storage. Therefore, the IoT is perfectly aligned with emerging fog computing: Equipped with easy-to-access micro computation access points (i.e., fog nodes), it provides low-latency and ubiquitously available computation services [1, 2].

Mobility is an intrinsic trait of many IoT applications in a fog computing environment [3], such as smart transportation, smart sport, smart tourism, and geo-based games. Consider an exemplary application, an augmented reality (AR) assisted tour guide: a mobile tourist holds an IoT device that continuously records its current view and streams the multimedia contents to a nearby fog node, while the fog node performs scene recognition and sends back contextual augmentation labels, to be displayed on the IoT device overlaying the actual scene. Many mobile applications like this require not only powerful computing resources but also sufficiently low latency in data transfer and processing to guarantee users' desired quality of experience. However, we observe that IoT mobility inevitably poses significant challenges for realizing reliable and punctual computing services. Since each computation access point provides only limited wireless coverage, movement of IoT devices will call for handovers among different computation access points, causing complicated internetworking issues and interrupted services.

In conventional mobile networks, mobility of a mobile terminal is enabled by horizontal and vertical handover procedures when it changes serving access point (or base station). However, such handover mechanisms are insufficient to support reliable and punctual computation offloading in the environment of fog computing, where the access point additionally processes jobs uploaded from users. When a handover occurs, those pending jobs (e.g., jobs unprocessed, being processed, or processed but not fed back) must be recovered after the handover. Meanwhile, the new access point should allocate new computing resources to resume job processing. Mobile devices, on the other hand, should also be prepared to recover from potential losses and delays.

There is a set of pioneering studies, based on theoretical analyses, simulations, and testbed experiments, to address the mobility issues in a computation offloading environment, such as "whether to conduct handovers" and "where to hand over." However, one critical question remains unanswered: what is the exact timing procedure of a handover to minimize service interruptions? The answer is vital to many delay-sensitive applications. In the aforementioned AR guided tour scenario, the overall latency, from the beginning of uploading the current scene to the end of successfully downloading augmentation labels, must be reasonably

Digital Object Identifier:
10.1109/MCOM.2017.1700363

The authors are with the University of Sydney.

0163-6804/17/$25.00 © 2017 IEEE

small, and this latency requirement should not be impacted by handovers — our ultimate goal is to make the tourist unaware of the existence of handovers. During a handover, important procedures such as disconnection from the old fog node, connection to the new fog node, and job recovery at the new fog node must be conducted in an orderly and timely manner so that the additional latency caused by handover is minimized.

We see a clear gap between the requirements for a seamless handover and the lack of an efficient handover timing scheme in a fog computing environment. Therefore, we are motivated to propose Follow Me Fog (FMF), a new framework supporting seamless handover timing schemes. The objective of FMF is to orchestrate a sequence of procedures at both the mobile IoT devices and fog nodes, to guarantee service continuity and reduce latency during handovers.

The key design feature of FMF is motivated by an important observation on handovers in the fog computing environment. There are two major processes involved in a handover:

1. *Connection redirection*
2. *Service recovery*

In the connection redirection process, the mobile IoT device disconnects from its original fog node and then connects to a new node. In the service recovery process, the pending jobs offloaded to the old fog node will be recovered at the new node. One can consider a straightforward solution that conducts service recovery after connection redirection: As soon as the mobile device reaches the new fog node, it suggests that there are pending jobs in the old fog node. However, such a handover procedure may cause severe service interruptions, since the computation offloading service is completely halted during both connection redirection and service recovery phases, as shown in Fig. 2a.

Fortunately, we can observe that the service recovery can actually be conducted *in advance* to substantially reduce service interruptions. Our proposed FMF *pre-migrates* computation jobs when the handover is about to happen. Such expectation of handover can be indicated by constantly monitoring the received signal strengths (RSSs) from different fog nodes. When the RSS from the current fog node keeps decreasing while the RSS from a neighboring one keeps increasing, the pre-migration of computation jobs is triggered prior to the connection redirection. As a consequence, the service can be resumed in a timely way when the mobile device is redirected to the new fog node, as shown in Fig. 2b. It is worth noting that FMF follows a cross-layer design: both the RSS levels in the physical/medium access control (MAC) layer and the status of computation jobs in the application layer will influence the decision on the timing of connection redirection and job (pre-) migration.

In what follows, we present the background and compare the differences between FMF and the existing frameworks. We present the FMF design. We present a prototype implementation of FMF and show evaluation results of the implementation. Finally, we conclude our article and discuss some future directions.

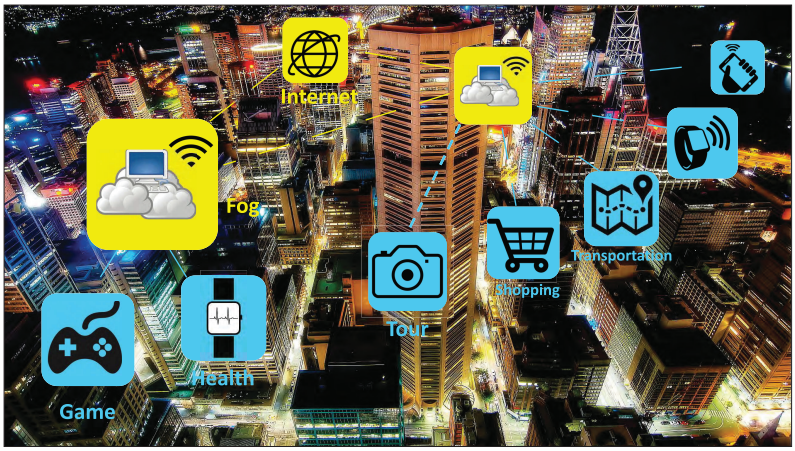

Figure 1. Fog computing environment for IoT applications.

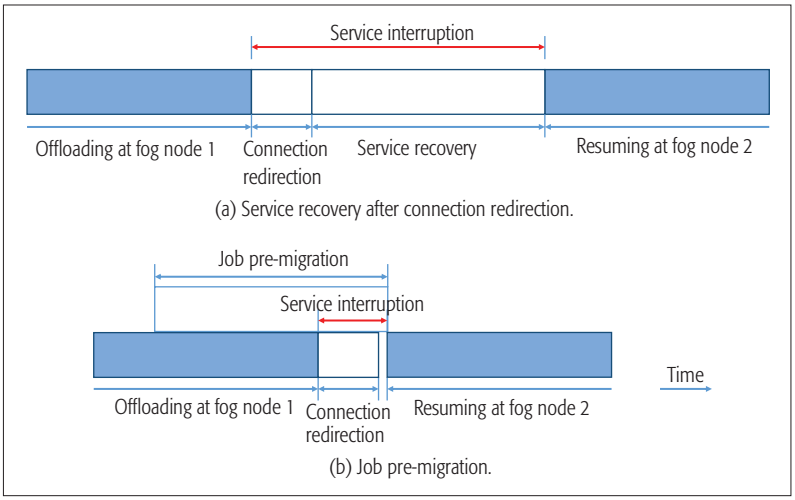

Figure 2. Potential delay reduction of pre-migration.

Background and Related Work

Mobility Management in Wireless Networks

Handover timing algorithms have been intensively investigated in traditional mobile networks. These algorithms aim to transfer mobile terminals to the "best" access point without perceivable interruptions in data transfer. Ping-pong effects should also be avoided. One type of algorithms employ threshold comparison of one or several metrics (e.g., RSS, bandwidth, etc.) [4]. A second type is based on mobility load balancing, where cells suffering congestion can transfer load to other cells [5]. A third type uses dynamic programming or artificial intelligence techniques to improve the effectiveness of handoff procedures (e.g., [6]). However, these algorithms are insufficient in the presence of delay-sensitive computation offloading, since they do not handle promptly and efficiently migrating computation jobs among fog nodes when a handover occurs.

Live Migration

Live migration was first proposed and used in cloud data centers to move running applications between different physical machines without disconnecting the users. It has also been used to hand over computation jobs between fog nodes. Virtual machine (VM) migration [7] and container migration [8] (e.g., Docker [9]) are the two

No existing framework for live migration takes into account the time sensitivity of the applications when they are handed over between fog nodes since delays caused by wireless disconnection and reconnection are out of the scope of live migration.

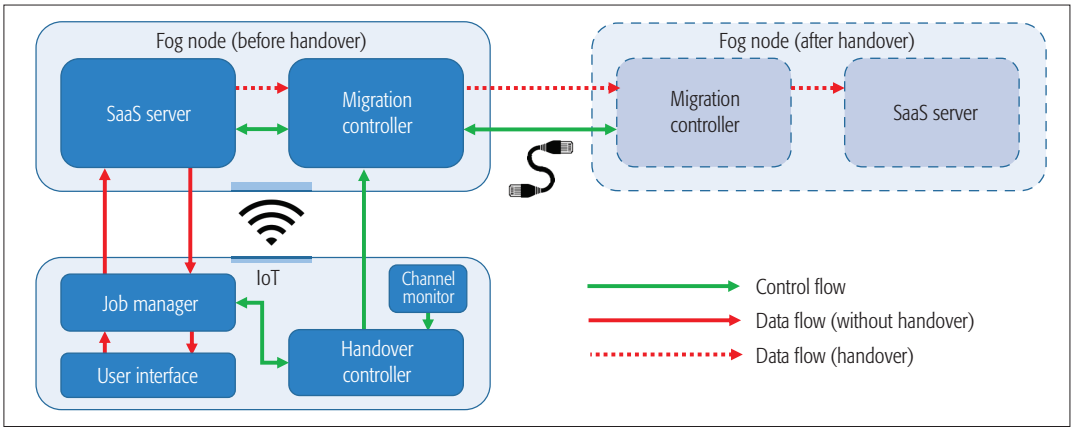

Figure 3. FMF module design.

main solutions. VM migration directly moves the full image of the computing node, including the OS, file system, and all software installed, to the destination node. It is a flexible solution and only requires the destination node to have enough resources to accommodate the VM. Container migration is a lighter-weight solution that can move a single application between computing nodes. It has lower migration cost but requires the container platform to be pre-installed in computing nodes. However, no existing framework for live migration takes into account the time sensitivity of the applications when they are handed over between fog nodes since delays caused by wireless disconnection and reconnection are out of the scope of live migration.

Path Selection

Live migration may not be a convenient solution when a large amount of data are to be migrated among computing nodes — it may introduce large delays and burden backhaul links. An alternative approach is path selection [1], where a more suitable path is found for delivery of computed data from a data center to a mobile terminal. By combining the mechanisms of live migration and path selection, path selection is employed if a reasonably good path can be found, while live migration is performed if even the best available path is not satisfactory. However, like live migration, delays caused by wireless disconnection and reconnection are not considered.

Follow Me Cloud

The concept of Follow Me Cloud (FMC) is to support smooth migration of IP services between a data center and mobile device to another data center with no service disruption [10]. FMC is a service migration mechanism (among data centers) in the wired network, while the wireless handover timing procedure is not redesigned (i.e., it follows a standard procedure in traditional cellular networks). As discussed earlier, in order to promptly hand over delay-sensitive applications, it is more desirable to jointly design the connection redirection and job migration, which is the emphasis of FMF studied in this article.

Analytical Models

There is another category of studies using analytical models to address computation migration problems; examples include the random walk model [11], a Markov decision process [12], combinatorial cost minimization [13], and integer programming [14]. In the scope of the handover timing scheme, decisions are made within small timescales. Analytical models and real-world experiments that can capture variations and make decisions within small timescales are much more desirable.

FMF Design Outline

The Problem

The overall problem is to design a seamless handover timing procedure to avoid service interruptions during handovers in the scenarios where a moving IoT device offloads its delay-sensitive jobs to fog nodes it passes. In this article, we address this issue by developing FMF, a new paradigm for a smooth handover mechanism, and then implement a prototype system.

A Module Design of FMF

The module framework of FMF is shown in Fig. 3 with two parties. IoT devices and fog nodes are connected via wireless links, and the fog nodes are interconnected via a wired network.

The Fog Node: In the framework of FMF, we consider the scenario where each fog node provides software as a service (SaaS) to mobile IoT devices in its *SaaS server*. The SaaS model is suitable for many special-purpose time-sensitive IoT applications (e.g., smart sport and smart tourism) conducted within a given geographical region, and the job processing service for that application is managed by one provider (e.g., administrator of a stadium or museum). The software is deployed in each fog node as a service, which is in the "always-on" status to process offloaded jobs from IoT devices in a timely way and return outcomes. In order to conduct smooth handovers, the *migration controller* is responsible for migrating the jobs to the new fog node to which the IoT device is handed over. Pre-migration is applied so that the migration is triggered prior to connection redirection.

In this article, we focus on the scenarios where an SaaS server is installed at each access point. The designed FMF is also applicable in scenarios where one SaaS server provides computing services to multiple access points. When a mobile IoT device is handed over between two access

points served by two different SaaS servers, both job migration and connection redirection occur, requiring our proposed seamless handover timing scheme.

The IoT Device: At the IoT device, the *job manager* uploads unprocessed jobs and downloads processed jobs (outcomes); the *user interface* interacts with human users; and the *channel monitor* keeps monitoring the channel states from different fog nodes. The key module at the IoT device is the *handover controller*, which is responsible for:

1. Checking the channel states (e.g., RSSs) from different fog nodes and learning their changing tendencies
2. Checking the status of offloaded jobs (e.g., which job is "in flight")
3. Making decisions on job migration and connection redirection
4. Interacting with the migration controllers at fog nodes to advise job pre-migration prior to connection redirection and job recovery after connection redirection

The handover timing scheme is jointly realized by the handover controller at the IoT device and the migration controller at the fog node. The handover controller analyzes the current situation of the IoT device and makes decisions on job pre-migration and connection redirection. When a pre-migration decision is made, the handover controller notifies the migration controller at the fog node, so jobs are migrated to the new fog node. When the IoT device connects to the new fog node, the handover controller contacts the new migration controller to recover the jobs offloaded to the old fog node. The handover timing scheme is further specified as a finite state machine in the following subsection.

Handover Timing Scheme: A Finite State Machine Design

In this subsection, we present the design guideline of the handover timing scheme to be conducted in the FMF framework. The designed handover timing scheme follows a finite state machine, as shown in Fig. 4. There are four states: the connection state, pre-migration state, redirection state, and resuming state. We specify the four states first and then introduce the transitions between the four states.

The States: In the *connection state*, the IoT device continuously offloads computation jobs to its connected fog node and downloads the outcomes. The job offloading is performed in a pipeline fashion on the IoT side: the next job could be uploaded without waiting for the outcome of its previous job, but the maximum number of uploaded but pending jobs is limited by a "window." On the fog side, the jobs are processed in a first-come first-served way. The outcomes of the jobs are sent back to the IoT as soon as processing is completed.

In the *pre-migration state*, an upcoming handover is expected, so pre-migration is conducted. However, job uploading and downloading may continue in this state. On the fog side, the current fog node establishes a connection to the expected fog node and starts to transfer the processed jobs. If a job is not processed, it will be processed first and then migrated. In our current design version, the fog will only migrate the processed jobs (outcomes) rather than unprocessed jobs or jobs being processed. Due to the potential of a "false alarm" in an upcoming handover, it is possible that the IoT will finally cancel the handover even if it enters the pre-migration state. Whenever a "false alarm" happens, migrating unprocessed jobs causes unnecessary job processing at two fog nodes, and migrating jobs being processed causes substantial communication overhead. Please note that "false alarms" are inevitable due to the unpredictability of user mobility and instability of wireless channels. Please further refer to our discussions on "transition from pre-migration to connection state" below.

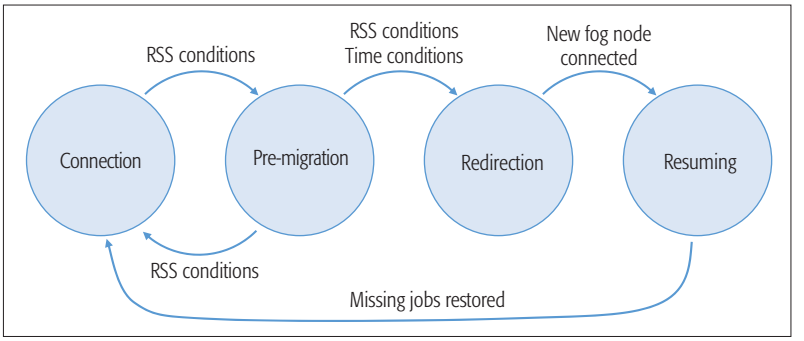

Figure 4. Finite state machine of the handover timing scheme.

In the *redirection state*, the IoT device disconnects from the old fog node and establishes connection to the new one. The old fog node keeps processing and migrating the jobs offloaded prior to reconnection if they are not completely processed and migrated.

In the *resuming state*, the IoT device requests missing jobs (sent to the old fog node before connection redirection) to the new fog node and the new fog node will return the processed jobs. The IoT device is also allowed to send new jobs to the new fog node. The old fog node keeps processing and migrating the jobs prior to connection redirection if they are not completely processed and migrated.

State Transitions: *From connection to pre-migration state:* In the connection state, the IoT device keeps monitoring the RSSs from its current fog node and neighboring ones. The expectation of handover can be predicted by an RSS-based criterion (e.g., relative RSS with hysteresis [4], adaptive lifetime-based [15]). If the criterion is satisfied, the transition from connection to pre-migration state is triggered.

From pre-migration to connection state: During the pre-migration state, the IoT device may notice that the transition to pre-migration state is a false alarm. This can be implied by another RSS based criterion (e.g., the RSS from the current fog node becomes large again). The state is reversed to connection state if such a criterion is satisfied. The pre-migration from the current fog node to the (previously) expected fog node is cancelled.

From pre-migration to redirection state: The transition is triggered if the RSS condition from the new fog node keeps outperforming that from the current fog node for a while. The trigger of the state transition could be reasonably postponed for a small time period if there is a job being transferred (e.g., an unprocessed job being

The handover timing scheme is jointly realized by the handover controller at the IoT device and the migration controller at the fog node. The handover controller analyzes the current situation of the IoT device and makes decisions on job pre-migration and connection redirection.

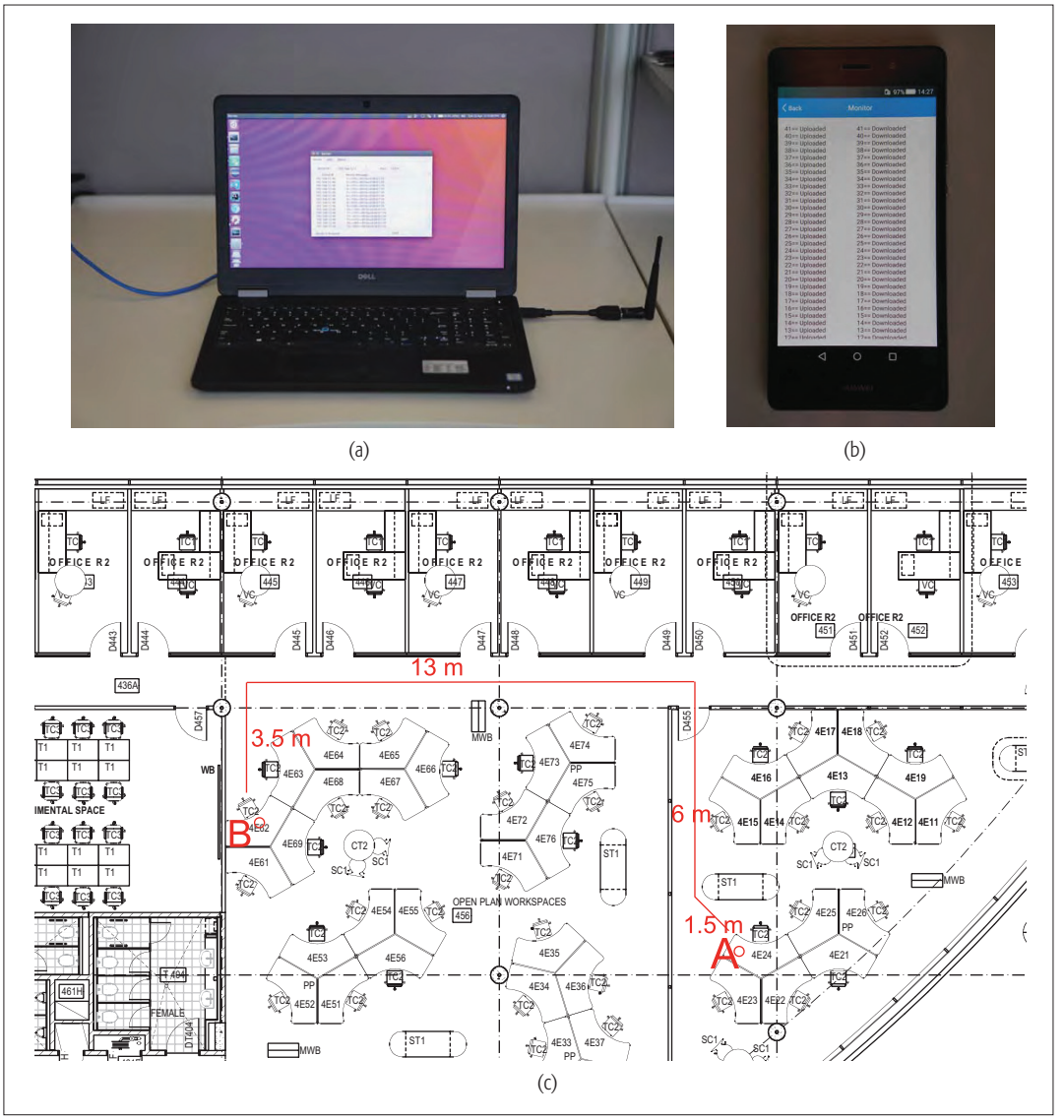

Figure 5. Implementation setup: a) fog node; b) mobile device; c) experiment environment.

uploaded and/or a processed job being downloaded), to wait for the completion of that job transfer.

From redirection to resuming state: The transition happens as soon as the IoT device is connected to the new fog node.

From resuming to connection state: The transition happens as soon as the outcomes of the jobs previously offloaded to the old fog node are successfully sent back to the IoT device via the new fog node.

Prototype and Performance Evaluation

In this section, we present a prototype of FMF that supports job pre-migration. We start with the implementation description of the prototype and then discuss the experiment setup. Finally, we present the evaluation results of the performance.

Implementation Description

To evaluate the proposed FMF framework, we develop an FMF prototype in a real fog computing environment. To create the fog computing environment, we use two laptop computers (Dell Latitude E5570, Core i7-6600U CPU and 8 GB memory) to act as fog nodes. Each fog node has two network adapters:

1. External 300 Mb/s wireless USB adapter (EDiMAX EW-7612UAn V2) that provides local WiFi coverage for the wireless IoT devices
2. Internal Ethernet adapter (Intel I219-LM) that provides connections to other fog nodes, as shown in Fig. 5a

We use smartphones (HUAWEI ALE-L02 P8 Lite) to act as mobile IoT devices and offload jobs to the fog nodes, as shown in Fig. 5b. The IoT devices and fog nodes are connected via WiFi interfaces, and the fog nodes are interconnected via Ethernet interfaces. The FMF application is implemented in both fog nodes and IoT devices. In each fog node, we install the Ubuntu 16.04.2 LTS 64-bit operating system and adopt the QT application framework to implement the application server and migration controller, and deploy them in the SaaS manner. For each IoT device, we adopt the Android platform (Android 5.0.1, API 21) to implement the job manager, user interface, channel monitor, and handover

controller in an all-in-one Android application called "FMF Client." All the communications in the prototype are implemented via sockets programming based on TCP. Between IoT devices and the fog node, three sockets are created with different port numbers for uploading jobs, downloading outcomes, and sending pre-migration signals, respectively. Between every two fog nodes, sockets are created to migrate computation jobs and exchange control signals with each other. WiFi RSS monitoring, connection, and reconnection are realized by Android API class `wifiManager`.

Experiment Setup

Experiment Field: As shown in Fig. 5c, we implement our prototype on the fourth floor of Building J12 in the University of Sydney. We install the two fog nodes at A and B, shown in the figure. In each round of the experiment, the Android phone is held to move from A to B along the red lines, with a total distance of 24 m. The speed of the mobile device is 0.63 m/s. In the experiment, the phone keeps offloading the same job (same communication load and processing load) to its connected fog node. The final performance is averaged on the results of 10 rounds of experiment.

State Transition Conditions: In the prototype implementation, the transition from connection to pre-migration state is triggered if the RSS from the new fog node is at least 10 dB greater than that from the current fog node for 1 s. The transition from pre-migration to connection state is triggered if the RSS from the current fog node is at least 10 dB greater than that from the new fog node for 0.5 s. The transition from pre-migration to redirection state is triggered if pre-migration state lasts for 1 s without transiting back to connection state.

Please note that the system performance can be further improved by carefully designing the transition conditions. However, that may involve complicated theoretical analysis and optimization. It is out of the scope of this article's focus on the framework design of FMF.

Performance Metrics and Benchmark Approach: We evaluate the delay performances of three categories of offloaded jobs. The first category of offloaded jobs do not experience handovers. They are offloaded to one fog node, and their outcomes are sent back from that fog node. The second and third categories of offloaded jobs experience handovers. For the second category, a benchmark re-uploading approach, instead of FMF, is employed: whenever a handover occurs, the mobile device gives up the pending jobs uploaded to the previous fog node and re-uploads these jobs to the new fog node. For the third category, FMF is employed.[1]

Performance Evaluation

The delay performance of the three categories of offloaded jobs are shown in Fig. 6. Each total delay (i.e., from the beginning of uploading one job to the end of successfully downloading its outcome) is further separated into uploading delay, processing delay, downloading delay, and "additional delay." Uploading, processing, and downloading delays correspond to the time durations to upload, process, and download an offloaded job *once*. They are unavoidable for any offloaded job. Additional delay corresponds to total delay minus uploading, processing, and downloading delays. Please note that if a handover happens, jobs in the second category may be re-uploaded or re-processed, and such delays contribute to additional delay; job migration of the third category may also contribute to additional delay.

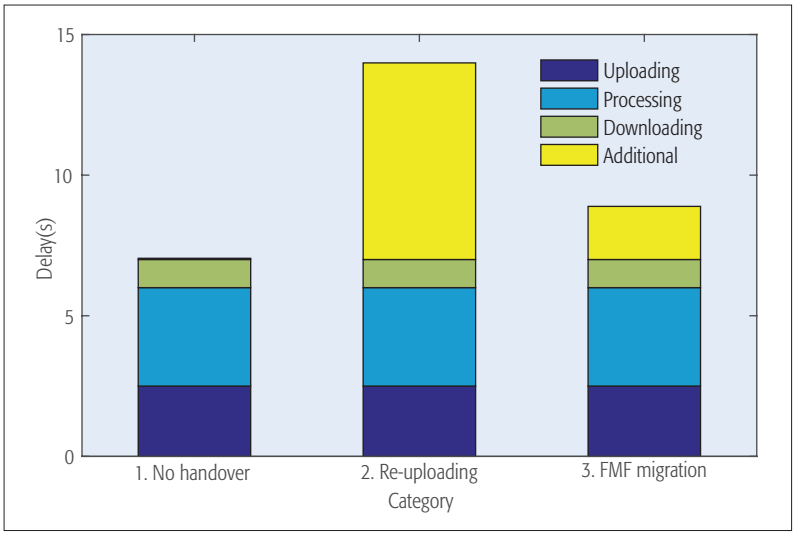

Figure 6. Delay performance by experiment.

From the figure, we notice that for the first category of jobs, the additional delay is quite small. However, for the second category of jobs (FMF is not employed), additional delay is substantially increased since the jobs uploaded to the old fog node are completely wasted, leading to large re-uploading and re-processing delays. For the third category of jobs, since FMF pre-migration is employed, additional delay can be largely avoided. In our experiment, it is 5.1 s smaller than that of the second category, but only 1.9 s greater than that of the first category. When FMF is employed, the additional delay (1.9 s) caused by handover is small compared to unavoidable uploading, processing, and downloading delays (7.0 s in total). The overall delay is reduced by 36.5 percent compared to that of the second category.

Consecutive Handovers

The designed FMF is able to handle the scenario where the IoT device makes two (or more) consecutive handovers within a short period of time (i.e., from fog node 1 to 2 and then from 2 to 3). If there are still missing jobs uploaded to fog node 1 when the second handover occurs, these missing jobs must be recovered at fog node 3. Fog node 1 will send the pending jobs to fog node 2 as usual. Then the jobs will be further forwarded to fog node 3 from fog node 2. The above approach is also applicable when the IoT device makes three or more consecutive handovers.

Using our prototype, we have tested the scenarios where the IoT device moves among three fog nodes (a third laptop is added), making two consecutive handovers within a short period of

[1] Another possible benchmark approach is that job processing remains in the pre-handover serving node. In this case, it is straightforward to show that (1) the delay performance of those jobs uploaded to the old fog node and downloaded from the new fog node is the same as that of FMF, and (2) the delay performance of those jobs uploaded to the new fog node and downloaded from the new fog node is worse than that of FMF, since the new node has to send the unprocessed jobs to the old fog node, and the old fog node has to send the processed jobs to the new fog node.

[2] Due to the space limitation, the experimental results are omitted in this article.

FMF addresses the gap between the requirements for a seamless handover and the lack of an efficient handover timing scheme in a fog computing environment. Our real-world evaluation results showed that FMF can achieve a substantial latency reduction when a mobile device is handed over from one fog node to another.

time. In all test cases, the processed jobs are successfully returned to the IoT device.[2]

Conclusions and Further Discussions

In this article, we propose FMF, a framework supporting a new seamless handover timing scheme among different fog nodes. FMF addresses the gap between the requirements for a seamless handover and the lack of an efficient handover timing scheme in a fog computing environment. Our real-world evaluation results show that FMF can achieve a substantial latency reduction when a mobile device is handed over from one fog node to another.

There can be many future research directions for this study. First, we observe that the conditions (e.g., RSS, job status, and timing) for state transitions influence the delay performance, so our next plan is to further optimize these conditions. Second, in the current design version, the fog will only migrate processed jobs, which may not be suitable for applications with heavy data loads after job processing. The next version of the FMF system will also wisely decide if unprocessed, being processed, or processed jobs should be migrated.

Acknowledgment

Wei Bao would like to acknowledge the support of the University of Sydney DVC Research/Bridging Support Grant. Albert Y. Zomaya would like to acknowledge the support of the Australian Research Council Linkage Grant (LP160100406).

References

[1] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 3, 3rd qtr. 2017, pp. 1628–56.
[2] Y. Mao *et al.*, "Mobile Edge Computing: Survey and Research Outlook," arXiv:1701.01090 [cs.IT], accessed 13 June 2017.
[3] X. Ge, Z. Li, and S. Li, "5G Software Defined Vehicular Networks," *IEEE Commun. Mag.*, vol. 55, no. 7, July 2017, pp. 87–93.
[4] G. Pollini, "Trends in Handover Design," *IEEE Commun. Mag.*, vol. 34, no. 3, Mar. 1996, pp. 82–90.
[5] R. Kwan *et al.*, "On Mobility Load Balancing for LTE Systems," *Proc. IEEE VTC-Fall*, Ottawa, Canada, Sept. 2010, pp. 1–5.
[6] E. Stevens-Navarro, Y. Lin, and V. Wong, "An MDP-Based Vertical Handoff Decision Algorithm for Heterogeneous Wireless Networks," *IEEE Trans. Vehic. Tech.*, vol. 57, no. 2, Mar. 2008, pp. 1243–54.
[7] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," *Proc. USENIX Annual Technical Conf.*, Anaheim, CA, Apr. 2005, pp. 391–94.
[8] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, May–June 2016, pp. 42–52.
[9] T. Harter *et al.*, "Slacker: Fast Distribution with Lazy Docker Containers," *Proc. USENIX Conf. File and Storage Tech.*, Santa Clara, CA, Feb. 2016, pp. 181–95.
[10] T. Taleb and A. Ksentini, "Follow Me Cloud: Interworking Federated Clouds and Distributed Mobile Networks," *IEEE Network*, vol. 27, no. 5, Sept.–Oct. 2013, pp. 12–19.
[11] T. Taleb and A. Ksentini, "An Analytical Model for Follow Me Cloud," *Proc. IEEE GLOBECOM*, Atlanta, GA, Dec. 2013, pp. 1291–96.
[12] S. Wang *et al.*, "Dynamic Service Migration in Mobile Edge-Clouds," *Proc. IFIP Networking*, Toulouse, France, May 2015, pp. 1–9.
[13] S. Wang *et al.*, "Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs," *IEEE Trans. Parallel and Distrib. Systems*, vol. 28, no. 4, Apr. 2017, pp. 1002–16.
[14] X. Sun and N. Ansar, "PRIMAL: Profit Maximization Avatar Placement for Mobile Edge Computing," *Proc. IEEE ICC*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.
[15] A. H. Zahran, B. Liang, and A. Saleh, "Signal Threshold Adaptation for Vertical Handoff in Heterogeneous Wireless Networks," *Mobile Networks and Applications*, vol. 11, no. 4, Aug. 2006, pp. 625–40.

Biographies

Wei Bao received his Ph.D. degree in electrical and computer engineering from the University of Toronto, Canada, in 2016. He is currently a lecturer at the School of Information Technologies, University of Sydney, Australia. His research covers the area of network science, with particular emphasis on 5G systems, the Internet of Things, and mobile computing.

Dong Yuan [M] received his Ph.D. degree from Swinburne University of Technology, Australia, in 2012. He is a lecturer in the School of Electrical and Information Engineering, University of Sydney. His research interests include cloud computing, data management in parallel and distributed systems, scheduling and resource management, the Internet of Things, business process management, and workflow systems.

Zhengjie Yang received his Master of Information Technology in software engineering (research pathway) from the University of Sydney. He is interested in networking, wireless technologies, software development, mobile computing, cloud computing, and big data.

Shen Wang is a Master's student at the School of Information Technologies, University of Sydney. His research interests include gaming, networking, and software.

Wei Li [SM] received his Ph.D. degree from the School of Information Technologies at the University of Sydney. He is currently a research associate in the Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney. His current research interests include wireless sensor networks, the Internet of Things, task scheduling, energy-efficient algorithms, and optimization. He is a member of ACM.

Bing Bing Zhou (bing.zhou@sydney.edu.au) received his B.S. degree from Nanjing Institute of Technology, China, and his Ph.D. degree in computer science from Australian National University. He is currently an associate professor at the University of Sydney. His research interests include parallel/distributed computing, cloud computing, parallel algorithms, IoT and bioinformatics. He has a number of publications in leading international journals and conferences. His research has been funded by the Australian Research Council through several Discovery Project grants.

Albert Y. Zomaya [F] is a Chair Professor and director of the Centre for Distributed and High Performance Computing at the University of Sydney. He haspublished more than 500 scientific papers and is an author, co-author, or editor of more than 20 books. He is the Editor-in-Chief of *IEEE Transactions on Sustainable Computing* and serves as an Associate Editor for 22 leading journals. He is a Fellow of AAAS and IET.