Abdullaziz, O. I., Wang, L. C., Chundrigar, S. B. y Huang, K. L. (2019). Enabling Mobile Service Continuity across Orchestrated Edge Networks. *IEEE Transactions on Network Science and Engineering*, 7(3), pp. 1774-1787.

DOI: https://doi.org/10.1109/TNSE.2019.2953129

Enabling Mobile Service Continuity across Orchestrated Edge Networks

Osamah Ibrahiem Abdullaziz, *Student Member, IEEE,* Li-Chun Wang, *Fellow, IEEE,* Shahzoob Bilal Chundrigar and Kuei-Li Huang

Abstract—Edge networking has become an important technology for providing low-latency services to end users. However, deploying an edge network does not guarantee continuous service for mobile users. Mobility can cause frequent interruptions and network delays as users leave the initial serving edge. In this paper, we propose a solution to provide transparent service continuity for mobile users in large-scale WiFi networks. The contribution of this work has three parts. First, we propose ARNAB architecture to achieve mobile service continuity. The term ARNAB means *rabbit* in Arabic, which represents an **Ar**chitecture for Tra**n**sparent Service Continuity vi**a** Dou**b**le-tier Migration. The first tier migrates user connectivity, while the second tier migrates user containerized applications. ARNAB provides mobile services just like rabbits hop through the WiFi infrastructure. Second, we identify the root-causes for prolonged container migration downtime. Finally, we enhance the container migration scheme by improving system response time. Our experimental results show that the downtime of ARNAB container migration solution is 50% shorter than that of the state-of-the-art migration.

Index Terms—Mobility, Service continuity, SDN, MEC, Virtualization, Container, Migration.

1 Introduction

Future wireless networks will soon provide ultra-reliable and low latency services, such as industrial automation, e-health, and entertainment applications. For a delay-sensitive service, multi-access edge computing (MEC), network function virtualization (NFV) [1], and software defined networking (SDN) [2] [3] are three key network technologies. First, the MEC reduces end-to-end latency by bringing the services closer to the edge of the network. Second, NFV separates the network functions and applications from the underlying hardware by implementing them as software. Third, SDN provides a dynamic and responsive network for these new services. Orchestrating SDN, MEC and NFV can improve network performance. In particular, these technologies can be integrated for access convergence. For example, the 5G-CORAL project [4] has developed an integrated edge platform for multiple radio access technologies such as WiFi and cellular mobile networks. In this platform, lightweight virtualization, such as Linux containers (LXC), enables high-quality and real-time services in WiFi networks.

However, large-scale edge-enabled WiFi networks must address two challenges to provide seamless mobile services: 1) connection interruptions due to frequent handoffs, and 2) network delay due to backhauling. First, the handoff process in the standard IEEE 802.11 protocol can take up to two seconds [5]. This long interruption time may cause the connection to be dropped, thereby degrading the quality of user experience. Second, the end-to-end delay increases due to backhauling when the users leave the initial serving edge. This latency can affect the performance of delay-sensitive applications.

In the literature, several research works aim to resolve the problem of handoff interruption in WiFi networks and backhaul latency across edge networks [6]–[17]. On the one hand, the handoff interruption is resolved by reducing the scan and authentication time [6]–[10]. Although these efforts reduce interruptions, they introduce additional signaling and require modification of the user equipment. On the other hand, the backhaul latency problem is resolved by relocating applications to the vicinity of the users [11]–[17]. However, the state-of-the-art (SoA) container migration [17] has significant downtime even for small applications.

In this paper, we present an **Ar**chitecture for Tra**n**sparent Service Continuity vi**a** Dou**b**le-tier Migration (ARNAB) to resolve the frequent connection interruptions and the backhaul latency issues[1]. ARNAB uses a double-tier migration approach to provide continuous service. The first tier migrates user connectivity, while the second tier migrates containerized edge applications. Connectivity migration reduces the handoff interruption time, while application migration eliminates the backhaul latency. In the context of ARNAB, a service is a combination of user connection and applications. There are three contributions to this work:

1) *ARNAB Architecture* - A new architecture is proposed to provide transparent service continuity for mobile users. To the best of our knowledge, ARNAB is the first work to orchestrate connectivity and application migration simultaneously for edge-enabled WiFi networks.
   - Based on [19] we develop a virtual access point solution to enable ARNAB connectivity migration.

1. A preliminary and conceptual version of ARNAB is presented in [18]. In this manuscript, we analyze ARNAB in a realistic large-scale campus WiFi scenario with double-tier orchestration policy. More importantly, we characterize the root-causes for significant migration downtime, present novel strategies to reduce the downtime, and conduct experimental evaluation over a real system.

- We develop a pre-copy migration scheme to relocate containerized applications in edge environment.

2) *Downtime Analysis* - We identify the root-causes of prolonged downtime during container migration.
3) *Migration Enhancement* - Based on the analysis, we enhance the container migration scheme by incorporating real-time computing capabilities and fast storage into ARNAB. Our experimental results outperform the SoA container migration by 50% less downtime.

The rest of the paper is organized as follows. Section 2 introduces the prominent virtualization technologies and introduces 5G-CORAL as a user of these technologies. Section 3 reviews the prior art in the area of connectivity and application migration. Section 4 presents the ARNAB architecture and the double-tier migration scheme. Section 5 first analyzes the root-causes for significant downtime, and then presents our enhancements to the container migration scheme. Section 6 shows our experimental results and Section 7 describes the relevant use cases that show the applicability of ARNAB in various scenarios. Finally, we give our concluding remarks in Section 8.

2 Background

2.1 Virtualization Technologies

Virtualization is the concept of abstracting the underlaying physical infrastructure and providing virtualized resources for edge and cloud applications. Edge and cloud service providers are the driving force behind the recent advances in virtualization technologies. Traditionally, server virtualization is associated with hypervisor-based virtualization. Recently, container technology has become a promising solution for efficient resource virtualization. Many cloud service providers are now using containerization technologies [20][21].

Nevertheless, there are still blurry lines on the differences between traditional hypervisor-based virtualization, system-based containerization and application-based containerization techniques in the literature [22][23]. Thus, we provide a concise comparison between those technologies and we consider kernel-based virtual machines (KVM), LXC and Docker as references.

2.1.1 Hypervisor-based virtualization

Traditional hypervisor-based virtualization runs at the hardware level and provides independent and host-isolated virtual machines (VM). Each VM runs its own kernel and operating system (OS). Therefore, the hypervisor can create Windows guests on Linux host. However, isolation and host abstraction features come at a cost. Memory, disk, and CPU resources must be specified at runtime to execute VM kernel and OS. Also, hardware emulation is required for I/O operations (see Fig. 1a). In the case of high-density virtualization, VM deployment becomes resource-inefficient, especially for small edge and cloud applications.

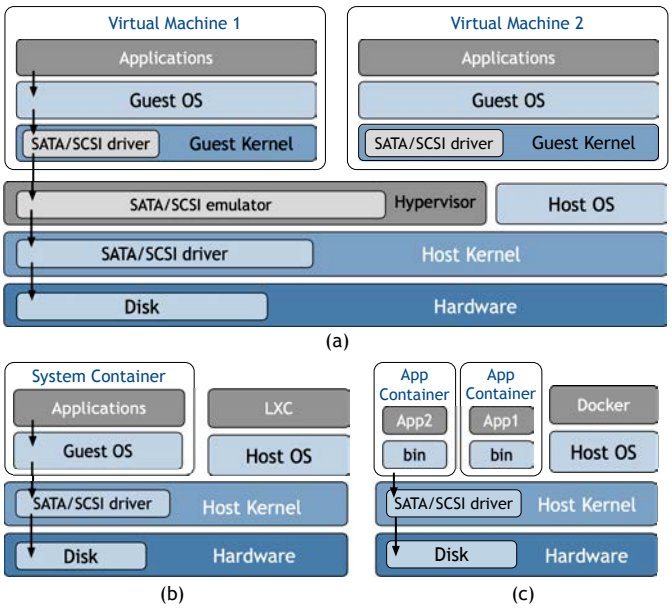


Figure 1: (a) Virtual machines (KVM), (b) System containers (LXC), and (c) Application containers (Docker)

Table 1: Virtualization technology comparison

Criteria	Virtual Machine	System Container	App Container
Boot time	long	short	shorter
Memory footprint	large	small	smaller
CPU utilization	high	low	lower
I/O ops	less efficient	efficient	more efficient
Resource allocation	predefined	fine-grained basis[2]	fine-grained basis[2]
Maintenance	multiple kernel updates per host	single kernel update per host	single kernel update per host
Varity of OS	multiple choices	Linux-based only	Linux-based only

[2] Idle memory is made available to the rest of the system

2.1.2 System-based containerization

Container isolates processes at the OS level and runs on top of the host kernel. There are two types of containers, namely system container and application container. System containers (also known as machine containers) behave like a standalone Linux system. That is, the system container has its own root access, file system, memory, processes, networking and can be rebooted independently from the host. While system containers are lightweight due to the absence of guest kernel and hardware emulation, they can only run on Linux host and are bound to the host's kernel (see Fig. 1b).

2.1.3 Application-based containerization

An application container (also known as process container) isolates an application from other applications running on top of shared kernel and shared OS. Because of sharing the same kernel and OS, application containers are lighter than system containers. A well-known example of an application container is Docker. The application container only encapsulates the necessary libraries, configurations and depend-

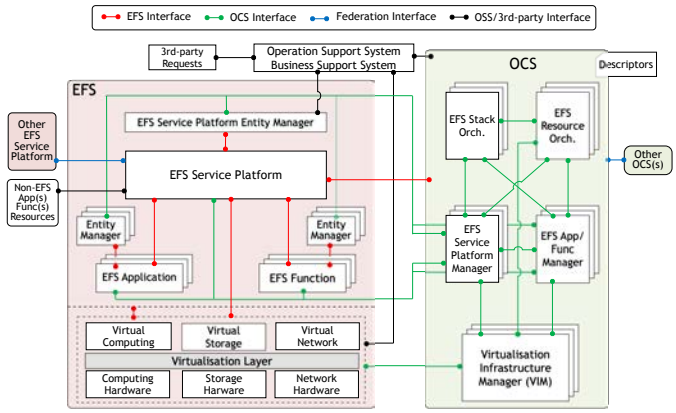


Figure 2: 5G-CORAL reference architecture [25]

encies needed to run the application (see Fig. 1c). Therefore, its resource footprint is significantly lower than VM and system containers. This fact enables the instantiation of lightweight containerized applications suitable for IoT services [24]. Table 1 compares the aforementioned virtualization technologies.

2.2 5G-CORAL

The proposed ARNAB architecture is built based on the 5G-CORAL approach to integrate MEC, NFV and SDN to deliver continuous service across the orchestrated edge networks. Here, we introduce 5G-CORAL reference architecture, including the edge and fog computing system (EFS) and orchestration and control system (OCS) as shown in Fig. 2.

2.2.1 Edge and Fog Computing System

The EFS is a logical system comprised of edge and fog resources belonging to a single administrative domain. An administrative domain is a collection of isolated resources managed by a single organization. The EFS virtualizes functions and applications and can interact with EFS in other domains. The software part of EFS consists of the following:

- *Service Platform* is a data storage for telemetric data collected from the EFS environment.
- *Function* is a virtualized instance deployed within the EFS for networking purposes.
- *Application* is a virtualized instance deployed within the EFS for serving end users and third parties.
- *Entity Manager* is responsible for applying configuration and management policies on the EFS elements as specified by the OCS.

2.2.2 Orchestration and Control System

The OCS is a logical system that manages, controls, orchestrates, and federates one or more EFS(s). It is designed to address challenges such as heterogeneity, mobility and volatility of resources. The OCS includes the following components:

- *Virtualization Infrastructure Manager (VIM)* is responsible for managing the virtual resources and the interaction between the EFS elements. Multiple VIMs can be deployed to manage one or more administrative domain(s).
- *EFS Manager* is responsible for managing the life cycle of the EFS elements. Life cycle management includes instantiating, updating, querying, scaling and terminating EFS elements.
- *EFS Orchestrator* is responsible for orchestrating the virtualized resources. It provides access to resources and also provides forwarding graph for interconnected functions and applications.

3 Existing Art

Here, we review the prior art related to handoff and connectivity migration in WiFi networks as well as containerized application migration. In addition, we also point out the challenges to achieving service continuity across edge networks.

3.1 WiFi Handoff and Connectivity Migration

In the standard IEEE 802.11, the association procedure begins with a discovery phase, where clients actively scan for available APs. During the scan, the APs that respond to probe messages become candidates for association. Once an AP is selected, the association is defined between the client's MAC address and the AP's basic service set identifier (BSSID). In this procedure, the infrastructure can not control the association decision. Therefore, to change an AP, it takes up to two seconds for the client to initiate the handoff process [5]. In order to reduce the re-association time, many schemes for fast handoff are proposed [6]–[10]. These schemes can be classified into scan-time reduction and authentication-time reduction. In the scan-time reduction, the goal is to identify a target AP for association as fast as possible. Examples for scan-time reduction schemes include sync scan [6], intelligent channel scan [7], neighbor graph [8], selective neighbor caching [9], AP prediction [10] and IEEE 802.11k. In authentication-time reduction, the concept of pre-authentication is introduced in [10] and also specified in the IEEE 802.11r. Although the proposed schemes can reduce the re-association delay, these schemes require modification to the client side and introduce additional signaling. This fact challenges the concept of bring-your-own-device, which implies that the infrastructure should accommodate diverse range of user devices.

To move the client-AP association decision from the client to the infrastructure, the concept of virtual access point (vAP) is introduced in [26]. The vAP is a network function abstraction which is created for connecting clients. Each client associates with a dedicated vAP, which consists of the following parameters: (1) client MAC address, (2) client IP address, (3) a fake BSSID, and (4) service set identifier (SSID), to be used in the communication. A client associated with a vAP periodically receives a beacon to keep it aware that its AP is still available within range. The signal level received from the client is stuffed in the beacons so that the neighboring APs can overhear. Every AP maintains two lists: managed and

monitored lists. The managed list contains the clients that are currently associated with the AP and the monitored list contains clients that the AP can hear. Connectivity migration occurs when a neighboring AP receives a beacon from the client with a signal level higher than what is advertised by the serving AP. This approach moves the association decision to the infrastructure, but has two drawbacks. First, all the APs operating on the same channel will hear the advertised signal level. This makes the solution impractical for actual deployment and frequency planning. In addition, since the management of vAPs is done in a distributed manner, there is a lack of a global view.

Later, a multichannel vAP handoff extension of [26] is proposed in [27]. In this solution, the APs operate in different channels and communicate with each other to manage client mobility. After the client connects to an vAP managed by an AP, the AP monitors the client signal-level. If the signal-level goes below a threshold, the AP sends a scan request to the neighboring APs. When a neighboring AP responds to the request, the client is forced to switch channel and continues the communication with the new AP. This solution overcomes the interference issue caused by operating on the same channel, but still lacks a global view of the entire infrastructure.

Alternatively, a software defined WiFi network framework called Odin is proposed in [28]. Odin integrates SDN-based solutions to the concept of vAP. That is, the programmability and global view of the network can be used to manage client mobility. In [29], the Odin framework is utilized to move vAP between different APs, while a client is generating game traffic. Although the solutions of [28] [29] offer management flexibility and scalability, it is still assumed that all the APs operate in the same channel. More recently, a solution that provides both the multichannel environment and a global view is presented in [19] [30]. In ARNAB, we develop user connectivity migration to enable service continuity at the network access.

3.2 Container Migration

Container migration can be classified into stateful and stateless. In stateless migration (also known as non-live migration), the volatile state (i.e., memory pages and execution state) of the container is not preserved when the container is moved to the destination node. In the case of stateful migration (also known as live migration), the state of the container is preserved when the execution of the service is resumed at the destination node. There are four types of stateful migration techniques as follows:

- *stop-and-copy* - freezes the container, checkpoints its state, copies the container's image and state to the destination, and then restore the container from the state checkpoint [31] [32].
- *pre-copy* - performs iterative state checkpointing and transfer while the container is running till the amount of the dirty pages is at minimum, and then concludes with a shorter stop-and-copy [33]. Iterative checkpointing reduces the size of the final checkpoint that is performed when the container is frozen. This minimizes the time required for the final checkpoint and the time it takes to copy the checkpoint to the destination node.
- *post-copy* - performs a short stop-and-copy to move container's execution state, and then starts the container at the destination node and retrieves the memory pages when needed [34]. This type of migration has small downtime, but containers may suffer from performance degradation due to the time needed to wait for the requested memory pages.
- *hybrid-copy* - combines pre-copy and post-copy schemes to reduce the total migration time and the performance degradation due to fault pages transfer [35] [16]. First, hybrid-copy performs a state pre-copy and transfer the dumped state while the container is running. Consequently, it freezes the container to checkpoint only the execution state and then transfers it to destination node. Finally, the container is resumed from the available state at destination and the outdated memory pages (i.e., pages which are dirtied after the pre-copy phase) are retrieved when needed.

In hypervisor-based virtualization, VM migration is well investigated [36]–[39] and several solutions are commercially available. For instance, a pre-copy VM live migration scheme is presented in [33]. An active VM continues to run while iteratively pre-copying state. During a consecutive iteration, only changes in memory are transferred. At last, a final state copy is performed while the VM instance is frozen and then transferred to the destination node. This way, the down-time is greatly reduced compared to a pure stop-and-copy scheme. Although VM migration has matured, many existing solutions are tailored to data center environment where network attached storage (NAS) and specific virtualization technology are utilized. NAS enables all host machines in the data center to access a network-shared storage eliminating the need to migrate disk storage. However, in the case of migration between edge networks, state and disk storage have to be relocated over the network [40] [41]. In [41], a non-NAS VM migration schemes is proposed for edge networks. Changes in VM disk storage and memory state are tracked then deduplication and compression are utilized in a parallel to improve agility and reduce bandwidth utilization.

Lately, container migration has caught more attention from the research community [14]–[17]. This is because containerization technology is superior to traditional hypervisor-based virtualization, especially in terms of resource efficiency and performance. For instance, Voyager [15] is a stop-and-copy container migration scheme which combines page-server for volatile state transfer and NAS for persistent data transfer in cloud environment. Furthermore, a comprehensive evaluation of different migration schemes for application-based containers in edge environment is introduced in [16]. Recently, a framework for migrating system-based containers is presented in [17]. This is the first framework to consider edge environment for container migration. Fundamentally, the framework is a layered model designed to reduce the downtime during the migration process. Al-

though this framework is suitable for edge networks, it suffers from long downtime and relies on stop-and-copy migration, which is not an effective method for containers with large state. In ARNAB, we develop a pre-copy migration scheme and propose strategies to significantly reduce the downtime when migrating system-based and application-based containers across edge networks.

3.3 Mobility Support in Edge Networks

In wireless networks, the low latency provided by edge networks can only be leveraged within the coverage of the corresponding network. In other words, mobility can cause service interruption when users move away from the initial serving edge. Supporting user mobility in edge networks has been discussed by many research efforts [12]–[14], [42]–[47].

For instance, follow me cloud (FMC) [13] is a novel architecture that enables cloud services (i.e., running in distributed data centers) to follow the users as they roam through the network. The FMC controller manages computing and storage resources of the data centers and decides which data center the user should be associated with. Based on FMC, a migration mechanism is developed to ensure service low latency [42]. However, the minimum reported migration downtime remains high for seamless service experience.

In [43], companion for computing (CFC) is a novel platform proposed to support IoT mobility through orchestrated container migration. CFC continuously monitors the latency between a fog service (i.e., a container) and user device. Once the latency is above a defined threshold, CFC finds a suitable fog node at the near proximity of the user. CFC caters for user mobility at the computing tier while being application latency-aware. However, mobility support at the access network is not enhanced, migration downtime remains significant and migration policy is not transparent to the user device. Similar to [43], Foglets [44] migrates geo-distributed IoT application components based on application latency measurements. The experienced latency is continuously monitored through ping messages, and then the application component is migrated to a suitable node when the latency exceeds a threshold. However, there is no details on which migration scheme is used to migrate application components between fog nodes and the standard radio handover is assumed.

In [45] and [46], follow me fog (FMF) and seamless fog (sFog) are proposed to pre-migrate computation jobs before radio handover occurs during user mobility. This is accomplished by constantly monitoring the received signal strength indicator (RSSI) from different fog nodes. Once the RSSI of the current node (i.e., serving the user) keeps decreasing and the RSSI of another node keeps increasing, the computing jobs are pre-migrated to the new node before the re-association takes place. This way, FMF and sFog support user mobility by predicting the target fog node beforehand and thus reducing the waiting time for computing jobs to be available. However, these schemes are not transparent to user device since the procedures are orchestrated at both the user device and fog nodes. In addition, service interrupt due to the actual handover event and the container migration scheme remain significant.

Furthermore, an architecture called SharedMEC is proposed to support user mobility in edge cellular networks [14]. The architecture combines the standard cellular handover process with service handover. In [47], a mobile-edge gateway is proposed to enable mobility in cellular networks by constructing mobile user location and steering traffic to respective edge based on the calculated location.

In cellular networks, a handover event is handled by the infrastructure and its interruption time is within $200\ ms$ [48]. However, the handoff decision in WiFi networks is locally made by the user equipment and re-association process takes up to two seconds [5]. For these reasons, ARNAB moves the handoff decision from the user device to the infrastructure for control flexibility and interruption management. More importantly, ARNAB is transparent to user and capable of migrating containerized applications with very small downtime. The simultaneous orchestration of user connectivity and applications in WiFi networks makes ARNAB a novel solution for smart cities, campuses, airports, museums and shopping malls.

4 The ARNAB Architecture

We propose a new architecture called ARNAB to provide mobile service continuity. ARNAB stands for: **Ar**chitecture for tra**n**sparent service continuity vi**a** dou**b**le-tier migration[3]. The term ARNAB means *rabbit* in Arabic and it is dubbed for the architecture because it illustrates the behavior of the service hopping through the infrastructure following the user. Moreover, ARNAB is transparent to the user because it does not require any modification to the user device. The ultimate goal of ARNAB is to provide a seamless user experience through continuous service delivery. ARNAB employs a double-tier migration, namely user connectivity migration and edge application migration. On one hand, the first tier utilizes vAP to minimize WiFi handoff interruptions and to move the association decision from the clients to the infrastructure. On the other hand, the second tier utilizes pre-copy migration to reduce downtime when relocating containerized application across edge networks. Fig. 3 illustrates the architectural model of ARNAB which adopts centralized management approach for campus WiFi infrastructures. On a high-level, ARNAB architecture consists of three layers namely, network access, aggregation and management layers. The network access layer is partitioned into regions R_i and comprised of the APs which provide WLAN connectivity to the users. Each region consists of multiple APs AP_n which connect to the same access switch. The aggregation layer interconnects the APs and edge nodes (i.e., EFS), and OCS. Finally, the management layer consists of the OCS and its management applications.

3. The software components of the double-tier scheme are available at https://github.com/oiasam/ARNAB

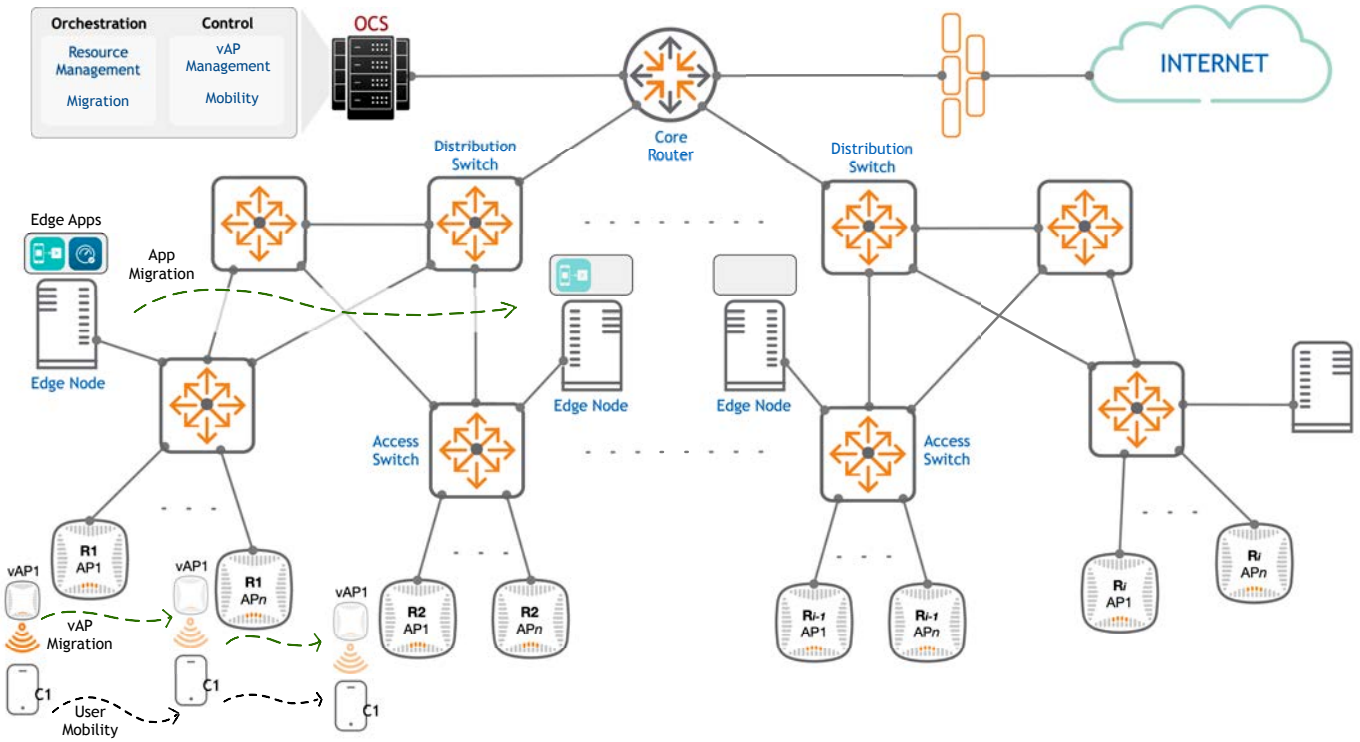


Figure 3: Architectural model of ARNAB

4.1 Tier 1: User Connectivity Migration

ARNAB leverages several innovative technologies namely, SDN, Odin framework [29], click modular router [49] and OpenWRT firmware [50] to deliver seamless WiFi experience using commodity hardware. Odin for ARNAB consists of a master and agents, and provides two southbound protocols, namely OpenFlow and Odin protocol. The master, which is the control part of the OCS, uses OpenFlow protocol to install flow rules in APs to redirect traffic when migrating vAPs. In addition, the controller uses Odin protocol to communicate with the agents running on the APs. In the experimental setup, the used WiFi APs have two wireless interfaces and run click modular router as part of OpenWRT. Finally, the clients' association with vAPs and the migration of the vAPs during mobility are managed by a mobility application running in the controller. The Logical steps and example of user connectivity migration are detailed as follows:

Step 1: *vAP-association* - a client $C1$, scanning for a network, associates with a dedicated vAP_1 that is spawned by the agent running on AP_1 and operates in channel CH_6.

Step 2: *vAP-keep* - vAP_1 periodically unicast beacon frames to its corresponding client to keep it informed that AP_1 is still within its radio reach.

Step 3: *RSSI-monitoring* - AP_1 monitors RSSI from $C1$ and informs the OCS when the RSSI is below a predefined threshold.

Step 4: *vAP-migration* - The OCS commands all the APs at the proximity of the client to switch their secondary interface to CH_6 to listen to the client's packets. Then, the OCS identifies the best candidate AP for the vAP migration. In this example, AP_n operating in CH_1 is selected. Next, the OCS command AP_1 to inform $C1$ to switch to channel CH_1. Concurrently, the OCS initiates vAP_1 in AP_n which starts unicasting beacons to $C1$. Finally, once $C1$ has switched to CH_1, its connection is handed over to AP_n and the vAP_1 in AP_1 is terminated.

From the client's point of view, the AP that the client is associated with has not changed. In fact, its vAP is still the same, but its connection has been migrated to a different physical AP. In this way, the handoff decision is shifted from the client equipment to the network infrastructure. At the same time, the handoff is no longer required to support user mobility. Instead, channel switch and its relatively small delay is what the client experiences when changing APs.

4.2 Tier 2: Edge Application Migration

ARNAB application migration enabling technologies include LXC, checkpoint and restore in user-space (CRIU), and remote synchronization (*rsync*). The orchestrator part of the OCS manages the lifecycle of the containerized application in the EFS nodes. It supports lifecycle management operations such as instantiation, cloning, migration, scaling and

termination. CRIU is used to dump the in-memory state of the migrating containers. The local-disk (file system) and the state of the containers are transferred by utilizing *rsync* for its remarkable speed and efficiency.

To migrate a container between edge nodes with minimal downtime, ARNAB utilizes a pre-copy procedure which is summarized as follows:

Step 1: *Local-disk-copy* - the container file system is assumed to be available in all edge nodes to reduce traffic overhead and to keep the total migration time to a minimum. Local-disk synchronization is performed to copy application related files.

Step 2: *Iterative-pre-copy* - the container state is dumped to source node storage, and then copied over to the destination while the container continues to run. Next, pre-copy iterations are performed to dump and copy only the memory pages that have changed (dirtied) since the previous checkpoint.

Step 3: *Freeze-then-copy* - the container is frozen in this step, and then a final local-disk synchronization, state checkpoint, and copy are performed. The downtime observed by the user occurs during this step.

Step 4: *Restore-and-terminate* - the container is restored at the destination node and the frozen container in the source node is terminated.

It is worth noting that the edge node and the AP can be the same physical device which has communication, computing and storage capabilities. Also, the connectivity migration is not necessarily a trigger for the application migration. Different applications demand different latency requirement. In the following subsection, we discuss possible orchestration policies for the proposed double-tier migration scheme.

4.3 Orchestration Policy

Although application migration enables low-latency connection to the edge, edge application migrations should be minimized to maintain seamless service delivery and achieve scalability since it often involves large data transfer and relatively longer downtimes than connectivity migration. As such, application migration is not coupled with the connectivity migration. Users of ARNAB can still enjoy the required latency even after re-associating with different APs. Here, we discuss two possible policies that can govern the orchestration of application migrations.

- *Latency Monitoring* - The latency can be measured between the user device and the serving edge. Applications latency can range from tens of milliseconds (e.g., tele-surgery) to several seconds (e.g., video streaming). The advantage of using such policy is the high-level of granularity in making migration decision based on the latency reports and application requirement. However, transparency to user device is a disadvantage. The user device has to collaborate with the edge for the latency monitoring.
- *Regional Access Change* - Infrastructure planning can play a major role on the migration decision making. Instead of continuously measuring the application latency, the WiFi infrastructure can be partitioned into regions clustering APs together at the access layer and interconnecting them at the aggregation layer. This way, a region can be served by edge node(s) and a regional access change can trigger application migration. Edge nodes can also be provisioned at the aggregation layer. Depending on the application latency requirement, the migration can be triggered if the number of hops has reached a predefined value for the respective application. For example, in the case of a time-sensitive application, mobility Case 2 (see Fig. 3) may trigger the application migration from Edge Node 1 to Edge Node 2. The advantages of using such policy are: 1) simplicity of implementation - the controller is aware of the user proximity and hence obtaining regional information is easy, 2) transparency to the user - the infrastructure solely handles the orchestration policy tasks, thus the user device is not involved in the latency monitoring. However, this policy requires good infrastructure planning.

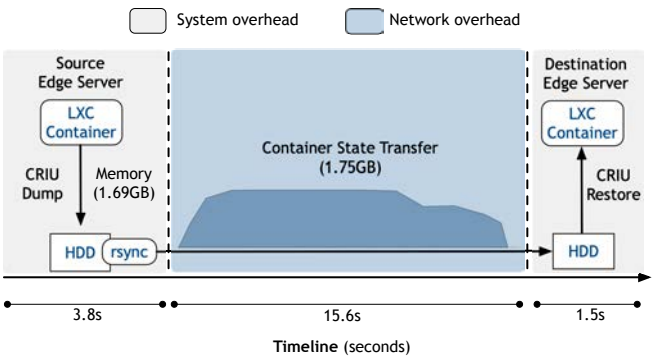


Figure 4: Stop-then-copy downtime anatomy

5 Container Migration Enhancement

We acknowledge that even with the improvement introduced by the developed pre-copy migration procedure, the downtime remains significant even for small applications (i.e., close to three seconds). As such, we first investigate the root-causes of latency associated with the container migration, and then introduce edge node enhancements to further reduce the downtime.

5.1 Downtime Anatomy and Definition

In general, migration schemes share a common process which involves freezing the container, checkpointing memory state (whole or part) to the source storage, copying the state to destination, and finally restoring the container at the destination. To understand the reasons behind prolonged migration downtime, we take a closer look at the stop-and-copy migration scheme. Fig. 4 shows an anatomy of the stop-and-copy scheme when migrating an Ubuntu system-based container. We note that there are three vectors that contribute to the downtime, namely a) system latency, which is the time taken by the system to execute migration tasks,

b) network latency, which is the time taken to copy container state from source to the destination, and c) container image overhead, which is the overhead cause by running unnecessary processes in the container. Running unnecessary processes increases the size of the container state and hence increases the time taken to checkpoint, copy and restore the container.

In the example illustrated in Fig. 4, the time taken to checkpoint and restore the container is 2.42 seconds, collectively. It consists of the time required to execute the CRIU codes and the time it takes to write and read the checkpoint files. On the other hand, the network latency (i.e., 0.44 seconds) represents the time taken to copy the volatile and persistent states over a 1 Gbps link using *rsync* and also the time that it takes the destination node to write the received files.

Accordingly, modeling the downtime in pre-copy migration depends on the following factors: the system response-time in executing migration tasks, the size of the container state, the container workload in terms of dirty pages rate, and the network transmission rate. We denote the maximum number of iterations as n and the memory copied in every iteration as M_i where i is $0 \leqslant i \leqslant n$. The data copied in the iteration i is calculated as:

$$M_i = \begin{cases} M_{init}, & \text{if } i = 0; \\ D_{rate} \cdot T_{c_{i-1}}, & \text{otherwise} \end{cases} \tag{1}$$

where D_{rate} is the memory dirtying rate and T_{c_i} is the elapsed time while copying memory at each iteration. The T_{c_i} can be calculated as:

$$T_{c_i} = \frac{D_{rate} \cdot T_{c_{i-1}}}{C_{rate}} = \frac{M_i}{C_{rate}} \ , \tag{2}$$

where C_{rate} is the copying rate. Note that the copying time depends on the amount of memory and the copying rate. The system latency during checkpointing is denoted as T_{cp_i} and represented by:

$$T_{cp_i} = T_{cpf_{i-1}} + T_{fw_{i-1}} \ , \tag{3}$$

where T_{cpf_i} is the time taken to execute the checkpoint function and T_{fw_i} is the time taken to write the process tree and resources into files. Similarly, the restore time is represented by:

$$T_{rt} = T_{fr} + T_{rtf} \ . \tag{4}$$

From (2) and (3), we obtain the system and network latency at each iteration:

$$T_i = T_{cp_i} + T_{c_i} \ . \tag{5}$$

Combining (2), (3), (4) and (5) for the n^{th} iteration, the migration downtime can be calculated by:

$$\begin{aligned} T_{down} &= T_{i=n} + T_{rt} \\ &= T_{cpf_{n-1}} + T_{fw_{n-1}} + \frac{M_n}{C_{rate}} + T_{fr} + T_{rtf} \ . \end{aligned} \tag{6}$$

Lastly, the total migration time can be obtained by adding container image synchronization T_{isync}, the summation of T_i, and the restore time. Then, it follows that

$$T_{migrate} = T_{isync} + \sum_{i=0}^{n} T_i + T_{rt} \ . \tag{7}$$

Most of the existing works attempt to reduce the migration downtime by reducing the network latency [42][17]. However, we believe that the system latency and container overhead are equally important. In the following, we discuss how to reduce the migration downtime by means of real-time computing capabilities and fast storage.

5.2 Real-time Computing Capabilities

Checkpoint and restore (C/R) functions are computationally expensive. The checkpoint function collects the process tree and resources, freeze the process, and then write them into files. The restore function reads the files, resolves the shared resources, forks the process tree, and then restores the process resources. Obviously, the migration scheme spends long time waiting for the checkpoint and restore functions to execute. This is because the Linux general-purpose kernel can not provide time guarantees for applications execution. The lack of time guarantees can cause time-sensitive tasks to experience unpredictable delays [51]. Systems with real-time computing capabilities can resolve this issue by providing deterministic behaviors. In fact, the concept of real-time execution is usually misinterpreted. Real-time execution is not to execute a task as fast as possible, but rather it is to execute a task as fast as specified (i.e., deterministic).

Currently, there are three common approaches for providing real-time capabilities in Linux system. In the first approach, an embedded real-time OS is used for critical tasks and a general-purpose kernel is used for non-critical tasks in the same system [52]. The second approach utilizes a dual-kernel where a microkernel runs in parallel with a separate Linux kernel. This approach is employed by Xenomai [53] and RTAI [54]. The third approach uses a single kernel with real-time preemption support. Real-time Linux (RTL) project [55] is an example of this approach. RTL initiative aims to create a predictable and deterministic environment to turn Linux into a real-time capable kernel.

To bring real-time capabilities to ARNAB, we utilize the real-time preemption (PREEMPT-RT) patch maintained by RTL. The deployed patch improves the overall system response-time through the following features:

- *Priority inheritance* - resolve the phenomenon of priority inversion in which a high priority task being blocked by a low priority task for an unspecified period of time. Priority inversion can cause a critical issue for real-time applications since it affects scheduling and predictability of execution. The priority inheritance solves this issue by allowing the low priority task to inherit the highest priority of the blocked tasks to execute its critical section then return to its original priority after exiting.

- *High resolution timers* - allow precise timing for scheduling tasks and eliminate the need for periodic scheduler ticks.
- *Kernel preemption* - allow most of the kernel to be preempted, except for small critical regions. This is accomplished by converting kernel spinlocks to real-time mutexes that adopt priority inheritance and hence are preemptive [56].
- *Interrupts management as threads* - allow interrupt request (IRQ) handlers to be run in kernel threads. Managing IRQs as threads enables priority assignment, allowing the IRQ handlers themselves to be preempted, thereby mitigating latency due to interrupts.

Beside applying the PREEMPT-RT patch, we also implement additional configurations to further improve the ARNAB system response time. The additional performed configurations are detailed in Table 2.

Table 2: Additional kernel configurations used in our experiment

Configuration	Remarks
File system with mount *noatime* option enabled	Registering file reading access time is disabled. As such, using *noatime* may lead to significant performance gains.
CPU *frequency scaling* with performance governor	Clock scaling allows changing the clock speed of the CPUs on the fly. Performance governor sets the CPU to operate at the highest frequency.
Reduced memory *swappiness*	Reduce tendency of copying RAM contents to the system swap space. This is also useful when using RAM disks.
High resolution kernel timer (1000Hz)	Increase the resolution of the hardware timer for interrupting the kernel (best-case resolution of 1ms). This reduces the kernel latency when performing process accounting, scheduler time slice accounting, and internal time management.
A *tickless timer* support	Turn off the timer tick only when a CPU is idle.

5.3 CPU Shielding

Beside the real-time tunings applied to kernel, CPU shielding can further improve execution response of migration tasks. CPU shielding is the ability to dedicate a CPU core to running real-time tasks and the interrupts associated with those tasks. This feature allows the CPU resources to be reserved for high-priority tasks thus providing a more deterministic environment to support the execution of real-time applications. To shield a CPU, the system kernel must permit binding one or more processes to one or more CPU cores. This concept is called CPU affinity. In Linux, an interface for setting and retrieving a process CPU affinity is introduced since the 2.5 kernel. There are two kinds of CPU affinity namely, soft affinity and hard affinity. On one hand, soft affinity refers to the tendency of a scheduler to attempt to keep processes on the same CPU core as long as possible. The processes move to another core when it becomes infeasible to remain on the same core. On the other hand, hard affinity refers to the ability to enforce process-CPU binding. This is achieved by CPU affinity system interface. When a process is bound to a CPU core, it must adhere and only run on the assigned CPU core.

One issue that may affect time-sensitive applications is processes bouncing between CPU cores. This can cause cache invalidations and hence increases cache-miss rate. CPU affinity prevents this issue and improves cache utilization. Time-sensitive application processes can be bound to one core and other system processes can be bound to the remaining cores. This guarantees that the time-sensitive tasks obtain complete attention from the dedicated processor. In ARNAB, we scale the CPU frequency of the EFS nodes to performance governor which allows operating at the maximum frequency and thus enhancing system response. The higher the CPU frequency the more instructions can be retired by the CPU over a unit of time. However, scaling up CPU frequency comes with a cost. The higher the frequency the more power is drawn by the CPU. As such, we combine the benefits of both, the CPU affinity and the CPU frequency scaling, to reduce the effect of this tradeoff. That is, first we shield a CPU core from the kernel scheduler in ARNAB multiprocessors systems so it can be dedicated to time-sensitive processes. Then, we bind all the migration related tasks to this dedicated core. Finally, the shielded CPU core frequency is scaled to performance governor while the rest of the CPU cores continue to operate at power-save mode.

5.4 Storage Performance

The checkpoint function writes the process tree and resources into files, while the restore function reads the files to fork the process tree and restore the resources. Both of these functions perform memory and I/O operations which are usually slow, especially on rotational block devices such as hard drives. Therefore, we observe that storage speed is an important factor in the performance of the checkpoint and restore function and ultimately the migration scheme. Here, we study different types of storage, namely hard disk drive (HDD), solid state drive (SSD), RAM disk, also known as temporary file system (TMPFS), and persistent memory file system (PMFS). First, we briefly explain each storage technology and state their advantages and disadvantages. Then, in Section VI, we evaluate each storage speed, taking into account different file sizes to determine which storage is best for various migration scenarios.

5.4.1 Hard disk drive (HDD)

HDD is a rotational storage device that uses hard magnetic surface to persistently write data in blocks of 512 bytes. To read from a magnetic surface, HDD senses the magnetic patterns on the disk and to write to disk it induces a change into the magnetic surface. This mechanism is accomplished by disk rotation and disk head. Normally, input/output (I/O) operation in HDD experience delays including seek time (i.e., the time it takes the head to find the target track), rotational delay (i.e., the time it takes the target sector to arrive under the head after rotation) and transfer time. Although HDD technology has evolved throughout many years, it has not

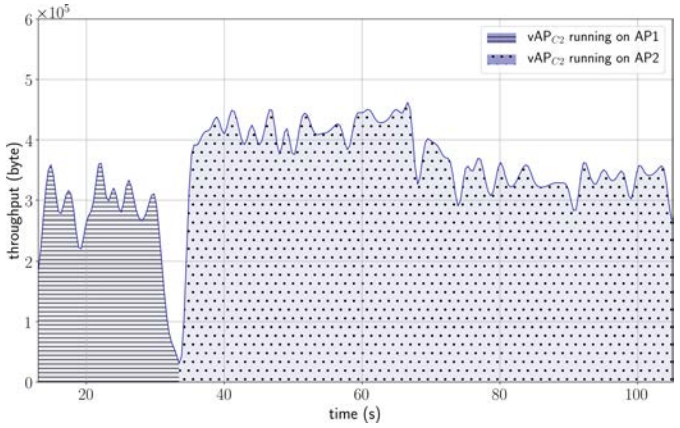


Figure 5: Throughput impact of video streaming application during vAP migration

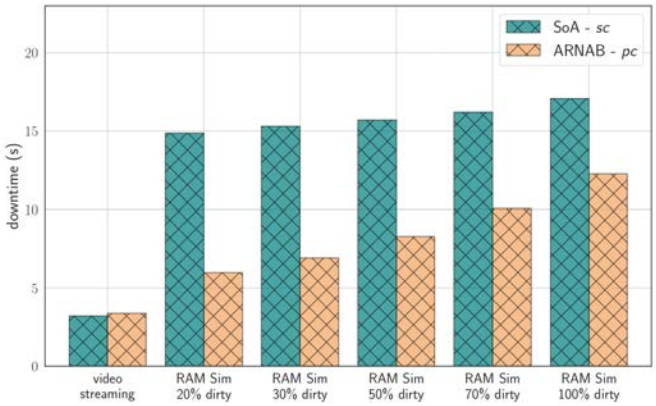


Figure 6: Preliminary comparison between ARNAB pre-copy (pc) and the SoA [17] stop-and-copy (sc) migration downtime.

delivered a cost-effective solution for enterprise applications that require high capacity and performance at the same time. It is reported in [57] that HDD capacity increases by 40%, while its I/O performance increases by only 2% annually.

5.4.2 Solid state drive (SSD)

SSD has no mechanical mechanism because it is built from flash-type memory chips. Since there are no moving parts in SSDs, they have no rotational delay and near-zero seek time which allow them to access data with high speed and precision. In addition, unlike HDDs, SSDs have consistent performance because they are not affected by fragmentation. While SSD have exceptionally higher I/O performance than HDD, they offer less capacity per drive, relatively more expensive, and have endurance limit.

5.4.3 RAM disk (TMPFS)

TMPFS does not use traditional non-volatile media to store files. Instead, TMPFS reside in the virtual memory maintained by the kernel. TMPFS is primarily designed to enhance performance as it allows short-term files to be written and read without generating disk I/O. The enhanced performance is due to leveraging the kernel resource management policies [58]. However, TMPFS has no dedicated disk space and its data can be swapped out to drive to free up virtual memory resources for other needs.

5.4.4 Persistent memory (PMFS)

Recently, flash storage technology has brought down the performance gap between storage and memory significantly. For example, non-volatile DIMM (NVDIMM) connects storage directly to a dual data rate interface. NVDIMM is becoming popular due to its ability to provide high input/output operations per second (IOPS) with low latency and predictable performance. But, NAND-based NVDIMM is still accessed as a block device since it has a separate address space. PMFS has the non-volatility feature of flash storage and the byte-addressable feature of memory [59]. Since PMFS is byte-addressable like memory, it is optimized to have a direct access to memory without going through the block layer. That is, PMFS utilizes memory-mapped I/O where load/store are used instead of read/write instructions. Currently, PMFS emulation is supported in Linux systems with direct access (DAX) to persistent memory.

Table 3: Hardware and software specifications used in the experimental setup.

Edge Servers	Hardware	Model	ProLiant DL160
		CPU	Intel Xeon 2.10GHz
		RAM	124GiB DIMM 2400
		Network	2 × I350 Gigabit
	Software	OS	Ubuntu 16.04
		Kernel	4.4.0-119-generic
		LXC1	3.0.3
		CRIU	3.11
Access Points	Hardware	Model	TL-WR1043ND v2
		Wireless Adapter	Qualcomm Atheros
		Flash	8MB + (16GB)
		Network	4×Eth 2×Wireless
	Software	OS	OpenWRT (15.05.1)
		Kernel	3.18.23
		OVS	2.8.2
		Click	2.1

6 Results and Discussions

In this section, we first present preliminary results of the proposed double-tier migration scheme, including a) throughput evaluation of a containerized video streaming application during user connectivity migration, and b) downtime evaluation during the migration of applications. Then, we evaluate the enhanced container migration in terms of execution determinism, system latency and overall downtime. Fig. 3 also shows the experimental setup of ARNAB. The hardware and the software specifications for the setup are detailed in Table 3.

6.1 Preliminary Experimental Results

To measure the impact on application throughput due to connectivity migration, a video streaming server is installed in a container. This server is made accessible through the client browser. Fig. 5 shows the impact of the vAP migration

on the application throughput. The dashed segment of the plot represents the application throughput when the client is associated with vAP_1 running on AP_1 that operates in CH_6. The dotted segment of the plot represents the application throughput after the migration, when the client is associated to vAP_1 running on AP_n that operates in CH_1. Because the signal level is below the threshold, the OCS migrates vAP_1 from AP_1 to AP_n. Note that the client only sees a channel switch. Thus, the client's TCP session resumes after the migration. The impact on the video streaming throughput is momentary because channel switching interruption is very small. Channel switching delay depends on the hardware of the physical APs. In our case, the channel switching time of the APs used in experiment is between $90\ ms$ to $110\ ms$ [19].

To benchmark the application migration, we implement stop-and-copy (*sc*) scheme to reproduce the results presented in [17] and evaluate its downtime against the proposed pre-copy (*pc*) migration scheme. In this evaluation, we measure the downtime during the migration of two containerized applications, video streaming (14 MB of container state) and RAM simulation (1.2 GB of container state) applications. Both applications are encapsulated within LXC system-based containers running Ubuntu 16.04. The RAM simulation application represents those applications with intensive use of memory (i.e., high rate of dirty pages) such as in-memory database, big data analytics and deep learning.

To simulate RAM simulation application, a very small program is used to load data and manipulates the loaded data to reflect page dirtying. This is particularly important when *pc* migration is utilized. The container state size of every iteration in *pc* procedure depends on the rate of dirty pages. Fig. 6 shows the downtime when performing *sc* [17] and the proposed *pc* migration. The results are based on average values of ten trails for each reported case. In the case of video streaming application, *pc* shows minimal improvement over *sc*. Most of the downtime is due to the execution of the checkpoint and restore of both migration schemes rather than the time taken to copy the container state to the destination.

In the case of RAM simulation, note that the downtime of *sc* is much higher than *pc* due to the amount of data being transferred when the container is frozen. For example, when 20% of the state frequently changes, the downtime of *sc* and *pc* are approximately $14.7\ s$ and $6\ s$ during state copy of 1.2 GB and 171 MB, respectively. Furthermore, as the rate of dirty pages increases, the downtime of the *pc* scheme slightly increases but still outperforms the *sc* scheme. Note that the downtime of the *sc* also slightly increases with the dirty rate. This state increase is attributed to the OS executing data manipulation and not to the application data itself.

6.2 Enhanced Container Migration

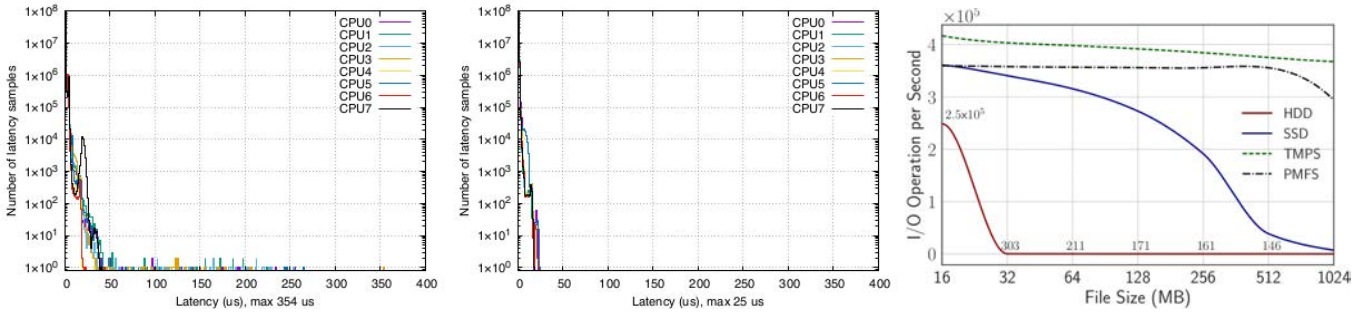
The ARNAB's EFS kernel latency is evaluated using the Linux kernel tool *cyclictest* which measures the difference between the expected wake-up time of a thread and its actual wake-up time. This way, *cyclictest* provides statistics about the EFS latency. Fig. 7(a) and Fig. 7(b) show the latency and the predictability analysis of the general-purpose and ARNAB configured kernels. It can be seen that the configured EFS kernel has very low latency compared to the general-purpose kernel. The maximum measured latency for the general-purpose and the configured kernels are $354ms$ and $25ms$, respectively. In addition, Fig. 7(b) also indicates that the EFS latency is predictable because the measured latency does not vary much. Therefore, it is clear that ARNAB can provide determinism and low response time for delay-sensitive applications. The parameters used in this experiment are shown in Table 4.

For the evaluation of storage speed, we utilize a tool called flexible I/O [60] to measure the input/output operation per second (IOPS). Random read/write and different file size are considered in this experiment to understand the behavior of storage technologies when the file size increases. Fig. 7(c) shows that when the file size is small, the IOPS is not significantly different especially in the case of SSD, TMPFS and PMFS. However, when the file size increases, HDD and SSD performance degrades substantially. Thus, PMFS and TMPFS are recommended when migrating containers with large in-memory state.

To show the impact of the enhanced EFS on migration, we evaluate the determinism behavior and the execution time of LXC functions namely freeze, checkpoint and restore. Fig. 8(a) shows thirty trails for performing each function on a blank Alpine system container. The *x*-axis shows the number of the sample taken while the *y*-axis shows the respective execution time in milliseconds. Clearly, the general-purpose kernel exhibits unpredictable delays with standard deviations of $449\ ms$, $57\ ms$, and $25\ ms$ for freeze, checkpoint, and restore functions, respectively. On the contrary, the enhanced EFS provides predictable execution time with low standard deviations of $1.03\ ms$, $11.5\ ms$, and $8.9\ ms$. Furthermore, the enhanced EFS kernel not only improves determinism, but also reduces the system latency. We compare the average time of LXC function execution between the general-purpose and the enhanced EFS kernels. The result in Fig. 8(b) suggests that the average performance gains when running the enhanced kernel are 95%, 74%, and 33% for freeze, checkpoint, and restore, respectively.

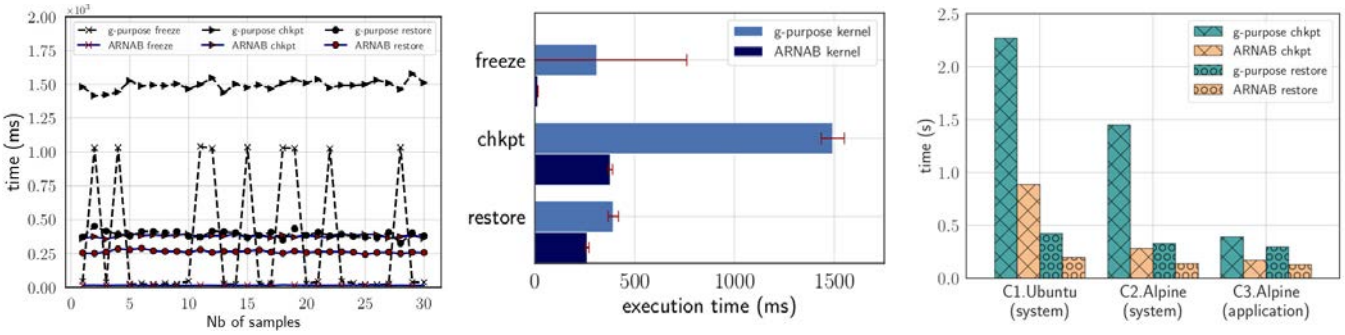
In addition, container overhead plays an important role in migration performance. Optimizing container image reduces the number of processes to checkpoint and restore during migration. In Fig. 8(c), we analyze the execution time when performing checkpoint and restore for system containers (Ubuntu and Alpine) and application container (OCI Alpine). The obtained result shows significant execution time reduction in the case of the enhanced EFS. This also proves that the lighter the container image, the shorter the migration downtime.

In Fig. 9(a), we plot the empirical cumulative distribution function (eCDF) to show the benefit of using CPU shielding for the migration tasks. A blank Alpine 3.7 is used to compare between general purpose kernel and ARNAB with and without process affinity. The *x*-axis represents the downtime in seconds of each checkpoint experiment while



(a) General-purpose kernel latency (b) ARNAB EFS kernel latency (c) IOPS for different storage types

Figure 7: Kernel latencies and storage IO performance comparison.



(a) Impact of determinism on LXC (b) LXC function execution time (c) C/R for different LXC images

Figure 8: Determinism and execution time comparison between the general-purpose and ARNAB kernels.

the *y*-axis represents the cumulative percent. For example, the downtime values at the 80th percentile are 0.742 s, 1.044 s and 1.654 s, respectively. Also, the downtime variation between migration trails of ARNAB with affinity is very small which highlights the improved determinism.

Finally, we evaluate the migration downtime in Fig. 9(b)-(c) for containers running *lighttpd* video streaming server. We employ the proposed *regional access change* orchestration policy to trigger the container migration. The interconnection between the source and destination edge nodes is configured to 1 Gbps. Gigabit Ethernet links are common in campus WiFi deployments and dedicated wide area networks [40] [61]. In Fig. 9(b) the (left) *y*-axis represents the observed downtime while the (right) *y*-axis shows the size of the accumulative checkpoint files in megabytes for the respective container and migration scheme. In Fig. 9(c), we analyze the latency between the user and the edge node while migrating C2 container. The result shows the round-trip time (RTT) obtained by a ping test and the interruption caused by the container migration. Compared to [17], the EFS enhancements can further reduce the downtime by approximately 50%. More importantly, our migration scheme proves to be capable of migrating small containers with very short downtime, making it suitable for IoT systems.

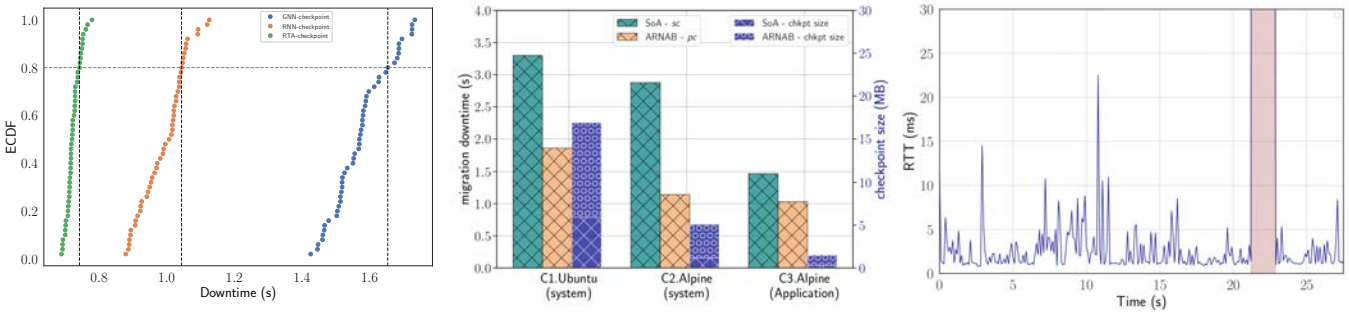
7 ARNAB Use Cases

In this section, we present few use cases that could leverage ARNAB to improve user experience and fulfill real-time application requirements. Here, the presented use cases are not limited to edge-enabled WiFi networks. Edge-enabled cellular networks can also benefit from the support of live container migration across edge networks.

Table 4: Experiment settings.

Experiment		Item	Value
Kernel Latency		generic kernel	4.4.0-116
		real-time kernel	4.4.0-157-rt174
		runtime	5h 33m
		N# of thread	1/CPU
		sleep precision	clock nanosleep
Storage Speed	HDD	Brand/Model	TOSHIBA
		RPM	7200
		Interface	SATA III 6Gb/s
	SSD	Brand	Transcend SSD370
		Flash Type	MLC NAND flash
		Interface	SATA III 6Gb/s
	RAM	Manufacturer	Micron
		Speed	2133 MHz
		Form Factor	DIMM
Container Migration	C1	OS release	Ubuntu 16.04
	C2	OS release	Alpine 3.7
	C3	Image	OCI - Alpine 3.7

- *Augmented Navigation*: Mobile service continuity is critical for augmented reality (AR) based indoor navigation applications. AR applications for public places, such as campuses, shopping malls, museums, airports, and exhibition centers can benefit from ARNAB's double-



(a) Checkpoint with CPU affinity (b) Migration downtimes (c) Latency analysis during migration

Figure 9: CPU affinity and migration downtime analysis.

tier migration to support user mobility. That is, the infrastructure can host containerized application for each user and maintains the state and connectivity as users roam through the network.

- *Cloud Robotics*: There are various cases where cloud robotics can be used to handle tasks in indoor environments such as factories, warehouses and also shopping malls. For example, in shopping malls, robots can be used for safety, operational assistance, goods transport. In addition, robots can be part of the edge networks hosting network functions such as APs in case of big events. Again, an uninterrupted and low latency communication are critical for these applications to work safely and effectively.
- *Connected vehicle*: Vehicle vendors, network operators and service providers are moving towards smart transportation. To enable connected and smart vehicles, massive amount of information, such as position, speed, traffic conditions and road hazards, must be processed and shared among neighboring vehicles on the fly. Edge networks can host applications to provide warnings, driver recommendations and parking lot reservation. These applications require tight collaboration between edge networks and vehicles, low latency and high computational capacity. ARNAB can provide continuity and reduce backhaul latency by instantly relocating connect vehicle applications to the proximity of the users.

8 Conclusion

In this paper, we propose a solution to enable mobile service continuity in large-scale edge-enabled WiFi networks. Although edge networking is a promising technology for providing services at the proximity of users, it does not guarantee continuous service delivery for mobile users. As users move away from the initial serving edge, handoff interruption time and network latency become challenges to maintaining seamless service delivery. In this work, we make three contributions to achieve mobile service continuity. First, we propose a new architecture with double-tier migration scheme to provide transparent mobile service continuity. Then, we identify the root-causes of prolonged container migration downtime. Finally, we enhance the container migration scheme by improving system response-time. Our experimental results show that the ARNAB container migration downtime is 50% shorter than the state-of-the-art.

9 Acknowledgement

This work has been partially funded by the H2020 Europe/Taiwan joint action 5G-DIVE (Grant #859881) and also partially funded by the Ministry of Science and Technology, under the Grant Number MOST 108-2634-F-009-006 - through Pervasive Artificial Intelligence Research (PAIR) Labs, Taiwan.

References

[1] Y.-J. Chen, L.-C. Wang, F.-Y. Lin, and B.-S. P. Lin, "Deterministic quality of service guarantee for dynamic service chaining in software defined networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 991–1002, 2017.

[2] O. I. Abdullaziz, Y.-J. Chen, and L.-C. Wang, "Lightweight authentication mechanism for software defined network using information hiding," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

[3] O. I. Abdullaziz, L.-C. Wang, and Y.-J. Chen, "HiAuth: Hidden authentication for protecting software defined networks," *IEEE Transactions on Network and Service Management*, 2019.

[4] 5G-CORAL H2020 project. [Online]. Available: http://5g-coral.eu/

[5] A. Mishra, M. Shin, and W. Arbaugh, "An empirical analysis of the IEEE 802.11 MAC layer handoff process," *ACM Computer Communication Review*, vol. 33, no. 2, pp. 93–102, 2003.

[6] I. Ramani and S. Savage, "SyncScan: practical fast handoff for 802.11 infrastructure networks," in *24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1, 2005, pp. 675–684.

[7] K. Kwon and C. Lee, "A fast handoff algorithm using intelligent channel scan for IEEE 802.11 WLANs," in *6th IEEE International Conference on advanced Communication Technology.*, vol. 1, 2004, pp. 46–50.

[8] S.-H. Park, H.-S. Kim, C.-S. Park, J.-W. Kim, and S.-J. Ko, "Selective channel scanning for fast handoff in wireless LAN using neighbor graph," in *IFIP International Conference on Personal Wireless Communications*. Springer, 2004, pp. 194–203.

[9] S. Pack, H. Jung, T. Kwon, and Y. Choi, "SNC: a selective neighbor caching scheme for fast handoff in IEEE 802.11 wireless networks," *ACM Mobile Computing and Communications Review*, vol. 9, no. 4, pp. 39–49, 2005.

[10] C.-C. Tseng, K.-H. Chi, M.-D. Hsieh, and H.-H. Chang, "Location-based fast handoff for 802.11 networks," *IEEE Communications letters*, vol. 9, no. 4, pp. 304–306, 2005.

[11] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[12] S. Thananjeyan, C. A. Chan, E. Wong, and A. Nirmalathas, "Deployment and resource distribution of mobile edge hosts based on correlated user mobility," *IEEE Access*, vol. 7, pp. 148–159, 2019.

[13] T. Taleb, A. Ksentini, and P. Frangoudis, "Follow-Me Cloud: When Cloud Services Follow Mobile Users," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[14] W. Nasrin and J. Xie, "SharedMEC: Sharing clouds to support user mobility in mobile edge computing," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[15] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.

[16] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, p. 1488, 2019.

[17] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2018.

[18] O. I. Abdullaziz, L.-C. Wang, S. B. Chundrigar, and K.-L. Huang, "ARNAB: Transparent service continuity across orchestrated edge networks," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–6.

[19] L. Sequeira, J. L. de la Cruz, J. Ruiz-Mas, J. Saldana, J. Fernandez-Navajas, and J. Almodovar, "Building an SDN enterprise WLAN based on virtual APs," *IEEE Communications Letters*, vol. 21, no. 2, pp. 374–377, 2017.

[20] Google. Kubernetes engine - containerized application management at scale. [Online]. Available: https://cloud.google.com/kubernetes-engine/

[21] Amazon. Elastic container service. [Online]. Available: https://aws.amazon.com/ecs/

[22] R. Morabito, "Power consumption of virtualization technologies: an empirical investigation," in *IEEE/ACM 8th International Conference on Utility and Cloud Computing*, 2015, pp. 522–527.

[23] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *in 2015 International Conference on Advances Computer Engineering and Applications (ICACEA)*. IEEE, 2015, pp. 342–346.

[24] R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating performance of containerized IoT services for clustered devices at the network edge," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 1019–1030, 2017.

[25] 5G-CORAL D3.1. Initial design of 5G-CORAL orchestration and control system. [Online]. Available: http://5g-coral.eu/wp-content/uploads/2018/06/D3.19802.pdf

[26] Y. Grunenberger and F. Rousseau, "Virtual access points for transparent mobility in wireless LANs," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2010, pp. 1–6.

[27] M. E. Berezin, F. Rousseau, and A. Duda, "Multichannel virtual access points for seamless handoffs in IEEE 802.11 wireless networks," in *IEEE 73rd Vehicular Technology Conference (VTC Spring)*, 2011, pp. 1–5.

[28] J. Schulz-Zander, P. L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic orchestration of wifi networks," in *USENIX Annual Technical Conference*, 2014, pp. 347–358.

[29] J. Saldana, J. L. de la Cruz, L. Sequeira, J. Fernández-Navajas, and J. Ruiz-Mas, "Can a Wi-Fi WLAN support a first person shooter?" in *Proceedings of the International Workshop on Network and Systems Support for Games*. IEEE Press, 2015, p. 15.

[30] (2015) Wi-5 H2020 project. [Online]. Available: http://www.wi5.eu/

[31] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 377–390, 2002.

[32] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.

[33] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[34] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS operating systems review*, vol. 43, no. 3, pp. 14–26, 2009.

[35] P. Lu, A. Barbalace, and B. Ravindran, "HSG-LM: hybrid-copy speculative guest OS live migration without hypervisor," in *Proceedings of the 6th International Systems and Storage Conference*. ACM, 2013, p. 2.

[36] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 254–265.

[37] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proceedings of the 2013 ACM cloud and autonomic computing conference*. ACM, 2013, p. 11.

[38] V. Medina and J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

[39] M. Forsman, A. Glad, L. Lundberg, and D. Ilie, "Algorithms for automated live migration of virtual machines," *Journal of Systems and Software*, vol. 101, pp. 110–126, 2015.

[40] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, "XvMotion: Unified virtual machine migration over long distance," in *USENIX Annual Technical Conference*, 2014, pp. 97–108.

[41] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: agile VM handoff for edge computing," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 12.

[42] R. AkremAddad, D. L. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Towards a fast service migration in 5G," in *IEEE Conference on Standards for Communications and Networking (CSCN)*, 2018, pp. 1–6.

[43] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, and G. Merlino, "Companion fog computing: Supporting things mobility through container migration at the edge," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2018, pp. 97–105.

[44] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 258–269.

[45] W. Bao, D. Yuan, Z. Yang, S. Wang, W. Li, B. B. Zhou, and A. Y. Zomaya, "Follow me fog: Toward seamless handover timing schemes in a fog computing environment," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 72–78, 2017.

[46] W. Bao, D. Yuan, Z. Yang, S. Wang, B. Zhou, S. Adams, and A. Zomaya, "sfog: Seamless fog computing environment for mobile IoT applications," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 2018, pp. 127–136.

[47] A. Aghdai, Y. Xu, M. Huang, D. H. Dai, and H. J. Chao, "Enabling mobility in LTE-compatible mobile-edge computing with programmable switches," *arXiv preprint arXiv:1905.05258*, 2019.

[48] E. U. T. R. Access, "Requirements for support of radio resource management (3gpp ts 36.133 version 13.3.0 release 13)"," *ETSI TS*, p. V13, 2016.

[49] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[50] F. Fainelli, "The OpenWrt embedded development framework," in *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[51] C. Scordino and G. Lipari, "Linux and real-time: Current approaches and future opportunities," in *ANIPLA International Congress, Rome*, 2006.

[52] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Real-time linux communications: an evaluation of the Linux communication stack for real-time robotic applications," 2018.

[53] Xenomai project. [Online]. Available: https://xenomai.org/

[54] Rtai project. [Online]. Available: https://www.rtai.org/

[55] The RTL collaborative project. [Online]. Available: https://wiki.linuxfoundation.org/realtime/rtl/start

[56] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, "Linux PREEMPT-RT v2. 6.33 versus v3. 6.6: Better or worse for real-time applications?" *ACM SIGBED Review*, vol. 11, no. 1, pp. 26–31, 2014.

[57] V. Kasavajhala, "Solid state drive vs. hard disk drive price and performance study," *Proc. Dell Tech. White Paper*, pp. 8–9, 2011.

[58] P. Snyder, "tmpfs: A virtual memory file system," in *Proceedings of the autumn 1990 EUUG Conference*, 1990, pp. 241–248.

[59] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 15.

[60] Flexible I/O Tester. [Online]. Available: https://github.com/axboe/fio

[61] Aruba Networks. Aruba campus wireless networks. [Online]. Available: https://www.arubanetworks.com/website/vrd/CampusWNetworksVRD/wwhelp/wwhimpl/js/html/wwhelp.htm

Osamah Ibrahiem Abdullaziz received the B.S. and M.Eng.Sc. degrees from Multimedia University, Malaysia in 2011 and 2015, respectively. He is currently a Ph.D. candidate at the department of Electrical and Computer Engineering, National Chiao Tung University, Taiwan and an R&D engineer with Industrial Technology Research Institute (ITRI), Taiwan. His current research interests include software defined networks, multi-access edge computing, network security and network information hiding.

Li-Chun Wang (M'96 – SM'06 – F'11) received received Ph. D. degree from the Georgia Institute of Technology, Atlanta, in 1996. From 1996 to 2000, he was with AT&T Laboratories, where he was a Senior Technical Staff Member in the Wireless Communications Research Department. Since August 2000, he has joined the Department of Electrical and Computer Engineering of National Chiao Tung University in Taiwan and is jointly appointed by Department of Computer Science and Information Engineering of the same university. Dr. Wang was elected to the IEEE Fellow in 2011 for his contributions to cellular architectures and radio resource management in wireless networks. He won the Distinguished Research Award of National Science Council, Taiwan (2012). He was the co-recipients of IEEE Communications Society Asia Pacific Board Best Award (2015), Y. Z. Hsu Scientific Paper Award (2013), and IEEE Jack Neubauer Best Paper Award (1997). His current research interests are in the areas of software-defined mobile networks, heterogeneous networks, and data-driven intelligent wireless communications. He holds 23 US patents, and have published over 300 journal and conference papers, and co-edited a book, "Key Technologies for 5G Wireless Systems," (Cambridge University Press 2017).

Shahzoob Bilal Chundrigar received his B.S. degree in telecommunication engineering from M.U.E.T, Pakistan and his M.S. degree in electrical engineering and computer science from N.C.T.U, Taiwan. In 2012, he worked as O&M engineer for Mobilink. Later he joined 3DResults as a senior ERP consultant. He worked as a research Engineer in ITRI, where he was involved in several 5G projects in collaboration with EU. Currently, he is working as Telecom Manager in Groundhog Technologies. His current research interests include 5G wireless mobile communications, including mobile edge computing, high-speed networks, cloud computing involving software-defined network, and network function virtualization technologies.

Kuei-Li Huang received the Ph.D. degree from National Chiao Tung University, Taiwan, in 2012. Currently, He serves as an engineer in the division of Video & Multimedia Communications Technology in the Industrial Technology Research Institute (ITRI), Taiwan. His research interests include mobile computing, multi-access edge and fog computing and mathematical modeling.