DEGREE PROJECT

APOLLO

A System for Proactive Application Migration in Mobile Cloud Computing

Haidar Chikh

Computer Science and Engineering, masters level
2016

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

APOLLO: A System for Proactive Application Migration in Mobile Cloud Computing

Haidar Chikh

Lulea University of Technology
Dept. of Computer Science, Electrical and Space Engineering

September 2016

Abstract

Demands of modern mobile applications such as those related to remote healthcare and augmented reality put significant pressure on the mobile devices regarding computing and battery requirements. Further, the end users use these applications while roaming in heterogeneous access networks such as WiFi and 4G. One way to fulfil these demands is via application migration in a mobile cloud computing system, i.e., moving the applications from mobile device to the clouds. However, application migration comes with a set of challenges including those related to mobility management, and network and cloud selection. This thesis proposes, develops and validates a system called APOLLO. The proposed system incorporates a Reinforcement learning agent to select the best combination of cloud and network in case of stochastic conditions such as unpredictable network conditions. Using extensive simulations, we validate APOLLO and show that it efficiently supports proactive application migration in a mobile cloud computing system.

Acknowledgement

I would like to thank my supervisor Karan Mitra for the help during all stages of this thesis. His dedication, guidance and valuable insights were an integral part of making this thesis possible. Thanks also goes to my co-supervisor Saguna Saguna for her help and support.

I would also like to thank my friends and colleges, Basel Kikhia, Jean Paul Bambanza, Miguel Castano, Mustafa Dana, Medhat Mohamad, and Sebastian Svensson. A special thank goes to Emanuel Palm and Tambi Ali for their help and support.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. We are far apart, but this accomplishment would not have been possible without you. Thank you, and I hope to see you soon.

Haidar Chikh

Contents

CHAPTER 1

INTRODUCTION

1.1 Introduction

The vast spread of mobile devices has encouraged developing a new class of applications. Applications like augmented reality, virtual reality, patient monitoring, and mobile gaming are increasing at a rapid pace targeting mobile devices. However, these applications drain the limited resources of mobile devices in term of computing, storage, and battery. Today's mobile devices rely on Lithium-ion batteries as power sources and Silicon based ICs as a computing unit. However, Lithium-ion batteries have low energy density and Silicon based ICs generate heat as a side product of computation. There are no breakthroughs in batteries or CPUs technology in the near future. Consequently, we have to utilize remote resources to supply the ever increasing mobile applications with resources.

Cloud Computing is a model for delivering virtualized resources and services over the Internet [1]. To reach the cloud mobile devices utilize Heterogeneous Access Networks (HANs). HANs are Wireless Access Networks which operate by heterogeneous technologies (e.g. WiFI, WiMAX, 4G) each of which is optimal for a particular situation and for specific demands. Mobile Computing model enables mobile devices to send and receive data while roaming across HANs. Mobile Cloud Computing combines the recent adavances in the areas of Cloud Computing, Mobile Computing, and HANs.

Mobile Cloud Computing (MCC) is a computing paradigm that utilizes resources and services of the cloud to overcome the resource scarceness of mobile devices. MCC incorporates cloud computing, mobile computing and HANs to provide mobile devices with virtually unlimited resources [2]. MCC offloads data processing and storage from mobile devices to the cloud, increasing the storage capacity and processing power. In MCC, the mobile device acts as a smart terminal connecting to the cloud over HANs[3]; which eliminates any resource restriction and provisions mobile applications with unlimited resources.

We envision the next generation of mobile devices to be smart terminals that have the ability to (i.) run applications locally or on the cloud, (ii.) live migrate applications between mobile device(s) and cloud(s). By live migrate applications we mean, moving an application from one platform to another during runtime. And (iii.) will provision low latency handoffs between access networks for uninterrupted network connectivity with clouds.

1.2 Research Motivation

The research in this thesis aims at developing a Mobile Cloud Computing (MCC) system to extend the limited resources of mobile devices. Following are two scenarios that illustrate some benefits of an MCC system.

Scenario 1: A person playing a game on his mobile phone, within two hours the phone will be hot, and run out of battery. Now imagine the scenario where the application running on the mobile phone get migrates to a near by cloud. thereby utilizing resources of the cloud and streaming the game to the phone instead of running it on the phone itself . This process of application migration can assist in prolonging the mobile battery life time and enhance user experience.

Scenario 2: Consider, a more complex scenario, the person is playing the same game on his/her mobile phone while roaming in the university campus, but he/she is on the move to university (from home). The phone is connected to the Internet through 4G network, and don't have a "good" access to any cloud (high delay or limited throughput). The game will keep running on the mobile phone, but when he/she reaches to the campus, the phone will detects and connects to campus's WiFI network (now the phone has a "good" access to a cloud) and migrate the game to campus's cloud utilizing the remote resources instead of draining its own.

1.3 Research Challenges

The vision of Mobile Cloud Computing is to offload the computation to the cloud, enabling mobile devices to run several applications locally or on a cloud. Developing a MCC system has three main challenges (i.) network and cloud selection, (ii.) live application migration, and (iii.) mobility management.

1.3.1 Network and Cloud Selection

Network selection is a challenging task; mobile devices use wireless access networks to utilize remote resources, and the characteristics of a network (e.g. delay and through-

put) affect the performance of the applications running on the cloud. However, the performance of wireless access networks is unpredictable. For example, wireless access networks have limited and shared bandwidth, and have an interference prone nature. Further, wireless access networks are heterogeneous in term of technology (e.g. 4G and WiFI), where each technology suits specific demands. Thus network selection is time, location, and demand dependent. And the challenge is, how to select the best access network and cloud to get the best performance of the offloaded computation. The cloud has similar characteristics to wireless access networks. The shared and heterogeneous resources make it unpractical to apply static rules for network and cloud selection. To conclude, the pervasive and heterogeneous nature of networks and clouds dictate the need for dynamic network and cloud selection rules based on time, location, available remote resources, and application requirements.

1.3.2 Application Migration

To extend the resources of mobile devices, the applications need to be migrated from mobile devices to the cloud [3, 2]. However, mobile devices and clouds may use different Operating Systems (OSs) and hardware architectures. Therefore, applications may not be portable across these platforms. This problem is further complicated considering live application migration requirement. Live application migration is moving an application from one platform to another during runtime. In other words, the application keeps its state while migrating among platforms. The complexity is due to the fact that application's state is scattered within the OS. For example, in Input/Output buffers, application's sockets and memory pages. Extracting the state and injecting it back to another OS is not implemented in any standard OS. By standard we mean a off the shelf OS not special purpose OS (e.g. a cluster or datacenter OS). To achieve live application migration we can either build applications from ground up to support it or use an OS level tool to enable today's application to live migrate. Preserving applications state at the OS level will allow us to reuse all of the available applications instead of building special purpose applications which support live application migration.

1.3.3 Mobility Management

Terminal mobility is " The function of allowing a mobile node to change its point of attachment to the network, without interrupting IP packet delivery to/from that node "[4]. Maintaining a connection (i.e. IP packet delivery) while a mobile device is roaming in HANs or the application is live migrating in clouds is challenging. IPv4 was not designed with mobility management in mind. The main issue is that IP protocol couples the identity and location by a single identifier (i.e. IP address). Changing the point of attachment or migrating an application from one place to another must address this challenge.

1.4 Research Contribution

In this thesis, we aim to develop an MCC system that enables application migration while the users roam in HANs. Our proposed system for proactive application migration in mobile cloud computing or APOLLO efficiently selects the best cloud and network based on stochastic conditions such as cloud workloads, network delay and throughout to provision applications while the users are on-the-move.

Our key contributions in this thesis are as follows:

- We propose, develop and validate APOLLO - a system for proactive application migration in mobile cloud computing. APOLLO incorporates crucial functionality such as application migration, network-and-cloud selection, and mobility management;
- We propose, develop, and validate a mechanism for network-and-cloud selection based on Reinforcement Learning. The MN uses this mechanism to proactively select the combination of network-and-cloud for efficient application migration in stochastic network-and-cloud conditions; and
- We propose and validate the usage of Linux containers to facilitate application migration

1.5 Organization of the Thesis

The thesis is organized into five chapters:

- Chapter 2: illustrates MCC's building blocks. It provides an in-depth discussion of Virtualization, Heterogeneous Access Networks, and IP Mobility. Further, it highlights the state of the art in MCC's building blocks.
- Chapter 3: presents the architecture of our APOLLO system. It describes the system's components giving a detailed description of how we approached Network and cloud selection, Application mobility, and IP mobility.
- Chapter 4: validates and tests tow components of APOLLO system. It presents the reinforcement learning implementation and tests. Further, it describes the application migration test bed and tests results.
- Chapter 5: presents our conclusion, lessons learned during this thesis, and finally it lists the future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

Mobile Cloud Computing (MCC) Figure 2.1, is an emerging computing paradigm that aims to utilize resources and services of the cloud to overcome the resource scarceness of Mobile Nodes (MNs). MCC incorporates the cloud computing, mobile computing and HANs to provide MNs with virtually unlimited resources [2]. MCC offloads data processing and storage from MNs to the cloud, increasing the storage capacity and processing power of MNs, in MCC The MN acts as a smart terminal connecting to the cloud over HANs[3]. This chapter illustrates MCC building blocks i.e. Virtualization, Heterogeneous Access Networks, and Mobility. Furthermore, it outlines the state of art technologies in each of the aforementioned domains.

Figure 2.1: Cloud Computing[5].

2.1 The Relation between Hardware and Software

To understand the relation between hardware and software we have to go back in time to 1960s. Looking at computing evolution we see that computing has started with "dumb" terminals sharing a handful of resources, then evolved to self-contained units (PC), and eventually evolved to smart terminals sharing virtually unlimited resources[6]. Early computers were massive and expensive; IBM System/360 Model 25 occupied several rooms and cost 253.000$ in 1968 [7]. Therefore, machine time was expensive, and a handful of computers ever existed (by 1965 there were 20.000 computers in the world) [8]. The solution was to develop time-sharing platforms, where several users use "dumb" terminal devices to access a shared resources (e.g. TSS/360 OS running on System/360 Model 67) [9]. Later on, breakthroughs in semiconductors technology (i.e. size and cost) and the emergence of integrated circuits have paved the way for Personal Computer (PC) era [10].

During the PC era, the paradigm was a PC for every user, and software were built to run in self-contained units [11]. Software built to run in self-contained units was the de facto paradigm for stationarity and semi-stationary devices (i.e. Laptops) due to, i. the scarceness of access networks making it unpractical to access remote resources, and ii. to the fact that you can always get a bigger more powerful device. However, Self-contained units model is not applicable to mobile devices. A Key difference between mobile and stationary devices is resource limitations; mobile devices are resource constrained while stationary devices are not. This difference is due to the combination of three factors, i. mobile devices rely on an inefficient power source (batteries have low energy density), ii. Silicon based CPUs generate heat to compute, and iii. mobile devices have to be portable (i.e. small in size). Silicon based ICs have a direct relation between the generated heat and computing (computation produces heat), this is the key to the limited computational power of mobile devices (small surface area to dissipate the heat).

Today, The situation is the opposite of early computing days. The sheer number of computers (servers), and the widespread of access networks facilitate new solutions, where one user utilize resources of several remote servers. At this point Mobile Cloud Computing (MCC) comes into the picture, by offloading the computation to the cloud, MCC solves the resource scarceness of mobile devices. In the next section we illustrates the basics of Cloud Computing.

2.2 Cloud Computing

Cloud computing is defined according to the National Institute of Standards and Technology (NIST) [1] as " Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned

and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models ".

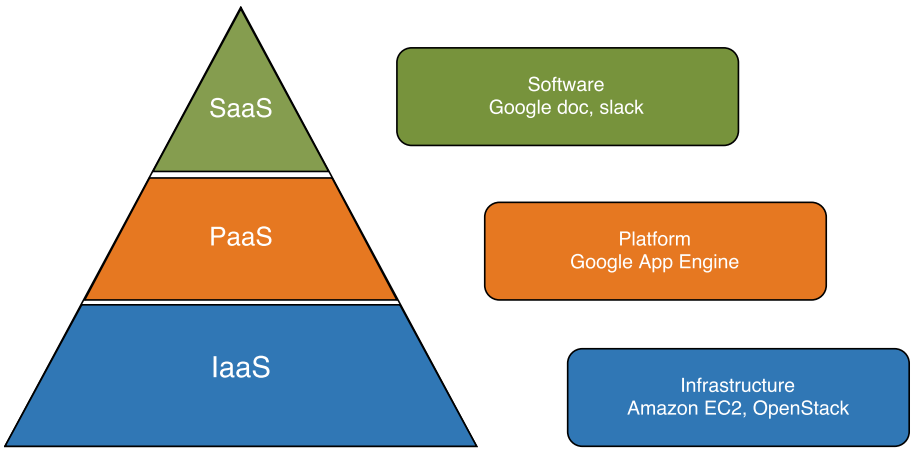


Figure 2.2: Cloud service models.

Essential Characteristics: according to [1], the cloud's five characteristics are

- **On-demand self-service** automatic provisioning of consumers needs such as computation capabilities, storage and network bandwidth without any human intervention.
- ***Broad network access*** the resources must be accessible through the network using standard networking protocols.
- **Resource pooling** resources have to be pooled to serve multiple consumers; the customers should be able to dynamically use and release different resources.
- **Rapid elasticity** resource provisioning should be elastic, and in many cases automatic (e.g. auto scaling)
- **Measured service** the consumers can utilize provider resources based on a pay-per-use paradigm, with the ability to monitor utilized resources status (e.g. CPU utilization)

Service Model: service model figure 2.2 outline providers and consumers responsibilities

- **Infrastructure as a Service (SaaS)** The consumer controls the fundamental computing resources such as storage, processing and some networking elements. For example, a firewall in front of consumer's service, or to install an arbitrary operating system on the hardware.

- **Platform as a Service (PaaS)** In PaaS the degree of abstraction is higher, the provider is responsible for managing the IaaS, while the consumers rule is to deploy their applications on top of the platform.
- **Software as a Service (SaaS)** One more abstraction layer, the provider manages the IaaS and PaaS and deploys an application on top, giving consumers means to access and use the application.

Deployment Models: deployment models outline cloud owners and consumers

- **Private cloud** The cloud is used exclusively by a single organization, multiple consumers. Private Cloud is owned and managed by the organization itself, a third party or a combination of both.
- **Community cloud** The cloud is used exclusively by multiple consumers who share the same interest such as security or availability. Community Cloud is owned and managed by the organization itself, a third party or a combination of both.
- **Public cloud** The cloud is owned, managed and maintained by a provider, and the cloud is public to use for all consumers.
- **Hybrid cloud** any combination of the types above.

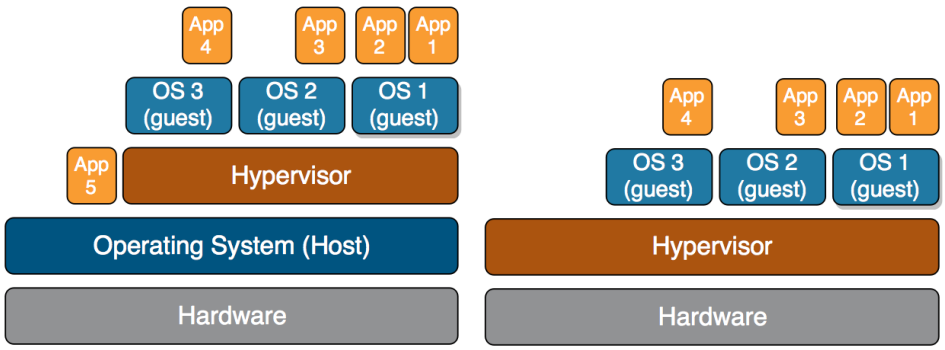
The definition, characteristics and deployment models clearly state the need to access a pool of isolated shared resources, which arise many challenges for the providers and concerns for consumers. On the one hand, the providers have to find a method to manage their resources efficiently, sharing the available physical resources among as many consumers as possible, taking into consideration that each customer gets what he paid for. However, the clients are concerned the most about their data security. The main technology which addresses these challenges is Virtualization.

2.3 Virtualization

Resource Virtualization is; using an additional software layer on top of an underlying system, this extra layer provides abstractions in the form of multiple instances of the underlying systems [12]. Virtualization addresses the challenges of, deploying applications as manageable units, resource management, resource control, multi-tenancy, and security. This is achieved by providing isolated virtual fundamental computing resources. Resource control specifies what resources from provider's pool are accessible by a consumer [13]. Memory and compute power are common criteria for resource control, which assure that a workload is constrained to an exact amount of memory and execution time. However, resource control is a secondary concern comparing to functional and security isolation, where a given two workloads cannot access each others data, or effect execution

correctness [13].

The main two methods to provide isolation are hardware level Virtualization and system level Virtualization, both of which tackle security and resource management challenges. Hardware level Virtualization provides Virtual Machines (VMs); a VM can be considered as a real machine, where we install an Operating System (OS), and install applications on top of the OS. On the other hand, system level Virtualization provides containers, where we install an application in a container; and deploy the container as a self-contained application on a shared operating system [14].



(a) Hosted hypervisor. (b) Bare metal.

Figure 2.3: Hardware Virtualization.

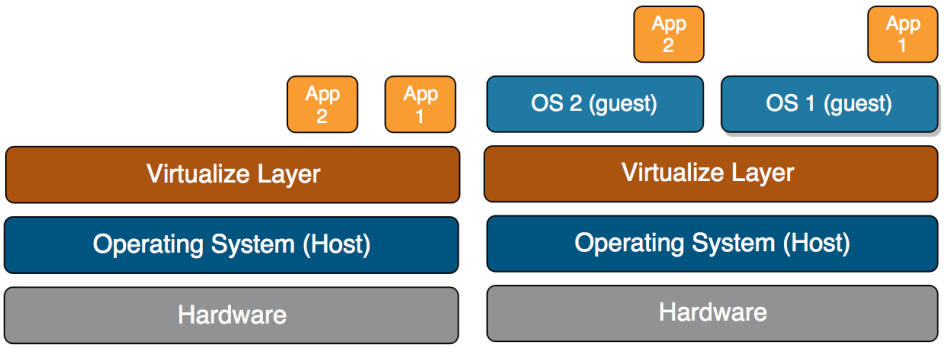
2.3.1 Hardware Level Virtualization

Hardware level Virtualization utilizes hypervisors, figure 2.3 shows two types of hypervisors, bare-metal and hosted. Using hypervisors, the providers can meet the cloud requirement, but using a complete operating system as a unit of deployments is expensive, in term of memory, storage and boot time. This is where system level Virtualization shines [15].

2.3.2 Operating System (OS) Level Virtualization

OS-level Virtualization is a lightweight alternative to hypervisors; it achieves isolation throw introducing virtual instances of the user-space [12]. The services/applications share the same underlying operating system, but each has an isolated view of that operating system. Figure 2.4 shows the difference between hardware level and OS-level Virtualization. In hardware level Virtualization, the unit of abstraction is a hardware (CPU,

memory, NIC, ...), while the unit of abstraction in OS-level Virtualization is an OS [12].



(a) Operating System virtualization. (b) Hardware Virtualization.

Figure 2.4: Comparison between Hardware Virtualization and Operating System virtualization.

2.3.3 OS level Virtualization in Linux

Container-based Virtualization is an implementation of OS level virtualization concept; it is more efficient than VMs in term of resource usage, where all the containers share the same kernel [13]. Container-based Virtualization is implemented using Linux kernel *namespaces* and *Control groups (Cgroups)* features. Linux kernel *namespaces* allows different applications to have an isolated view of the underlying system, it provides new instances of global namespaces (e.g. pid) for each container giving the illusion that the application is running on its own OS [12, 16]. *Cgroups* is used to constrain application usage of the physical resources. Linux Container (LXC) combines *namespaces* and *Cgroups* functionalities to tie up an application and its dependencies into a virtual container. This container can run almost on any Linux distribution, providing a near native Linux environment for the applications [15].

Linux namespaces

One of the main drives behind implementing *namespaces* is to facilitate moving an application with its saved state from one machine and restore it on another machine. Saving the application's state means, saving all of the global resources which the application uses such as PIDs and SYS V IPC identifiers. We cannot assure that, for example, a process on the new machine does not have the same PID of the saved application [16]. *namespaces* solves this dilemma. Currently, there are six different *namespaces* implemented in

Linux kernel, each of them abstracts a global system resource. Processes which belong to an instance of *namespaces* has no view on the global resource of that namespace [17].

- ***PID namespaces*** isolates the PID space. In the simplest form, processes belonging to different PID namespaces can have the same PID number, one of the main advantages of using PID namespaces is that each PID namespaces can have its own. *init* process (PID 1) and we can preserve the processes PID regardless of where the processes is running.
- ***Mount namespaces*** provides an isolated view for processes on the file system hierarchy.
- ***UTS namespaces*** isolates *nodename* and *domainname* where each group of processes (container) can have a different hostname and NIS domain name.
- ***Network namespaces*** provides an isolated networking environment including a separate networking devices, IPs, routing table and so on.
- ***User namespaces*** provides an isolated user and group id namespace.
- ***IPC namespaces*** isolates *System V IPC* and *POSIX*
- ***message queues*** each container gets a separate messaging queue space.

The processes running in one of the aforementioned namespaces have two identifiers; one is global (used by the host kernel), and one is local (used inside the namespace). This allows us to create a process with root privileges inside a container (user ID of 0) while has a restricted outside that container [17].

2.3.4 Linux Containers (LXC)

Linux kernel supports system level Virtualization throw a number of containment features. Linux Container (LXC) is a tool released in kernel 2.6.24 that uses Linux kernel containment features and provides a powerful API to create an isolated run environment for processes. LXC uses *namespaces*, *Cgroups*, *Apparmor*, *SELinux*, *secure computing mode (seccomp)* and *Chroot* to create a container [18, 19].

- ***namespaces*** allows creating an isolated view of the OS view, each process in Linux has six namespaces(net, pid, user, mnt, uts and ipc) and using namespace we create a separate instance for each process from these global namespaces [13, 15].
- ***Cgroups*** control container's usage of system resources, *Cgroups* manages CPU, memory and I/O for each container [15].
- ***Apparmor* and *SELinux*** are used to restrict application's permissions such as network access and write/read permissions [20].

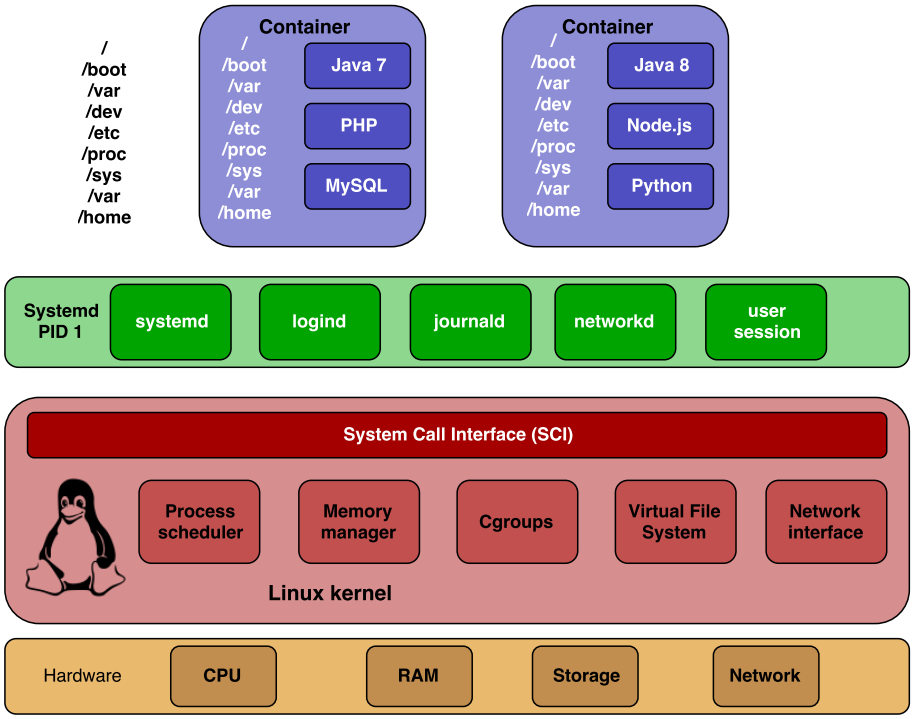


Figure 2.5: LXC in Linux architecture.

- ***seccomp*** is a security feature in Linux kernel, sand-boxing a process using *seccomp* limits the process to 4 system calls *read()*, *write()*, *exit()*, and *sigreturn()* [21].
- ***Chroot*** using *Chroot* we can change the root directory to a certain process and its children, changing the root directory puts a running process in a *jail* restricting its access to any commands or files outside its own root directory[22].

2.3.5 LXD Containers

LXD is a container hypervisor developed to simplify and extend the uses of LXC. LXD has two main components a command line client (LXC) and a system daemon (LXD). The daemon exposes a REST API enabling the command line client to control it locally or over a network[23]. LXD support live container migration by utilizing Checkpoint/Restore In Userspace library (CRIU).

From LXD's documentation [24] the main characteristics of LXD.

- **Running environment**
  - **Architecture** LXD run almost on any architecture that are supported by Linux kernel and Go programming language
  - **Kernel Requirements**

  - Kernel 3.13 and higher
  - *namespaces* (pid, net, uts, ipc and mount)
  - *cGroups* (blkio, cpuset, devices, memory, pids and net_prio)
  - *seccomp*
  - LXC 1.1.5 or higher
  - CRIU for live migration

- **Features:**
  - **Configuration database :** Rather than putting containers' configurations within each container's directory, LXD has a database to store the configurations. This helps to scale easily. For example, if we asked an LXD daemon what container are using eth0 interface, LXD will look up its database and give us the answer quickly. If the configurations are stored in containers' directories, LXD has to iterate throw all of them, load the configurations and look what network interface they use.
  - **Image based :** LXD is image based; containers start their life from an image. Images are kept in a built-in image store, where we can set the images to auto-update from an online store. The built in image-store allows us to publish our own images and make them available for public or private use.
  - **Secure remote communication :** LXD uses HTTPS to communicate with LXD daemon, with minimum TLS 1.2 and 4096 bit RSA.
  - **Clean and crisp API** all the communication between LXD daemon and client (LXC or other) is done using JSON over HTTPS.
  - **Storage backend** LXD support different storage backends for storing containers and images (e.g. plain dirs, Btrfs, ZFS, and LVM).

LXD Live Migration

LXD support live container migration, figure 2.6 two nodes (source and sink). The source node setup the operation and the sink node pull the container. The live migration uses CRIU, using CRIU we can checkpoint a running process, dump its state to a collection of files and restore it later on the same machine or sending the file to another machine and restore it[25]. The migration uses three logical channels [24].

1. Control stream channel: this channel carries information that describes the container, such as profile and configurations. Furthermore, it negotiates protocols used on CRIU and filesystem channels.

2. CRIU images stream channel: carry CRIU images, which hold the container state. Currently, LXD uses stop the world method, in future, iterative, incremental transfer using CRIU *p.haul* protocol is going to be implemented.

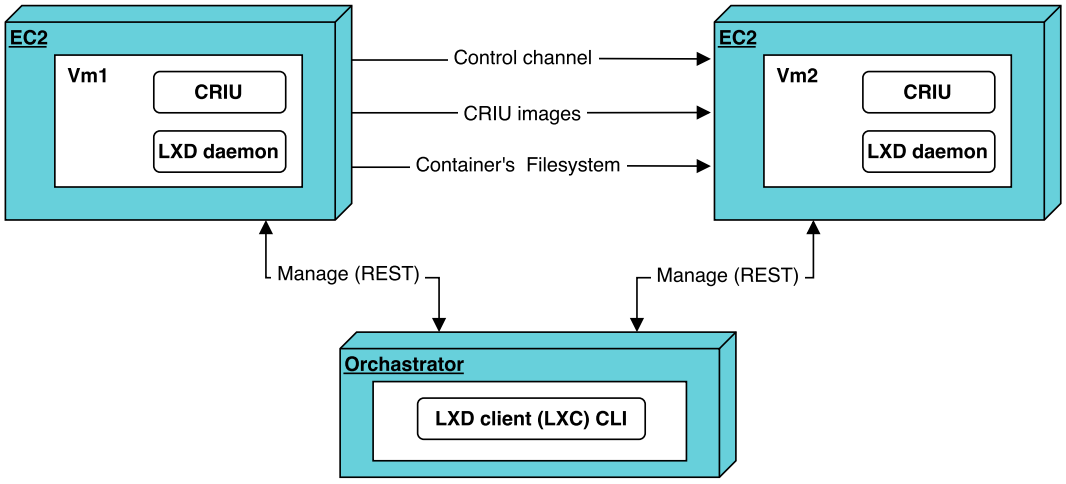


Figure 2.6: LXD live migration using CRIU.

3. Filesystem stream channel: carry container's filesystem, the protocol used depends on negotiation result from the control channel, it will use $LVM, btrfs, ZFS$ if both hosts support it or $rsync$ between incompatible hosts.

2.4 Mobility Management in Heterogeneous Access Networks

Access Networks (ANs) connect a user to a network. Figure 2.7 compare modern wireless access technologies in term of range and bandwidth where each technology has its advantages and drawbacks. Comparing LTE to WiFI, LTE has a longer range, but LTE's interface consumes more power to transfer the same amount data [26, 27]. Furthermore, LTE providers charge per data unit while in most cases WiFI access points are connected to a monthly base charging plan. These difference and more, are the main reason behind the heterogeneity of wireless access networks. Each network performs the best in a particular situation and for specific demands. Thus MNs today are equipped with several wireless interfaces (e.g. WiFI, WiMAX, 3G), The MN uses these interfaces to connect to an IP network. The MN can switch among the interfaces, enabling terminal mobility.

Terminal mobility is defined according to [4] by " The function of allowing a mobile node to change its point of attachment to the network, without interrupting IP packet delivery to/from that node ". To achieve mobility, a MN performs a handoff. Handoff is " The process by which an active MN changes its point of attachment to the network, or when such a change is attempted "[4]. There are two types of handoff, horizontal handoff, and vertical handoff. A horizontal handoff occurs when a mobile node moves from one station to another within the same technology, for example, a MN moves from

one HSDPA base station to another. On the other hand, a vertical handoff occurs when a mobile nodes moves from one technology to another (e.g. WiMAX to HSDPA) [28]. Maintaining a connection (IP packet delivery) to/from MN while the MN roams across HANs (perform Vertical handoff) is challenging, to understand the challenges we have to understand mobility in IP networks.

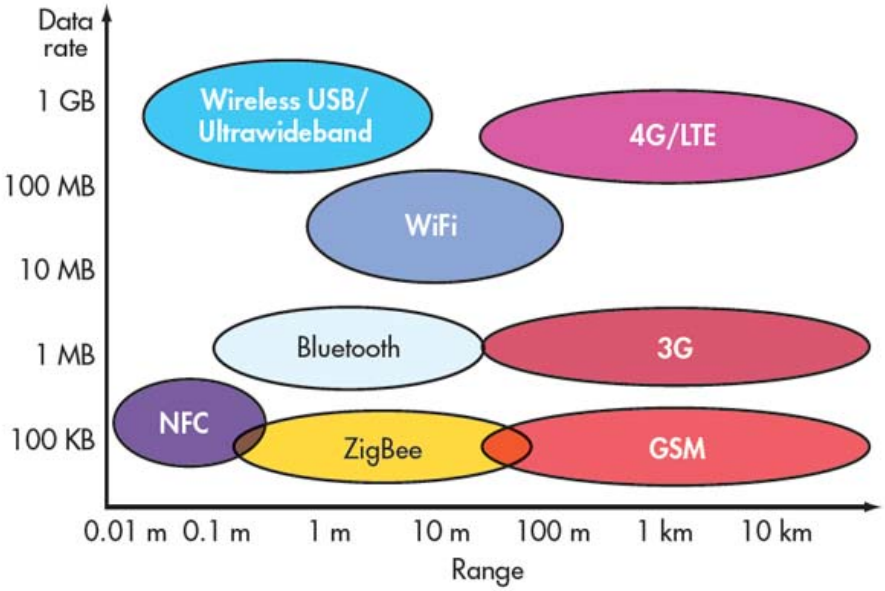


Figure 2.7: Wireless Technologies [29]

.

2.4.1 Mobility in IP Networks

To move data from node A to node B, several subtasks are to be performed. Addresses for A and B have to be set, along with means to map human-readable addresses to machine addresses. A route between A and B must be determined; the data has to be packetized into chunks, and the receiver has to be able to put the data back in order. Both sides have to ensure that data is not corrupted or compromised and to control sending rates not to overwhelm any node across the route with data. These tasks are segregated among protocols arranged in what we know as TCP/IP protocol stack. For example, node-to-node communication is handled by link layer protocols, while global routing and addressing are IP protocol's duty [30]. The fundamental problem in IP mobility is the bind between location (routing related) and identity (authentication related) with one identifier i.e. the IP address. All mobility protocols aim at breaking the bond between location and identity. Each protocol has a slightly different approach providing three

main entities which break this bind [31].

1. **Identifier:** stable identity for a MN.
2. **Locator:** a mean to reach the MN; usually, the locator is an IP address
3. **Mapping:** mapping between the Identifier and Locator.

Furthermore, mobility solutions are implemented in a different layer of the TCP/IP protocol stack; each solution is suitable for a particular application or scenario [28]. To understand IP mobility, we outline some of IP mobility protocols, their applications, and approaches to achieve mobility.

Columbia Protocol

This protocol was one of the early mobility protocols, developed to provide local mobility in Columbia University campus in 1991. Each wireless cell has a Mobile Support Station (MSS), which is the default access router for nodes in that wireless cell. The MN's has a fixed IP derived from a special IP prefix; MSS sends beacons to keeps track of the MN in its wireless cell, MNs reply with a message containing its stable identifier and its old MSS. The new MSS notify the old MSS that a MN has left the old cell[31, 32]. If a corresponding node (CN) sends a packet to a MN, the packet goes to the CN's MSS (MC), If the MC has the MN in its table it will forward the data. Otherwise, MC broadcast the query to all MSS and tunnels the data to the MSS, which has the MN[31, 32].

Virtual Internet Protocol (VIP)

VIP has two main entities, a home network where the mapping occur and two IP addresses for a MN. The MN IPs are a virtual IP address (identifier) and a regular IP address (locator). The Identifier is fixed and can be used to facilitate the use of TCP. This protocol modifies the IP header to carry the two addresses. CN sends the packets with the virtual IP address as locator and identifier, the Home Network revives the packet and forward it to the MN. To reduce triangular routing, the CN replace the locator address with the current locator of the MN after receiving a message from the MN [31].

E2E and mSCTP

E2E and mSCTP are transport layer mobility protocols. E2E protocol gets its name from its End to End (E2E) architecture; this protocol utilizes the DNS service to provide a stable domain for each MN (identifier). A MN obtains an IP address from its current access router and sends update using dynamic DNS to update the mapping between its domain name and IP address [31]. CN query the DNS to get MN IP address; when the session starts, the MN will be responsible for updating its current location to the CN. mobile Stream Transmission Protocol (mSCTP) is similar to E2E where it utilizes

dynamic DNS for mapping and allow both parties to add/delete IP address. mSCTP is defined as SCTP and its ADDIP extension. ADDIP extension enables endpoints to add or remove IP addresses from the SCTP association, and to change the primary IP addresses used by SCTP association [33].

IKEv2 Mobility and Multihoming Protocol (MOBIKE)

MOBIKE is an extension for IKEv2; it supports mobility and multihoming. IKEv2 provides us with an end to end secure tunnel, and MOBIKE extension allows the MN to keep current Security Association (SA) and IKE while moving in an IP network. This protocol allows both parties to have multiple IP addresses. The decision making in MOBIKE is asymmetry, only one peer is responsible for deciding which address to use. Furthermore, MOBIKE supports bidirectional and unidirectional address [34]. Kivinen and Tschofenig show in [35] a scenario where MOBIKE support two party mobility. IKEv2 usage is limited to Virtual Private Network (VPN), the end to end circuit service have not been widely adopted. Yin and Wang in [36] build an application aware IPsec policy system. Furthermore, Kivinen and Tschofenig show in [35] a scenario where MOBIKE support two parties mobility.

Host Identity Protocol (HIP)

This protocol uses a cryptographic public key as an identifier and uses IP address for routing only. The public key is used as the domain name, where a CN can query a Rendezvous Server (RVS) that keeps the mapping between public key and IP address[31].

2.5 Mobile IP (MIP)

MIP is a layer three mobility protocol. Each MN has a Home Agent (HA), the HA resides at MN's home network providing a Home Address (HoA). The MN obtain another IP address from its access router called Care of Address (CoA). The MN sends a Binding Update (BU) to its HA, the BU carry the CoA, and the HA keeps the mapping between the identifier (i.e. HoA) and locator (i.e. CoA) [37]. In other words, the MN is reachable throw a global address (HoA) regardless of its actual location. MIP incorporate many entities [38, 28].

1. Mobile Node (MN): A movable device such as a mobile phone, has the ability to access heterogeneous networks.

2. Home Network (HN): the home network where the HoA resides, correspondent nodes reach the MN throw this global address.

3. Foreign Network(FN): Any network except the HN.

4. Home Agent (HA): Is an application resides in home network's router or a separate device on the home network, the HA maintains a binding registry between HoA and CoA.
5. Foreign Agent (FA): Only for IPv4, Is an application resides in foreign network's router or a separate device, the FA maintains a visitors table tracking the visitors MNs.
6. Correspondent Node (CN): Any network node interested in reaching the MN.

2.5.1 MIP Working Mechanism

When a MN roams in a FN, the MIP mechanism has three main phases enabling MN to be reachable[28]:

Agent Discovery: Home and Foreign Agent (HA, FA) advertises their presence and services using ICMP Router Discovery Protocol (IRDP). The Mobile Node (MN) listen to the agents advertisements and obtain if it is on the home network or a foreign network. The MN can send an agent solicitation to force any agent on the segment to replay with an agent advertisement.Moreover, the foreign agent is designed to accept the solicitation request even if the solicited node has an IP, which does not belong to the same network address. In MIPv6 the MN obtain a CoA using Stateless Auto Configuration.

Registration: The MIP client at MN is configured with a shared key and the IP address of its home agent. The MN form a MIP registration request, add it to its pending list and send it to its home agent through the FA. The FA checks if the request is valid, adds it to its pending list and send it to the HA. The HA checks if the registration request is valid, creates a mobility binding, a routing entry for forwarding packets to the home address, a tunnel to CoA and sends a registration reply to the MN through the FA. The FA checks if the registration request is valid and exists in its pending list then it adds the mobile node to its visitor list, creates a routing entry for forwarding packets to the home address, creates a tunnel to the home agent and forward the registration reply to the mobile node. The mobile node checks if the registration reply is valid and exists in its pending list and sends all the packets to the FA. In MIPv6 there is no FA, the previous steps are used to register, except the one related to the FA.

Tunneling: Data addressed to the mobile node are routed to the home network where the HA intercept and route them through the tunnel to the MN.

2.5.2 MIP Extensions

MIP has many extension and enhancements which tackle some of its disadvantages like triangle routing and improve the handoff time. Following is an outline of the most

important extensions.

- Hierarchical Mobile IPv6 Mobility Management (HMIPv6): HMIPv6 is an extension for the MIP, where a Hierarchical mobility management is added to improve local mobility. HMIPv6 adds a new mobility entity called Mobility Anchor Point (MAP), the MAP acts as a local home agent providing mobility within a local subnet. HMIP decreases the number of BU to the HA since MAP handles local mobility [39].

- Multi-homed Mobile IP (M-MIP): M-MIP supports MIP soft handoff, where the MN is connected to several networks simultaneously. On the other hand, hard handoff; the MN disconnects from a network and then connects to the new network. The MN gets a different CoA on each interface. M-MIP enables the MN to send/receive data using both of its CoA without the need to do a hard handoff [28, 40].

- Route Optimization in MIP: The base MIP enables the MN to move while keeping the connection to CNs. The connectivity is maintained by using the HA as an anchor point; all the data is tunneled through the HA, which is called triangle routing. Route Optimization is to have a direct connection between the MN and CNs. To achieve Route Optimization, a caching capability is added to all nodes (including CN); allowing nodes to cache BUs [41].

- Proxy Mobile IP (PMIP): This protocol support mobility using the network itself, the mobility support is transparent to the MN. PIMP introduces two mobility management nodes, Local Mobility Anchor (LMA) and Mobility Access Gateway (MAG). LMA acts as a home agent and assigns home network prefix to MNs. This prefix is used as the MN identifier, the MAG monitor the location of the MN within the PMIP domain, and sends binding updates to LMA[42].

2.6 M2C2 System

Ubiquitous constrained mobile devices, heterogeneous access networks (HANs) and demands of modern mobile applications in term of computation and storage are the main factors behind developing M2C2 system [43]. The system is shown in figure 2.8 addresses mobility by implementing Multi-homed Mobile IP(M-MIP) and addresses the constrained nature of mobile devices by offloading the computation to Mobile Computing Cloud (MCC). M-MIP works by hiding the Mobile Node (MN) behind an Anchor Point (AP) giving the MN the ability to handoff between different networks without breaking its session with the outside world (i.e. used cloud). Selecting the best network to reach the AP and offloading the computation to the cloud, is achieved by cooperation of several entities. The system utilizes M-MIP to passive probe and chooses the best network (MN-AP), Cloud Probing Service (CPS) to probe the available clouds and, Cloud

Ranking Service (CRS) to use the information gathered by CPS and rank the available clouds[43].

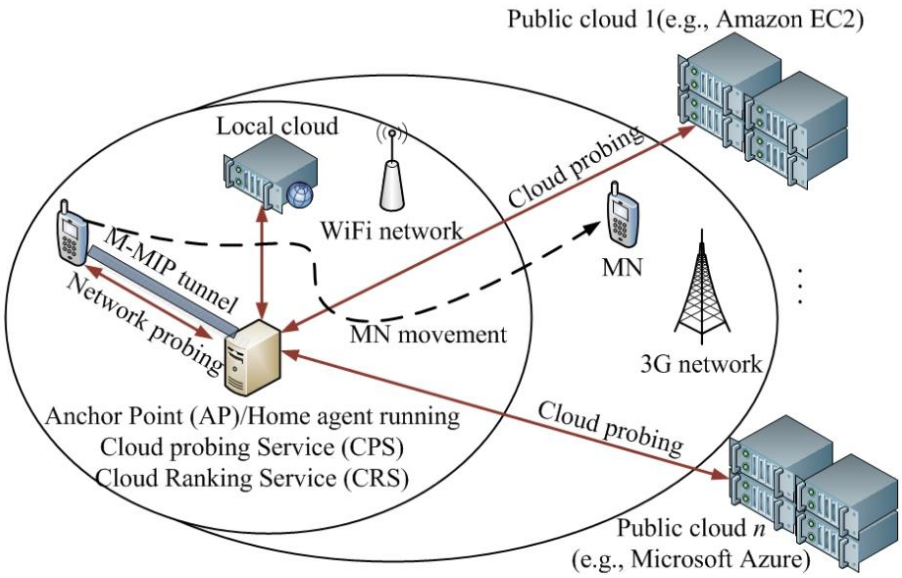


Figure 2.8: M^2C^2: A mobility management system [43].

2.7 Reinforcement Learning

To understand reinforcement learning we have to follow the chain of ideas which have lead to reinforcement learning. In essence, it is our need for mathematical models to predict physical phenomenon. Predicting a physical phenomenon is an easy task when we want to predict a square's area, raise the square's side length to the power of two, and congratulation you got the area ($A = l^2$). For the same side length, the area of a square is always going to be the same. Because, the square's area function is a deterministic function, where giving the same initial state we can predict with absolute certainty the outcome state (value). However, the physical world is not simple or kind; we cannot gather all the needed information to model everything using deterministic functions.

We can predict the outcome of a coin flip with certainty, if and only if we take in consideration a huge number of variables. The coin's original balance, air temperature, humidity, the flipping force, the exact gravitational forces and so on; this is unpractical and complicated. Here the probability theory comes for rescue; we use a probability

function which says, it is a 50% chance to get head or tail. This simple idea allows us to model complicated physical phenomena by a simple mathematical model. It is a trade-off between accuracy and complexity. We notate the expected outcome of a coin flip with a variable. And since we cannot predict the variable's value, we call it a random (stochastic) variable. Each random variable has a result space which is referred to as a distribution. In the case of a coin flip is a Binomial distribution $\{head, tail\}$. In case of a humans' height is a Gaussian distribution over the range $\{shortesthuman, longesthuman\}$.

The next step is to understand the stochastic process, which is a collection of random variables used to model a system. An example of a stochastic process is Markov chain. Markov chain has a state space and moves through this space according to a transition matrix. Markov chain is used to model systems where system's evolution can take more than one way. A distinct feature of Markov chain is Markov property where the probability distribution of the next state depends on the current state only.

We are almost there now, just Markov Decision Process (MDP). If we model a system using Markov chain, and we have the ability to make decisions, this means we have the power to change the transition matrix. The system's evolution is affected by our choices and system's nature (context) itself. We use (MDP) to model such a system. The core problem of MDP is to transfer MDP into a Markov chain, in other words, to figure out the transition matrix. From the transition matrix, we create policies which maximize the overall reward. The policy is a deterministic function maps states to actions $\pi(S \rightarrow A)$.

MDP has a state space S which include all the possible states of the system. In each $s \in S$ the decision maker has a set of possible actions $a \in A$ to take. The process moves from $s \rightarrow \grave{s}$ and gives the decision make a reward $R_a(s, \grave{s})$. The probability of moving from state $s \rightarrow \grave{s}$ is giving by the $P_a(s, \grave{s})$.

If we do not have the transition matrix or the reward function of the Markov decision process, it becomes a reinforcement learning problem. Reinforcement learning is defined according to [44] " is learning what to do, how to map situation to action so as to maximize a numerical reward signal ". In reinforcement learning we have an agent and an environment, the agent evaluates its action (delayed rewards) instead of taken the right action. We can identify three main elements in reinforcement learning Policy, Reward Function and Value Function [44].

Policy: We can compare policies to stimulus-response in psychology, for each state the agent receives from the environment, the agent response with an appropriate action.

Reward Function: In biology, actions are rewarded by pain, pleasure, and stress. We map the same concept to reinforcement learning. A numerical value rewards Agent's ac-

tions; the reward function calculates this value. The agent seeks to maximize the reward value [44].

Value Function: While reward function evaluates each action immediately, value function specifies what is good or bad in the long run. Value function, represent the total amount of rewards the agents is expected to gain starting from a specific state and acting optimally [44]. One of the simple most accurate analogies to explain the value function is writing, while it is frustrating and time-consuming to put your thoughts on paper, the long-term reward is the clarity you get after writing.

2.8 Summary

This chapter illustrated Mobile Cloud Computing (MCC) and its building blocks. It presented an in-depth view of Virtualization, Heterogeneous Access Networks (HANs), and IP mobility. Further, it presented and discussed the challenges of network and cloud selection, IP mobility management, and application mobility. Furthermore, It outlined the state of art technologies in MCC's building blocks (i.e. Virtualization, HANs, and IP mobility).

CHAPTER 3

APOLLO: A SYSTEM FOR PROACTIVE APPLICATION MIGRATION IN MCC

In this chapter, we present the architecture of our APOLLO system. APOLLO is motivated by M^2C^2 system [43]. M^2C^2 system utilizes M-MIP, Cloud Probing Service (CPS) and Cloud Ranking Service (CRS) to achieve two goals, (i) mobility in HANs and (ii) offloading computation to the cloud. However, M^2C^2 does not consider application migration and the challenge of learning from stochastic network-and-cloud conditions. Our proposed system APOLLO incorporates three components to overcome the resource constrained nature of mobile devices and to overcome the limitations of M^2C^2. These components are: (i) a reinforcement learning agent for $Network - Cloud$ selection, (ii) Linux containers to mobilize applications, and (iii) route optimized M-MIP for IP mobility management. Figure 3.1 shows the architecture of APOLLO. In the next section, we present APOLLO's architecture.

3.1 Architecture

The system's architecture is shown in Figure 3.1. The system main components are:

1. Mobile Node (MN): The Mobile Node holds the $Network - Cloud$ selection agent. The agent will decide to run applications on the MN itself or a cloud. When a user starts an application. The agent will live-migrate the application if the platform running the application does not satisfy the application requirements anymore.

2. Container as a Service (CaaS): We consider public and local clouds that can run Linux container as Container as a Service (CaaS), the agent will offload the containers to the cloud based on applications demands.

3. M-MIP: This protocol is used for IP mobility; the Home Agent is used for authentication, and BU/BA messages are used for passive cloud probing.

In the next sections we describe in details how APOLLO handles IP Mobility, Application Migration, and network-and-cloud selection.

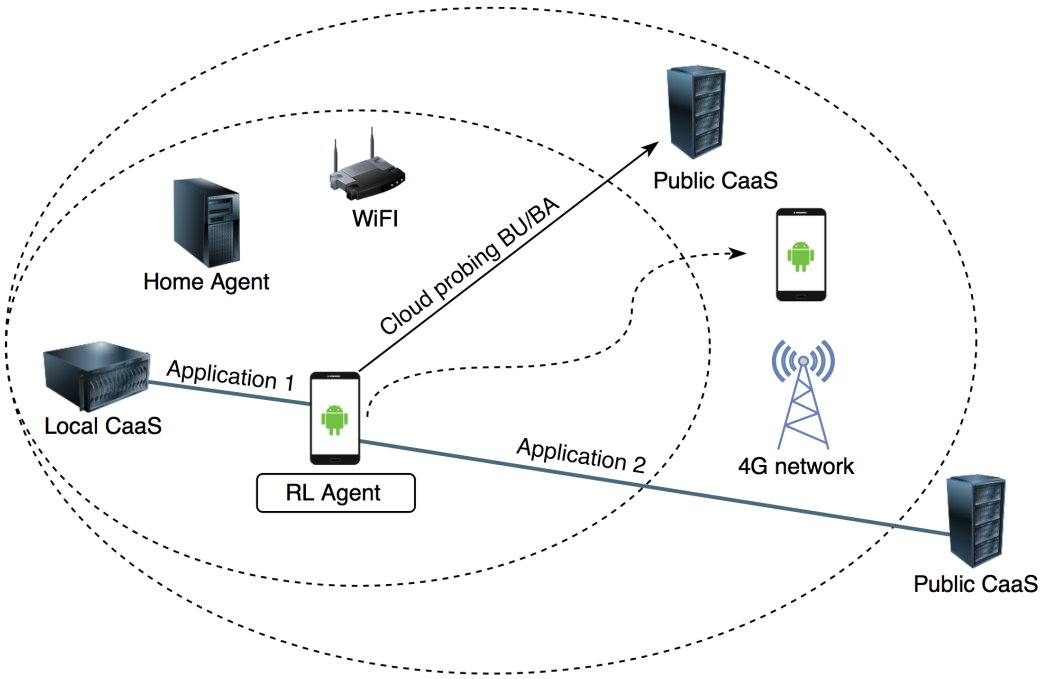


Figure 3.1: APOLLO's architecture.

3.1.1 IP Mobility

Ch2 survey some of IP mobility protocols, a fundamental difference among those protocols is; the TCP/IP implantation layer. Network layer mobility protocols exploit the thin waist of the TCP/IP protocol stack; it is the strategic place where a mobility protocol can serve every higher layer. However, this advantage comes at a price, for example, Mobile IP (MIP) has triangle routing, and is network depended i.e. Multihomed Mobile IP M-MIP must have a HA [30]. The research community has addressed the aforementioned issues, Perkins and Johnson in [41] introduce route optimization scheme for MIP. Moreover, HIMIPv6 uses Local Mobility Anchor (MAP) to improve local mobility and reduce HA's overhead [39]. In APOLLO, we propose using M-MIP, this choice was built on the following criteria:

- Multi-homing: Ahlund and Zaslavsky in [40] describe Multihomed Mobile IP, facilitating soft-handoffs among MN's interfaces.

- Authentication and network probing: The home agent (HA) handles the MN's mobility and authenticates the MN. The binding update (BU) and binding acknowledgment (BA) packets sent between the HA and MN can be used as probe packets to probe the networks. However, for end-to-end probing, we used similar BU/BA message pairs between the correspondent node (CN) and the MN as described by Perkins and Johnson in [41].

3.1.2 Application Migration

Today's mobile device are self contained units, where applications use the device resources to compute. This traditional device-application association drains the limited resources of mobile devices. Live migration is moving an application from one platform to another during runtime. Live-migration eliminates the constraints associated with mobile devices in term of computation and storage [45, 46]. However, migrating an application from one platform to another is not an easy task. As discussed in Ch 2, the common method to migrate applications is to build a Virtual Machine (VM), setup the application, and move the entire VM to another host. This comes at the expense of performance (i.e. Virtualization overhead) and size (a whole operating system)[13]. On the other hand, Linux Containers provides a minimal overhead in term of performance and size.

The vision is to use containerized applications to break the device-application association. Containers provide a standard application deployment unit, where containers can run on a MN, local cloud or public cloud. To illustrate how containers are used we give a practical example i.e. the agent's test during this work. The tests were done using the developer laptop, the test is CPU intensive and the average time for each test was 4 hours. Using the proposed system, we would have been able to setup the test, migrate the container to a CaaS, run the test and migrate the container back to the developer's laptop. APOLLO uses LXD containers, and this choice was based on the following criteria

- **Security:** Some container Implementations such as Docker has the option to run privileged containers, and this is a security risk. Any code running in a privileged container can act as root outside the container boundaries. The operator must pay special attention and fin tune each container, run in "non-privileged" and use security solution such as SELinux or AppArmor [47]. Limiting the container privileges will restrict the number of applications which can be containerized. On the other hand, LXD containers act as a hypervisor, running a privileged container inside a LXD container is like running an application inside a Virtual Machine[23].
- **Live migration:** Decoupling the device-application association will provision unlimited on-demand resources to the MN.
- **Compatibility:** LXD containers can run many other container management applications (e.g. Docker, OpenVZ, Rocket) facilitating code reuse.

3.1.3 Network and Cloud selection

So far we have a Mobile Node (MN) and a mobile application, but we do not have a decision mechanism to decide where to run the application and which network to be used by the MN. There is many decision methods (e.g. static rules, offline learning, online learning). We decided to use a Reinforcement Learning Agent, and this decision was built on the following criteria:

- **Proactive:** Using a decision mechanism that produces a custom policy for each user will allow proactive migrating of applications and handoff among networks. Static rules already have a significant error margin (one policy for all users), and this error margin will get bigger with proactive decisions.
- **Custom policies:** Wireless networks are unstable, and their performance is hard to predict, the decision mechanism must produce custom policies for each user.
- **Scalable :** The decision mechanism will face the challenge of choosing the best network for the MN and the best platform to run the application (e.g. on the MN itself, Cloud 1, Cloud 2 ....). The decision complexity is $M \cdot C$. Where M is the number of networks and C is the number of clouds.

Using reinforcement learning agent is a start, but what is a "good" or a "bad" network-and-cloud? What are the variables the agent should look for? We evaluate each network-and-cloud based on **(i)** End to End delay $MN-Cloud$ and **(ii)** throughput $MN-Cloud$. Delay and throughput are not the only variables to consider in our scenario, but this decision is due the following reasons

- **Thesis scope:** Considering variables such as service cost, handoff cost, and battery status are beyond the scope of this work.
- **Relevance to our scenario:** An IP network has numerous factors to measure its performance, packet duplication, packet loss, packet reordering, jitter, and delay. However, considering factors such as packet reordering are relevant to gauge the performance of a routing protocol not to our scenario.
- **Delay:** End-to-End delay is the time between the execution of an action and the end user perceive the result. Using End to End delay $MN - Cloud$ network-and-cloud state can be estimated. A high delay value is due to one of two reasons; the used network is congested, or the used cloud is overloaded [48]. In both cases, we get a good estimation of the network-and-cloud state, and we know that we should switch/keep the $Network - Cloud$.
- **Throughput:** Delay by its own may not enough to decide where to run a an application. Combining delay and throughput will allow us to run each application on a platform that suits its specific needs [49].

3.2 Network and Cloud Selection Agent

In the next sections we present details of the agent's design. Starting with the model formulation and *reward function*. Then we propose Explore and Learning functions to improve the agent's learning speed.

3.2.1 The Model Formulation

As we discussed in ch2; a Reinforcement learning problem has four components, time epochs, state space, action space, and a reward function. We use notation described in [50, 51] The agent's design is shown in figure 3.2. The agent observes the environment, learns from it and takes an action. Each action resolves in a reward which indicates how "good" or "bad" the agent's decision was.

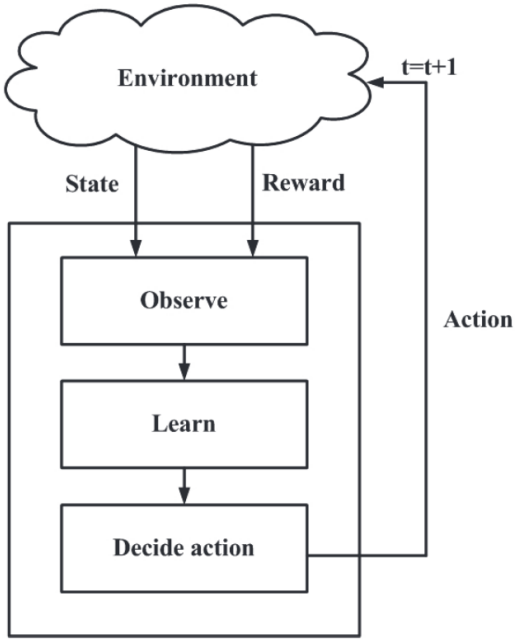


Figure 3.2: Q-learning agent design [28, 44].

Time epochs

The MN acquires information about the available network-and-cloud and takes a decision at each time epoch. The sequence $T = \{1, 2, 3, ..., N\}$ represents moments in time where the agent interacts with the environment. N is a random variable denotes connection termination time. Equation 3.1 denotes the time epochs.

$$T = \{1, 2, ..., N\} \tag{3.1}$$

T Time epochs.
N Termination time.

Action space

The agent interact with the environment to acquire information then take a decision, each interaction is called a time epoch. At each time epoch, the agent has to make a decision, whether to keep using the current network-and-cloud or not. The action space is shown in equation 3.2.

$$A = \{1, 2, ..., M\} \times \{1, 2, ..., C\} \tag{3.2}$$

A Action space
M Available HANs
C Available clouds

State space

MDP state space is represented by flat or factored state representation. Flat state representation simply gives each state an identity; it is a simple representation of a small state space. Factored state representation uses a number of variables to represent the state; it is more efficient in solving problems with a larger state space [52]. We represent our state space S using factored state representation. For each $s \in S$, the state includes the following information. End-to-end delay D and throughput TH for network-and-cloud combinations. Equation 3.3 denotes the state space.

The state space values are quantized into multiples of a unit n, this way we decrease the state space size and overcome the continuous nature of delay and bandwidth [53]. For example, if the delay MN-WiFI-Cloud1 = 11 ms, we quantize it to 20 ms, 18 ms to 20 ms and so on.

$$S = \{1, 2, ..., M\} \times \{1, 2, ..., C\} \times D^{MC} \times TH^{MC} \tag{3.3}$$

S State space.
M Available HANs.
C Available clouds.
D^{MC} Delay ($MN - Cloud$).
TH^{MC} Throughput ($MN - Cloud$).

3.2.2 Reward function

The random variables Y_t ,X_t denote the decision and state of the agent at time epoch t. The *reward function* $r(Y_t, X_t)$ reflects the state of network-and-cloud within time interval $(t, t+1)$. The proposed reward function consists of three sub reward functions. Given $s \in S$, $a \in A$ we donate $f_d(s,a)$, $f_{th}(s,a)$ and $f_g(s,a)$as *delay reward function*, *throughput reward function* and *general reward function* respectively. ω_d, ω_{th}, ω_g is the *weight* giving to the delay, throughput and the general of the state. where $\omega_x \in \{0,1\}$. Delay and throughput reward functions are related to the preference of the user and application requirements. For example, a Voice Over IP application is affected the most by delay and throughput a secondary factor, for such an application we set ω_d higher than ω_{th}. The third reward function $f_g(s,a)$ Eq. 3.6 is derived from Eq. 3.4 and Eq. 3.5 by substituting L_D and L_{TH} with zero, U_D and U_{th} with U_G. In the case, that none of the available networks and clouds meet the application's requirements, this function is used to choose the least worse network-and-cloud. Table 3.1 describe the used notation in $reward - function$

$$f_d(s,a) = \begin{cases} 1, & 0 < d_a < L_D \\ (U_D - d_a) \,/\, (U_D - L_D), & L_D < d_a < U_D \\ 0, & d_a \geq U_D \end{cases} \tag{3.4}$$

$$f_{th}(s,a) = \begin{cases} 1, & th_a \geq U_{TH} \\ 1 - (U_{TH} - th_a)/(U_{TH} - L_{TH}) & L_{TH} < th_a < U_{TH} \\ 0, & th_a \leq L_{TH} \end{cases} \tag{3.5}$$

$$f_g(s,a) = (th_a - d_a + U_G)/(U_G) \tag{3.6}$$

$$r(s,a) = \omega_d f_d(s,a) + \omega_{th} f_{th}(s,a) + \omega_g f_g(s,a) \tag{3.7}$$

3.2.3 Q-learning

The Q-learning algorithm is shown in 3.8. The equation has two variable which alters the behavior of the agent, Learning Rate(α) and Discount Factor (γ).

$$Q(s,a) \leftarrow Q(s,a) + \alpha \; [\; r(s,a,s^{t+1}) + \gamma \; max_{a^{t+1}} Q(s^{t+1}, a^{t+1}) - Q(s,a) \;] \tag{3.8}$$

Learning Rate

The learning rate $(\alpha) \in \{0,1\}$ effect the amount of learning the agent get from each state-action. To reach converges in a stochastic environment the learning rate has to be decreased over time [54], the convergence in this context means that during an episode the maximum change in the Q-matrix is 0. We implemented a function which returns a value tied to each state individually. The core concept of the function is, the agent will

Table 3.1: *reward function*'s notation

Notation	Description
$f_d(s,a)$	*delay reward function* for action a in state s
U_D	Maximum accepted delay by an application
L_D	Minimum accepted delay by an application
d_a	Delay value at the used $Netwok - Cloud$
$f_{th}(s,a)$	*throughput reward function* for action a in state s
U_{TH}	Maximum accepted throughput by an application
L_{TH}	Minimum accepted throughput by an application
th_a	Throughput value at the used network-and-cloud
$f_g(s,a)$	*general reward function* for action a in state s
U_G	An arbitrary value, usually $U_D > 10^4$
th_a	Throughput value at the used network-and-cloud
d_a	Delay value at the used $Netwok - Cloud$
$r(a,s)$	*reward function*
ω_d	Delay weight
ω_{th}	Throughput weight
ω_g	Throughput weight

learn a lot from the first visit to a particular state-action ($\alpha_{(s_1,a_1)^1} = 1$) and this value will decay each time the agent visit the same state-action ($\alpha_{(s_1,a_1)^2} = 0.98$) . Equation 3.9 shows the discount factor function.

$$\alpha_{(s,a)^x} = 1 - \{\tfrac{1}{2}[1 + \mathrm{erf}(\tfrac{(x-\mu)}{(\sigma\sqrt{2})}]\} \tag{3.9}$$

s s state $s \in S$
a a action $a \in A$
x occurrence number of this state-action

Explore function

One of the distinct features of Q-learning is the ability to improve agent's perception of the world regardless its action's quality. This ability is due to the separation between actions' reward and the Q-matrix, which represents its knowledge. In other words, the agent can learn the optimal policy regardless of its actions quality. In a traditional implementation of Q-learning agent, the agent has two separate modes namely exploration mode and exploitation mode. In exploration mode, the agent takes random actions, evaluate this action and update its Q-matrix. Exploration mode aims to increase knowledge of the agent. On the other hand, in Exploitation mode, the agent takes actions based on its Q-matrix (its knowledge) to increase the sum of cumulative reward.

This clear separation between exploration and exploitation adds complexity to real world usage of agents. How long should we explore? When should we explore again and for how long? These questions are hard to answer, and the answer depends on the each environment the agent lives in.

To solve this problem we have to do two things. First, we combine exploration and exploitation mode in on mode. We weight exploration versus exploitation by using a value $\epsilon \in \{0, 1\}$. Setting ϵ to 0.2 will make the agent explore 20% and exploit 80%. Second, we have to improve the exploration part, instated of taking a random action $a \in A$ from a giving state $s \in S$; we track all the previously taking $state - action$ and explore a new action instead of the possibility of trying to learn from the same $state - action$. Algorithm 1 shows the exploration function.

Algorithm 1 Explore function

```
1:  procedure getAction(State, Epsilon)
2:      qValues[] ← agent.getQ(State)
3:      roll ← random.nextDouble()
4:      if roll < Epsilon then
5:          for int i = 0 to qValues.size() do
6:              action ← qValues.getRandomAction()
7:              if action.occurance() < constantValue then
8:                  return action
9:              end if
10:         end for
11:     else
12:         action ← qValues.getMaxQ().getAction()
13:     end if
14: end procedure
```

3.3 Summary

This chapter presented APOLLO system. APOLLO addresses three challenges; IP Mobility, Application Migration and network-and-cloud selection. We proposed solutions to these challenges by utilizing M-MIP, Linux containers, and Reinforcement Learning Agent respectively. Furthermore, we proposed using Explore and Learning Rate functions to shorten the agent's learning time.

CHAPTER 4

IMPLEMENTATION AND EVALUATION

This chapter validates APOLLO system. APOLLO three main functionalities i. Application Migration, ii. network-and-cloud selection agent, and iii. Mobility management. In this chapter, we implemented and tested two separate components; the first component addresses application migration by setting up and testing an application migration environment. The second component addresses network-and-cloud selection by implementing a network-and-cloud selection agent.

4.1 Applications Migration

As discussed in ch 3, we proposed the use of LXD containers to facilitate application migration. LXD live-migration is still under development. Furthermore, the feature lacks documentation and experimentation. Therefore we did some tests, we set up a test bed using Amazon m4.xlarge EC2 instances. The test aims at building a service, containerizing it, and live-migrating the container to calculate the downtime of the service. The test bed consists of eight instances, two in Ireland, two in Virginia, two in Mumbai and two in Sydney. All the instances run Ubuntu 16.04 with a custom Linux 4.5 kernel compiled to support Checkpoint/Restore In Userspace (CRIU)[1]. The container is built from Ubuntu 14.04 image, it runs a Java web server (the service), and the container's size is 231 MB (Ubuntu 14.04 + Java 7 + the service). The web server has a counter which can be acquired by sending a HTTP get request to $< IP > /counter$. This setup allows us to calculate service downtime and to assure that the container is keeping the state i.e. the counter resumes counting after the migration. Apache Jmeter [55] is used to send HTTP get requests to the web server. Table 4.1 shows the test results of executing 49 application migrations among Amazon's data centers.

[1]CRIU is a library that enable check point/restore of Linux processes.

Table 4.1: LXD live migration test.

Zone	Average migration time	Average down time
Ireland - Virginia	32 sec	9 sec
Ireland - Mumbai	58 sec	13.6 sec
Ireland - Sydney	99 sec	24.5 sec
Ireland - Ireland	10 sec	2.8 sec
Virginia - Virginia	11.5 sec	2.8 sec
Mumbai - Mumbai	11.4 sec	2.6 sec
Sydney - Sydney	16.4 sec	3.8 sec

4.2 Network and Cloud Selection

To solve and validate the Reinforcement Learning problem proposed in 3.2. We implemented a data generator and a Reinforcement Learning Agent. Due to time restrictions, we did not collect real cloud statistics. Instead, we utilized the data generator to test the Reinforcement Learning Agent's implementation by simulating a cloud environment.

4.2.1 Data Generator

We used a Pareto-distributed data to simulate end to end delay [56], and Gaussian-distributed data to estimate the bandwidth [57]. The Colt library [58] and Apache Common Math library [59] are utilized to get a random variable with a Gaussian distribution and a random variable with a Pareto distribution respectively. The abstract class *SampleGenerator* class provides a shared code among distributions. *GaussianGenerator* and *ParetoGenerator* classes extend *SampleGenerator* and provide distribution specific parameters. To validate the libraries correctness, Fig. 4.1 shows 10^6 samples from a Gaussian and Pareto variables.

Both distributions (i.e. Gaussian and Pareto) are continuous distributions, To utilize the data generated from them, a rounding function shown in code block 4.1 is implemented. For debugging simplicity the function rounds up the values instead of rounding to the nearest value (i.e. rounding up and down). Furthermore, the rounding action satisfies state quantization proposed in 3.2.1. After implementing a data source, we implemented a *DataGenerator* which provides us with methods to setup a data generating plane and feed the data to our agent. Code block 4.2 shows an example of Pareto data generating plan.

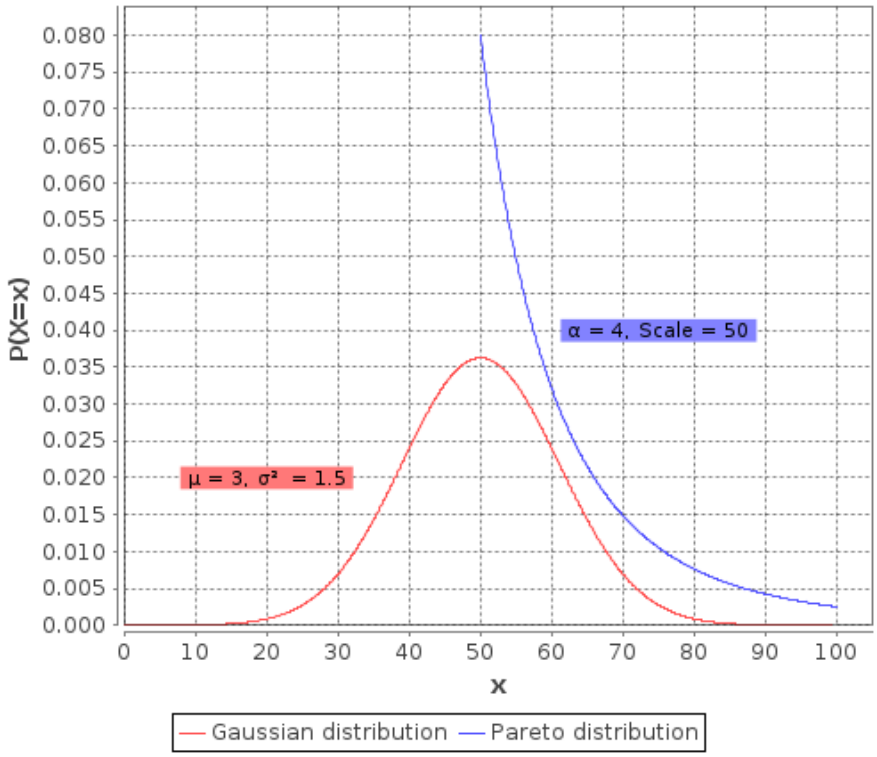


Figure 4.1: 10^6 samples rounded to 0.1 .

Listing 4.1: Rounding function

```
public static double round(double sample, double roundTo){
if(roundTo % 1 == 0){
// round to an integer
return Math.round((sample+roundTo/2)/roundTo) * roundTo;
}else{
// round to a double
return Math.floor(sample/roundTo) * roundTo;
}}
```

4.2.2 Network and Cloud Selection Agent

BURLAP is a Java library based on Object Oriented Markov Decision Process (OO-MDP) [60]. Implementing a Reinforcement learning agent using BURLAP is quite intuitive and inspired from the physical world.

In our physical world and over the course of history, we defined a number of attributes which describe how our world works. For example, if we take the earth as our domain,

Listing 4.2: Data generating plan.

```
// sample_number , scale , shape , roundTo
mG.addParetoPlan(500, 100, 5,10);
mG.addParetoPlan(200, 40 , 5,1 );
mG.addParetoPlan(100, 300, 6,5 );
mG.generate();
```

we have a latitude and longitude which describe the location of an object. Latitude and longitude are a domain attributes, and each object lives in that domain has to assign values to these attributes describing its position (state) on the surface of the earth. The domain has actions which apply to objects lives in that domain too. For example, we define an action move north which applies to an object lives in the earth. In BURLAP the process of creating a domain goes throw the following steps [61].

1. declare a domain-class (earth)
2. define the attributes which constructs this domain (e.g. latitude and longitude)
3. define object-class (objects) which will live in the domain (e.g. cars)
4. Assign the attributes to our object-class (a car has latitude and longitude)
5. define action applicable on this object-class (move north, move south ...)
6. instantiate the object-class to get an object (e.g. Bob's car)

We have defined *CloudWorld*, this domain contains attributes describing its state and has actions which are applicable to the object lives in it. Then we defined an agent lives in *CloudWorld* and assigned attributes to the agent. Code block 4.3 contains a simplified version of domain implementation.

4.2.3 Testing the Network and Cloud Selection Agent

To test and validate our implementation, we used the data as shown in Fig. 4.2. This dataset contains delay and throughput values of two networks ($WiFi, 4G$) and two clouds ($Cloud_1, Cloud_2$). Fig. 4.2 shows 1000 samples from one of our dataset (out of many), the samples are taken on the X axis, each sample contains the delay and throughput of four network-and-cloud. The first 800 samples has a clear division between what is considered as "good" and "bad" ($Network - Cloud$) combination. For example, for the first 200 samples, ($WiFi - Cloud_1$) has the least delay and highest throughput. Furthermore, in this dataset there are 100 points (between time 800 sec. to 900 sec.) where all network-and-cloud combinations do not provide a "good" delay or throughput. These 100 sample points aim to test a more complex situation where there is no obvious

Listing 4.3: Simplified domain implementation.

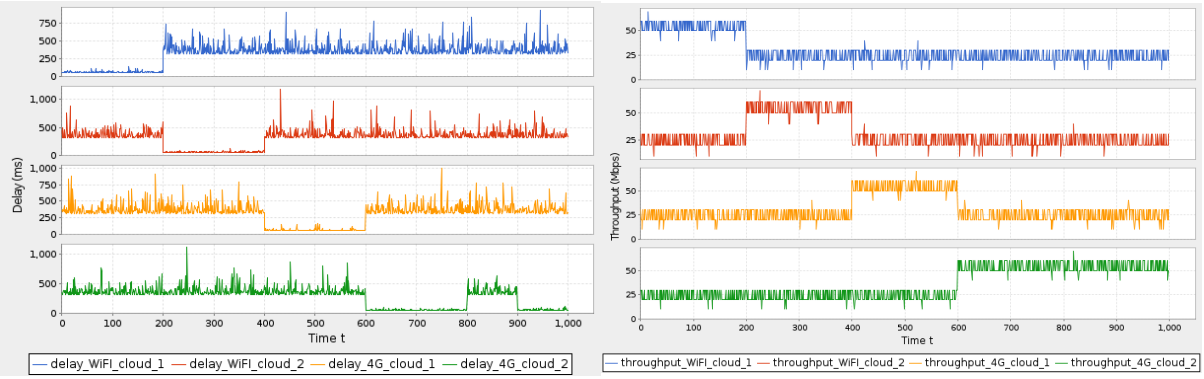
```
// a Single Agent Domain
SADomain domain = new SADomain();

// an attribute of the domain (Delay-Network1-Cloud1)
Attribute mD_N1_C1 = new Attribute(domain, D_N1_C1, Type.INT);

// an agent lives in the cloud world
ObjectClass mAgentClass = new ObjectClass(domain, "agent");

// This domain attribute is an attribute of the agent class
mAgentClass.addAttribute(mD_N1_C1);

// add action to the domain
new Migrate(MIGRATE_TO_N1_C1, domain, N1 ,C1);
```



(a) Delay (b) Throughput

Figure 4.2: Dataset to test the agent.

best network-and-cloud to use, i.e. delay and throughout values overlap significantly. In our dataset, the delay values (that follow Pareto distribution) contain 50 distinct values for each network-and-cloud (e.g. 50, 60, ... , 1200) ms, and the throughput values has 7 distinct values (e.g. 10, 20, ... , 70) Mbps for each network-and-cloud combination. The agent consider the whole sample as one state, and the total state space is $15 \cdot 10^9$ states ($50^4 \cdot 7^4$). This data is used as an input to verify the agent's behavior. However, before running the test, we need to set a couple of variables and these variable alter the agent's learning behavior.

- Greedy *Epsilon*: This value sets the greedy behavior of the agent. For example *Epsilon*$= 0.2 \rightarrow 20\%$ of the decisions will be random, and the remaining 80% will

be according to the Q-matrix.

- *Learning Rate*: *Learning Rate* determines to what extent the agent will weigh future learning experiences against the previous ones [62].
- *Discount Factor*: determines the importance of future rewards [62].

Test methodology

As mentioned above the three variables which alter the learning behaviour of the agent are *Epsilon*, *Learning Rate*, and *Discount Factor*. To get a better and practical understanding of Q-learning, algorithm 2 was implemented. The test goes through all the possible combination of *Learning Rate*, *Discount Factor*, *Epsilon* with a step of 0.1, we call each combination a tuple. The algorithm runs tests on 11^3 tuples. However, to understand the test shown in algorithm 2 and the results form the test we present some terminology.

- Learning: is the process in which the agent takes an action $a \in A$ in a state $s \in S$ and memorizes how "good" or "bad" the action was.
- Reward: The *reward function* (Eq. 3.7) tells the agent how "good" or "bad" its choice was.
- Episode: is a full run in the environment.
  - If we have a grid world with a goal state, an episode is all the actions from start to goal state.
  - The agent keeps the knowledge from previous episodes in the Q-matrix and uses this knowledge in future episodes.
  - In $CloudWorld$ domain, an episode is a full iteration on the generated data.
- Episode Reward: The agent takes some random actions (explore) and some action based on its Q-matrix (exploit). Episode reward is the sum of rewards during a learning episode. We have to keep in mind that regardless of how high or low the episode reward is, the agent updates the Q-matrix and improves the decision, this is a feature of the Q-learning.
- Greedy policy: After x learning episodes, we extract a greedy policy from the Q-matrix, the policy maps each state $s \in S$ to an action $a \in A$.
- Greedy Reward: is the sum of reward the agent gets when it acts according to its greedy policy. This reward represents the learning quality of the agent.
- Oracle agent: is an agent that has the optimal knowledge of the environment. In other words, this agent knows the best action to take in any given state.

- Convergence: In our test, we consider that the algorithm has converged, if and only if, the greedy reward is equal to the Oracle reward. In other words, the algorithm converges when the agent learns the optimal policy.

Algorithm 2 Testing the Implementation

```
for Epsilon = 0 to 1 step 0.1 do
    for Learning = 0 to 1 step 0.1 do
        for Discount = 0 to 1 step 0.1 do
            Initialize the agent ← (Epsilon, Learning, Discount)
            while GreedyReward < OptimalReward do
                episodeAnalysis ← agent.runLearningEpisode
                episodeReward ← EpisodeAnalysis.getCumulativeReward
                greedyPolicy ← agent.getGreedyPolicy
                episodeAnalysis ← agent.evaluatePolicy(GreedyPolicy)
                greedyReward ← EpisodeAnalysis.getCumulativeReward
            end while
        end for
    end for
end for
```

4.2.4 Agent's Testing results

From the 1331 tested tuples, 504 tuples have converged within 1000 learning episodes. Fig. 4.3 shows the relation between convergence and number of learning episodes. "good" tuples converge with less learning than "bad" tuples. The "best" tuples have converged with 99 learning episodes, while the "worst" have converged with 1000 learning episodes. Across the range {"best tuple","worst tuple"}, the number of converged tuples is fairly linear to the number of episodes, for example, within 500 learning episode 303 tuples have converged. Fig. 4.4 shows the relation between the each converged tuple (*Learning Rate* (X), *Discount Factor* (Y), *Epsilon* (Color) and the number of learning episodes (Z)) shows all the converged tuples. We can see that the area with high *Learning Rate* and low *Discount Factor* contains most of the converged tuples. As mentioned earlier the best tuple converged with 99 learning episode, the figure shows that a less efficient tuple can converge with more learning episodes or higher *Epsilon*. In both cases the agent overcompensate the "bad" tuples value with more learning episodes. Keeping in mind that these results reflect the nature of the input data. The state space is $15 \cdot 10^9$ states; we take a 1000 sample from this state space and feed it to the agent. The probability of any state to occur twice is fairly low; this is why the agent prefers high *Learning Rate* i.e.

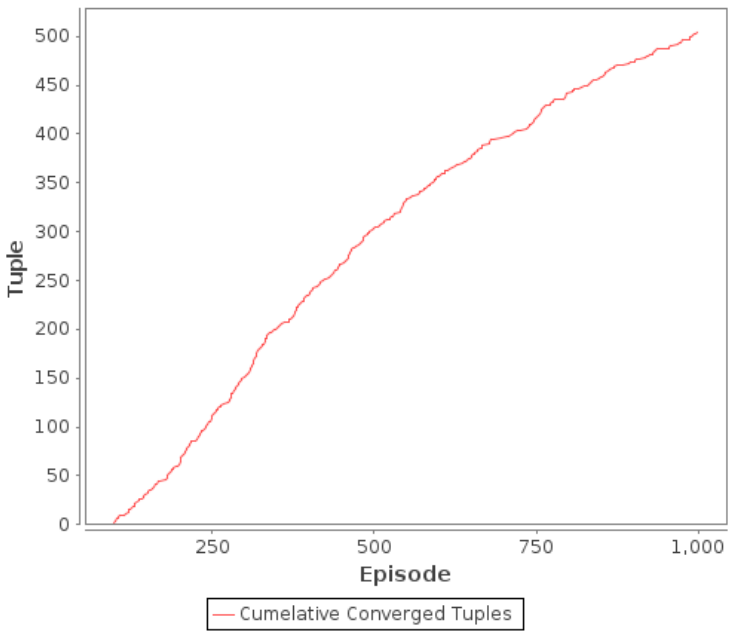


Figure 4.3: The relation between convergence and learning episodes.

the agent learns the most from each state because most likely the same state may not be encountered again. Moreover, the state space does not have any hierarchy, its a fully connected Markov chain. In a fully connected Markov chain considering future rewards will confuse the agent, and this why the agent prefer low *Discount Factor*. It does not matter in which state the agent is, it always can reach to all of the state space. To have a clearer view of this behavior (i.e. high *Learning Rate*, low *Discount Factor*). Fig. 4.5 shows converged tuples with *Epsilon* value set to 0.7 and 0.3.

4.2.5 Enhanced Agent

Testing the application live-migration environment showed that migrating an application is quite expensive in term of time. Therefore, we enhanced the learning process of the network and cloud selection Agent, aiming at making the agent learn faster. Furthermore, the enhancement overcome the puzzle of choosing constant values for *Epsilon*-greedy and *Learning Rate*. We improved the agent by implementing explore and *Learning Rate* functions, proposed in algorithm 1 and Eq. 3.9 respectively. The core component in these two functions is *StateActionOccurrenceMapper* class which as the name suggests, maps state-action to the occurrence. To test the enhanced agent we set the *Discount Factor* to 0 and 0.1, and we run the test for 100 times each. The enhanced agent converged with an average of 39.63 episode and 40.80 respectively. Table 4.2 compare the raw and enhanced agents, both agents reach to the Oracle agent knowledge, but the enhanced agent reach there 60% faster. Figure 4.6 shows the agent visualizer, the visualizer include the delay

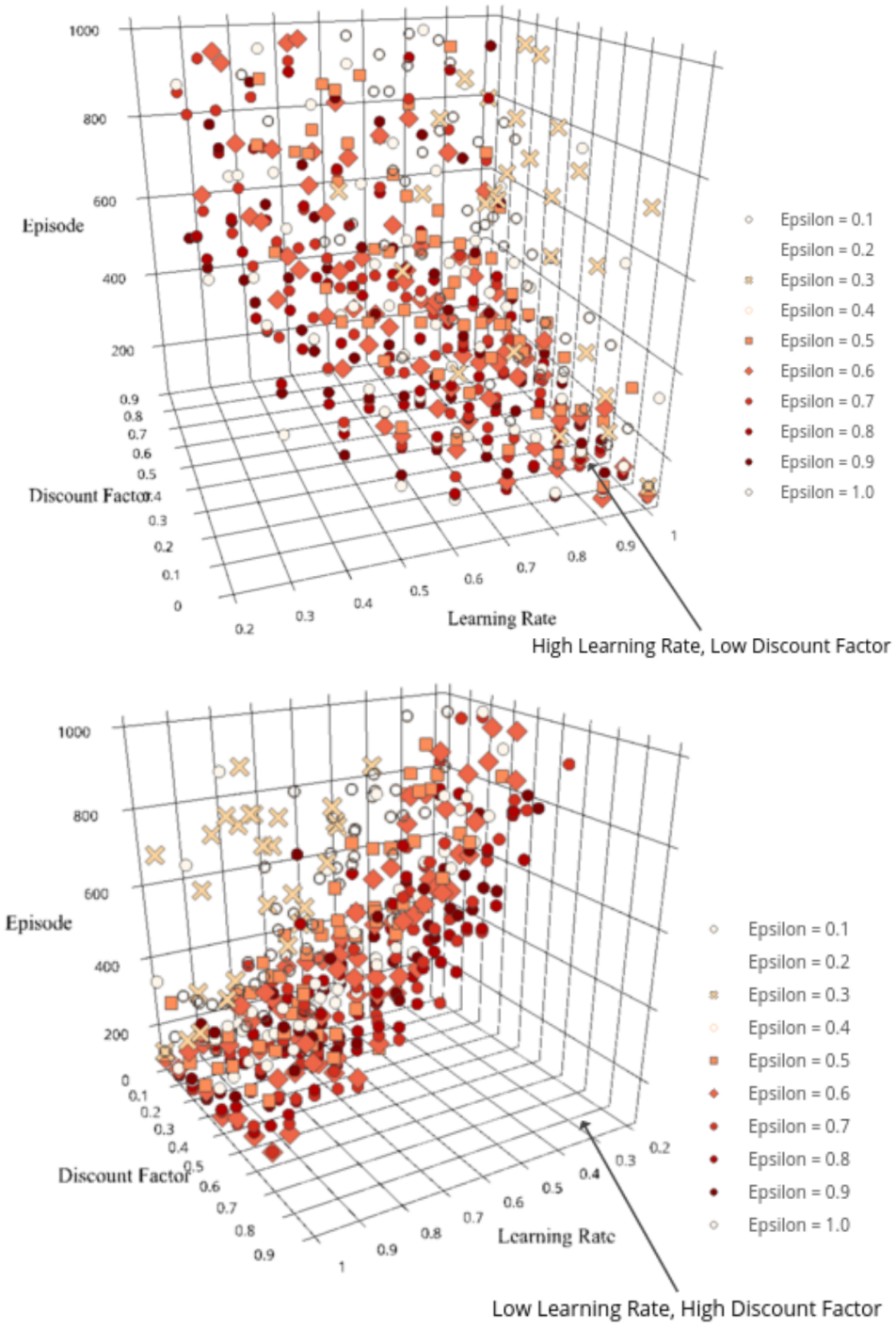
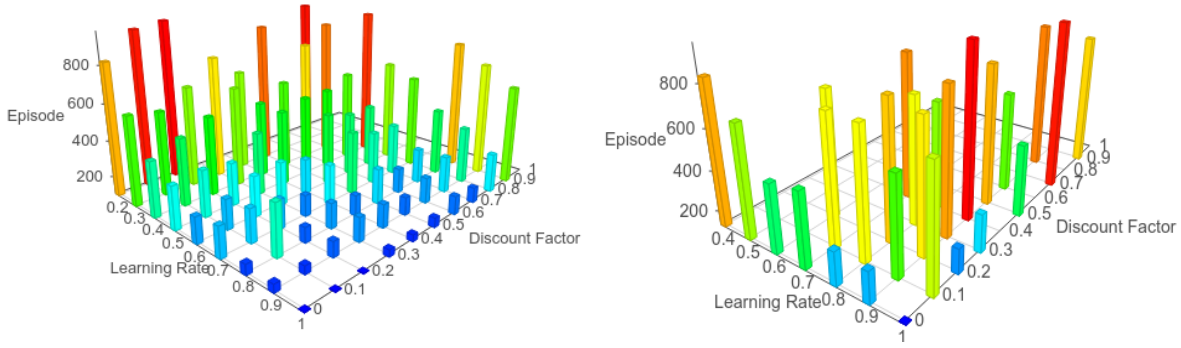


Figure 4.4: All the Converged tuples (1331).

and throughput statistics of all the available networks and clouds. Further, the visualizer shows where the application is running, where Figure 4.6a is the application position



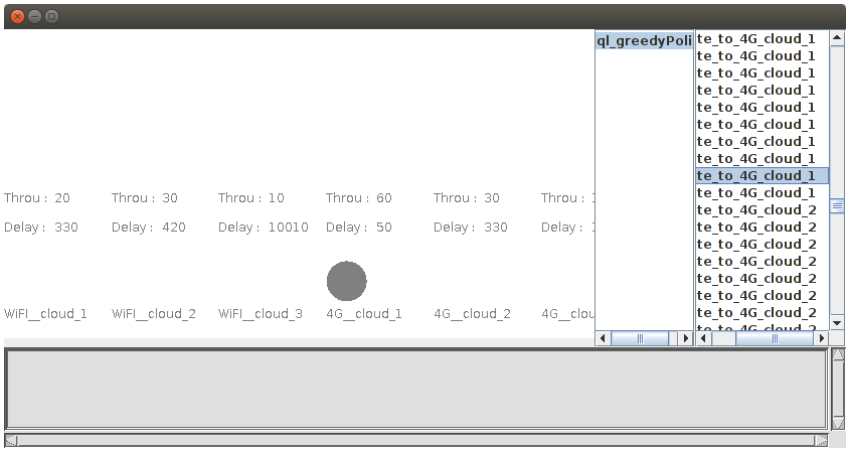
(a) *epsilon* = 0.7 (b) *epsilon* = 0.3

Figure 4.5: Converged tuples with *epsilon* set to 0.7 and 0.3 .

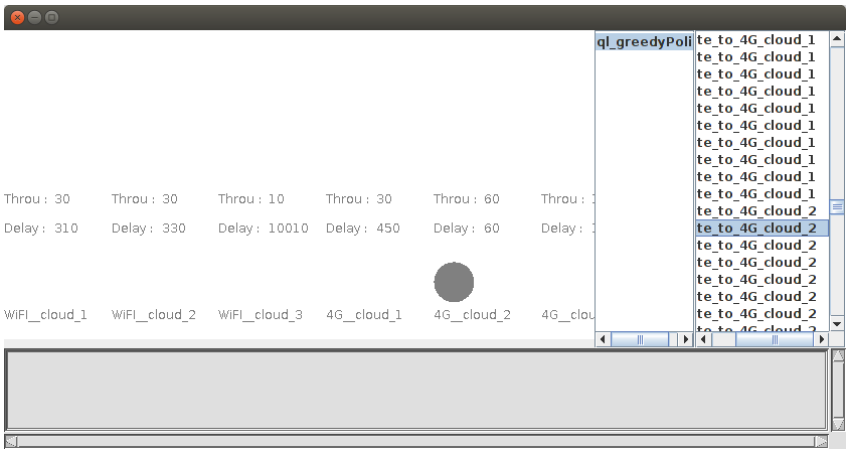
before performing a migration and 4.6b is the state after performing a migration. The visualizer does not play any role in the test, but it was a helpful tool to understand the agent's behavior.

Table 4.2: Comparison between raw and enhanced agents

Agent	Epsilon, Learning, Discount	Agent converge after x episode	Improvement
Raw	1.0, 1.0, 0	99	-
Raw	0.9, 1.0, 0.1	99	-
Enhanced	function, function, 0	≈ 40	$\approx 60\%$
Enhanced	function, function, 0.1	≈ 41	$\approx 59\%$



(a) Before migrating.



(b) After migrating.

Figure 4.6: Agent Visualizer.

4.3 Summary

In this chapter, we implemented and tested two separate components of APOLLO system; the first component addresses application live-migration by setting up and testing an application live-migration environment. The second component addresses network-and-cloud selection by implementing a network and cloud Selection Agent. After testing a "standard" implementation of the agent in 4.2.4, we enhanced the agent by implementing Explore and Learning Rate functions. Testing the enhanced agent shows 60 % improvement in learning time.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis has identified and addressed a number of research challenges in the area of Mobile Cloud Computing (e.g. Mobility management, Application migration, and network and cloud selection). Mobility management, Application migration, and network and cloud selection are challenging tasks; each one them belongs to its own domain of research. For example, to select the "best" network and cloud in a giving situation (i.e. infer which network has the highest throughput and lowest delay) has nothing in common with managing mobility (i.e. performing a seamless handoff). To offload the computation to the cloud (implementing MCC vision), we need a solution which cross bridge these areas of research. Our solution was introducing A Proactive Application Migration System in Mobile Cloud Computing (i.e. APOLLO). APOLLO incorporates M-MIP, Linux Containers, and Reinforcement Learning Agent to support terminal mobility and application migration respectively. To develop the system, we tested state-of-the-art container technology and validated its use to facilitate application migration. Further, we have proposed, implemented, and tested a Reinforcement Learning network and cloud selection agent.

5.1 Future Work

As a future work, we plan to incorporate the developed components in this thesis to test APOLLO as a complete unit. Implementing and incorporating M-MIP to the developed testbed and testing the architecture as a complete unit will be immensely valuable for any future work. Further, the application migration feature in the testbed operates by the "stop the world" method. Using "stop the world" CRIU stops the application, chick point it to a file and restore the file on another machine. However, CRIU supports iterative incremental transfer via p.haul protocol which we plan to utilize to reduce the

down time of the applications. The agent's test results shown in this thesis focused on reaching convergence as fast as possible, further analysis of the collected data ($3 \cdot 10^6$ learning episodes) have to be done to find the optimal variables that balance convergence time and user's QoE. Moreover, testing the agent with real network and cloud statistics will assure the correctness of agent's design.

5.2 Lessons learned

In this section, we summarize what we have learned during this research. This summary is concerned with abstract ideas and how this work contributed to our logical thinking. Figure 5.1 is an oversimplification of this work. However, we believe that this simplicity is valuable. The research question is how to extend the resources of mobile devices. To do so, we either have to find a hardware based solution or a software based solution. Hardware based solution is not happing soon. Looking for a software based solution, we find Mobile Cloud Computing (MCC). Implementing an MCC system has three main challenges each of which belong to a separate domain of research. We look at each of these challenges to find out why this is a challenge. Then what is the abstract solution of this challenge, and finally, we identify an implementation of this abstract solution.

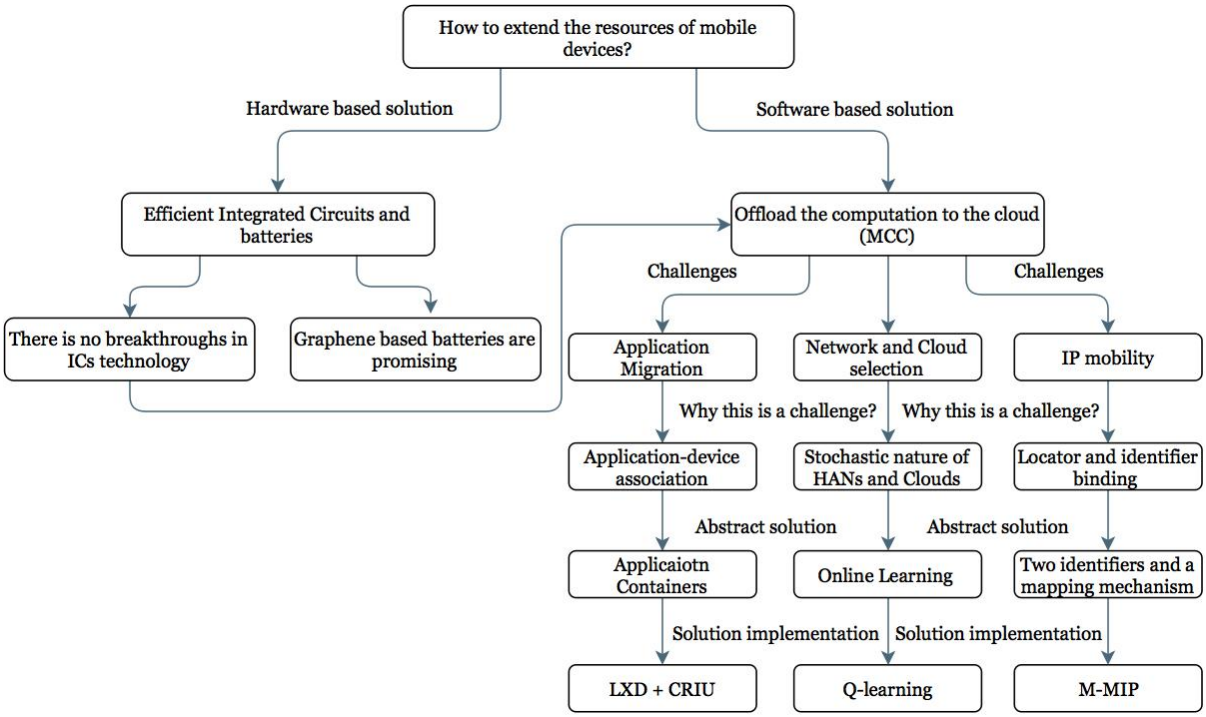


Figure 5.1: Thesis abstraction

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 154–172, 2015.

[3] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[4] J. Manner and M. Kojo, "Mobility related terminology," tech. rep., 2004.

[5] P. Richards, *How Cloud Computing is Changing the Mobile Space*, 2013 (accessed June 22, 2016).

[6] P. E. Ceruzzi, *A history of modern computing*. MIT press, 2003.

[7] IBM, *IBM Archives: System/360 Model 25*, 2016 (accessed June 26, 2016).

[8] C. N. Trueman, *The Personal Computer*, 2016 (accessed June 26, 2016).

[9] A. S. Lett and W. L. Konigsford, "Tss/360: A time-shared operating system," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 15–28, ACM, 1968.

[10] L. Łukasiak and A. Jakubowski, "History of semiconductors," *Journal of Telecommunications and information technology*, pp. 3–9, 2010.

[11] L. Press, "Personal computing: the post-pc era," *Communications of the ACM*, vol. 42, no. 10, pp. 21–24, 1999.

[12] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pp. 233–240, IEEE, 2013.

[13] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pp. 171–172, IEEE, 2015.

[14] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures–a technology review," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pp. 379–386, IEEE, 2015.

[15] A. Tosatto, P. Ruiu, and A. Attanasio, "Container-based orchestration in cloud: state of the art and challenges," in *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*, pp. 70–75, IEEE, 2015.

[16] E. W. Biederman and L. Networx, "Multiple instances of the global linux namespaces," in *Proceedings of the Linux Symposium*, vol. 1, pp. 101–112, Citeseer, 2006.

[17] M. Kerrisk, "Namespaces in operation, part 1: namespaces overview," *Online, http://lwn. net/Articles/531114*, 2013.

[18] C. Ltd, *Linux Containers LXC*, 2016 (accessed March 6, 2016).

[19] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[20] Z. C. Schreuders, T. McGill, and C. Payne, "Empowering end users to confine their own applications: The results of a usability study comparing selinux, apparmor, and fbac-lsm," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 2, p. 19, 2011.

[21] J. Corbet, "Seccomp and sandboxing," *LWN. net, May*, 2009.

[22] S. Friedl, "Go directly to jail: Secure untrusted applications with chroot," *Linux Magazine*, pp. 2002–12, 2002.

[23] C. Ltd, *Linux Containers LXD*, 2016 (accessed March 6, 2016).

[24] C. Ltd, *LXD documentation*, 2016 (accessed March 19, 2016).

[25] criu.org, *CRIU*, 2016 (accessed March 6, 2016).

[26] M. Y. Malik, "Power consumption analysis of a modern smartphone," *arXiv preprint arXiv:1212.1896*, 2012.

[27] G. P. Perrucci, F. H. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pp. 1–6, IEEE, 2011.

[28] K. Mitra, *Quality of experience measurement, prediction and provisioning in heterogeneous access networks*. PhD thesis, 2013.

[29] J.-J. DeLisle, *NFC Prepares For Wide Adoption*, 2016 (accessed July 05, 2016).

[30] W. M. Eddy, "At what layer does mobility belong?," *Communications Magazine, IEEE*, vol. 42, no. 10, pp. 155–159, 2004.

[31] Z. Zhu, L. Zhang, and R. Wakikawa, "A survey of mobility support in the internet," 2011.

[32] J. Ioannidis, D. Duchamp, and G. Q. Maguire Jr, "Ip-based protocols for mobile internetworking," in *ACM SIGCOMM Computer Communication Review*, vol. 21, pp. 235–245, ACM, 1991.

[33] S. J. Koh, M. J. Chang, and M. Lee, "msctp for soft handover in transport layer," *IEEE communications letters*, vol. 8, no. 3, pp. 189–191, 2004.

[34] P. Eronen, "Ikev2 mobility and multihoming protocol (mobike)," 2006.

[35] T. Kivinen and H. Tschofenig, "Design of the ikev2 mobility and multihoming (mobike) protocol," tech. rep., 2006.

[36] H. Yin and H. Wang, "Building an application-aware ipsec policy system," *Networking, IEEE/ACM Transactions on*, vol. 15, no. 6, pp. 1502–1513, 2007.

[37] K. Andersson, D. Granlund, and C. Åhlund, "M 4: multimedia mobility manager: a seamless mobility management architecture supporting multimedia applications," in *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, pp. 6–13, ACM, 2007.

[38] C. E. Perkins, "Mobile ip," *IEEE communications Magazine*, vol. 35, no. 5, pp. 84–99, 1997.

[39] H. Soliman, L. Bellier, and K. E. Malki, "Hierarchical mobile ipv6 mobility management (hmipv6)," 2005.

[40] C. Åhlund and A. Zaslavsky, "Multihoming with mobile ip," in *IEEE International Conference on High Speed Networks and Multimedia Communications*, pp. 235–243, Springer, 2003.

[41] C. E. Perkins and D. B. Johnson, "Route optimization for mobile ip," *Cluster Computing*, vol. 1, no. 2, pp. 161–176, 1998.

[42] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, "Proxy mobile ipv6," tech. rep., 2008.

[43] K. Mitra, S. Saguna, and C. Åhlund, "M 2 c 2: A mobility management system for mobile cloud computing," in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1608–1613, IEEE, 2015.

[44] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[45] M. Roman, H. Ho, and R. H. Campbell, "Application mobility in active spaces," in *1st international conference on mobile and ubiquitous multimedia, Oulu, Finland*, 2002.

[46] D. Johansson, K. Andersson, and C. Ahlund, "Supporting user mobility with peer-to-peer-based application mobility in heterogeneous networks," in *Local Computer Networks Workshops (LCN Workshops), 2013 IEEE 38th Conference on*, pp. 150–153, IEEE, 2013.

[47] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[48] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of qoe in cloud gaming based on subjective tests," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pp. 330–335, IEEE, 2011.

[49] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[50] M. L. Puterman, "Markov decision processes: Discrete dynamic stochastic programming," *New York, NY: John Wiley. doi*, vol. 10, p. 9780470316887, 1994.

[51] E. Stevens-Navarro, Y. Lin, and V. W. Wong, "An mdp-based vertical handoff decision algorithm for heterogeneous wireless networks," *Vehicular Technology, IEEE Transactions on*, vol. 57, no. 2, pp. 1243–1254, 2008.

[52] E. A. Hansen and Z. Feng, "Dynamic programming for pomdps using a factored state representation.," in *AIPS*, pp. 130–139, 2000.

[53] K. Mitra, A. Zaslavsky, and C. Åhlund, "Pronet: Proactive context-aware support for mobility in heterogeneous access networks," in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pp. 675–676, IEEE, 2009.

[54] L. Bottou, "Stochastic learning," in *Advanced lectures on machine learning*, pp. 146–168, Springer, 2004.

[55] A. S. Foundation, *Apache Jmeter*, 2016 (accessed July 02, 2016).

[56] W. Zhang and J. He, "Modeling end-to-end delay using pareto distribution," in *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*, pp. 21–21, IEEE, 2007.

[57] J. Kilpi and I. Norros, "Testing the gaussian approximation of aggregate traffic," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 49–61, ACM, 2002.

[58] C. E. O. for Nuclear Research, *Colt Library*, 2016 (accessed June 6, 2016).

[59] C. Math, "The apache commons mathematics library," *URL: http://commons. apache. org/proper/commonsmath/(visited on 09/08/2013)(on p. 52, 162)*, 2014.

[60] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 240–247, ACM, 2008.

[61] J. MacGlashan, *Building a Domain using BURLAP*, 2016 (accessed April 17, 2016).

[62] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *The Journal of Machine Learning Research*, vol. 5, pp. 1–25, 2004.