Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Networked Systems and Services

Cloud-Native IP Mobility Management: from theory towards implementation

TDK

Author: Mohamad Saleh Salah
Budapest University of Technology and Economics
Budapest, Hungary
mhd.s.salah@gmail.com

Supervisor: Ákos Leiter
Nokia Bell Labs
Budapest, Hungary
akos.leiter@nokia-bell-labs.com

Supervisor: László Bokor
Budapest University of Technology and Economics
Budapest, Hungary
bokorl@hit.bme.hu

28/10/2020

Contents

List of figures:

List of Tables:

1- Introduction

Nowadays, the amount of wireless data is increasing in an unpredicted way. The type of applications takes a new shape, which needs real-time seamless service provision. Moreover, some of these applications are sensitive to data loss. The problem is even more severe when the users (mobile node or any moving device connected to a wireless network) change their point of attachment in the network. The mobility management aims to provide continual service to the users when moving from one point to another. And the ultimate goal of mobility management is to provide seamless handover to the moving devices.

The idea of cloud computing is based on delivering the broadest scale of computing services like mobility management or networking in general, over a pool of infrastructures that form the cloud. The cloud-native approach aims to take the advantages of using cloud computing to "build and run scalable applications in modern, dynamic environments. Containers, service meshes, microservices, and immutable infrastructure, exemplify this approach." [9].

Since the Cloud Native Computing Foundation's establishment in 2015 [9], technology is leading companies to fit their solutions, protocols, and infrastructures into a cloud-native environment. Mobility management is one of those technologies to be merged into the cloud: it should adapt to the rapid technological evolution of mobile telecommunication infrastructures caused by cloudification, virtualization, and softwarization trends. We believe that the adoption of IP mobility management in a cloud-native environment is a promising approach to help mobile operators building a more efficient networking architecture[5].

Firstly, in our work, we introduce the general idea of cloud-native applications and services, how everything started, and the main steps of this evolution [9].

We study the state-of-the-art enabler technologies, including SDN, NFV, cloudification, containerization, etc.., where we focus on the core element of the cloud-native "Kubernetes" and highlight Kubernetes networking solutions [7].

We also analyze the evolution of mobility management techniques and focus on IP mobility management to summarize the evolution path from the beginning towards the cloud-native operation of the most popular IP mobility management protocols: Mobile IPv6 (MIPv6) [1] and Proxy Mobile IPv6 (PMIPv6) [2][3].

We take MIPv6 as the basis of a series of practical experiments and introduce a cloud-native MIPv6 design proposals. Potential implementation in an environment based on a Docker container engine [10] for advanced work, Kubernetes container orchestration platform [11] is applied.

2- Cloud-native

a. Microservices evolution

From the first beginning, a paradigm called service-oriented computing (SOC) refers to distributing processes and computing in object-oriented and component computing. SOC came with the idea of let some services provide functions to other components. Then a complicated method calls service-oriented architectures (SOA) came and described the requirement of the services. The next generation is the microservices that remove the complexity and aim to build a simple service with one functionality.

The microservice term is correlative with the idea of cloud-native. The microservices are defined as "a cohesive, independent process interacting via messages". The microservice architecture is a combination of distributed microservices modules.

The microservice paradigm has become popular these days in both academic and industrial domains. And it made a revolution in comparison to the monolithic legacy application.

Microservice characteristics:

- The microservice is characterized by the tiny size and small base code, which makes it easy to test. It minimal the bugs; when a bug is executed in a microservice application, it does not affect the whole system.
- With microservice, continuous integration is possible, and that is mean a new version of an application could be deployed beside the old version without affecting the environment.
- Working with small unit decrease the outage time during rebooting and maintenance.
- The microservice bonded to the containerization technology, and this provides a high level of freedom in developing and configuring the services.
- The microservices are very scalable and easy to maintain[12].

Later on, the cloud-native architecture builds on microservice terminology.

b. Cloud-native the big picture

Those days hundreds of technologies are introduced day after day, the need for transformation in the application structure is needed to handle this fast development, the business and IT leaders are forced to push to build a new agile generation of application, and the focus came on the cloud-native application [14][15].

There are Many definitions for cloud-native application as in [14] "Cloud native applications scale efficiently and autonomously; expect and tolerate partial failures; move fluidly from one environment to another for cost or performance optimization and much more.", or as in [15] "A cloud-native application is an application built to take advantage of cloud computing models to increase speed, flexibility, and quality while reducing deployment risks."

Cloud-native application has advantages of scalability, redundancy, efficiency, rapid delivery, and dynamic resource management [13][14]. Besides that, cloud-native applications could be delivered in highly automated methods, which increase the speed of delivery of the businesses; also, one of the most significant advantages of using cloud-native is the open-source of the most supported technologies.

To build a Cloud-native application, particular architecture and development guidelines should be considered [13]. Design, deploy, and develop the applications will be different [14].

Cloud-native applications already in the ongoing stage as the infrastructures have started to move from hardware-based to software-based, and virtualization-based development overtakes the legacy application development [14].

Moving to the cloud-native paradigm, go through 4 steps as in [14]:

1. The containerization: using containers is necessary. The container is a mature method to package code in a light and portable manner. This leads to easy integration in different environments and reduces environment-specific deployment costs, besides time and cost-saving advantages.
   Using Kubernetes containers orchestration platform even gives more advantages (described in section **3-e**) as Kubernetes can automate the application deployments in a very flexible way besides many other benefits.
2. Modernization: In this step, the organizations should design the whole life cycle of the application in a more cloud-native scheme, and to build a new cloud-native application, the following points should be taken into account:
   a) Deploy the Applications components in a scalable and independent manner using microservices- architecture.
   b) Apply continuous integration/continuous deployment automation to achieve agility in the design.
   c) Using containers and the benefits beyond that.
   d) Utilize efficient containers orchestration platforms like Kubernetes.
   e) Using DevOps terminology to integrating application development, operational, and process.

   On the other hand, modernized an already existed application is more complicated and convoluted. In this case, the modernization starts from moving the parts that need continual updates and the application into microservices.
3. Formalization: Many lessons from the previous experiences could be used within industries or in the smallest units in the formalization process. Some models and platforms highly support the whole life cycle of a cloud-native application like Cloud Foundry Platform and The Twelve-Factor App methodology.
4. Standardization: The procedure for delivering the applications should be standardized. The standardization increases the efficiency of the operations.

Figure 1 illustrates the whole process.

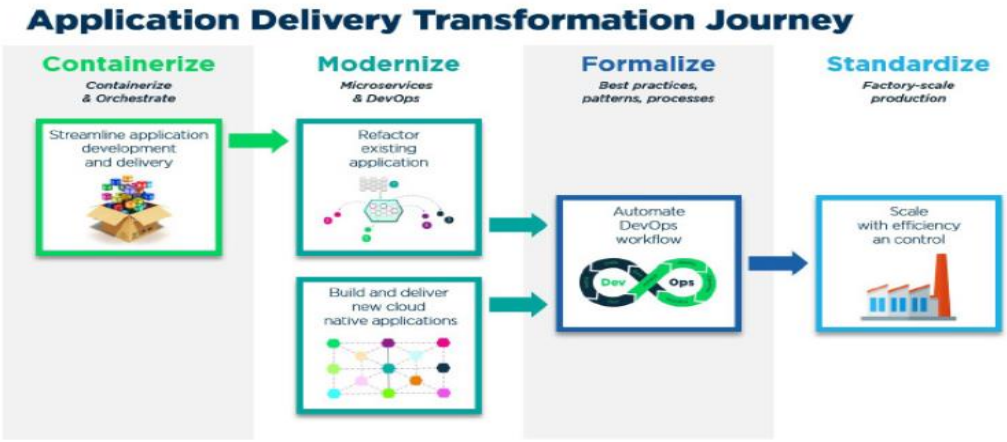

Figure 1 - The path to cloud-native[14]

Another point of view to move to cloud-native introduced in [15] and include eight steps; however, it leads to the same results:

1. "Evolve DevOps culture and practices.
2. Speed existing applications using fast monoliths
3. Use application services to speed development
4. Choose the right tool for the right task
5. Provide self-service, on-demand infrastructure
6. Automate it to accelerate application delivery
7. Implement continuous delivery and advanced deployment techniques
8. Evolve a more modular architecture."

Table 1 compares the development of a traditional application with the cloud-native application [15]:

	Traditional	Cloud-native
Focus	- Longevity - stability	- Speed to market
Development Methodology	- semi-agile development	- Agile development - DevOps
Delivery Cycles	- Long	- Short and continuous
Application Architecture	- Tightly coupled - Monolithic	- Loosely coupled - Service-based - Application programming interface (API)-based communication
Infrastructure	- Server-centric - Designed for on-premise - Infrastructure-dependent	- Container-centric - Designed for on-premise and cloud - Portable across infrastructure

	- Scales vertically	- Scales horizontally

Table 1 - comparison between traditional application and cloud-native application

Network functions could be cloud-native under some conditions using virtualization techniques. The basic principle to design VNF using cloud-native: The function should be designed as micro-services, lightweight, component-based, and should be restored quickly in case of any failure [16].

c. Cloud-native statistics

CNCF has over 500 members, including the world's most enormous public cloud and enterprise software companies and over two hundred innovative startups[9].

Based on the CNCF Survey at the end of 2019 (1337 participants from different levels and different companies), the use of cloud-native projects has increased in the production environment, also use of Kubernetes in the production environment jump from 58% in 2018 to 78% in 2019, and the usage of the container increase rapidly since 2016 (note: the statistics taken based on previous surveys)[17].

Figure 2 represents a survey for cloud-native usage evolution:

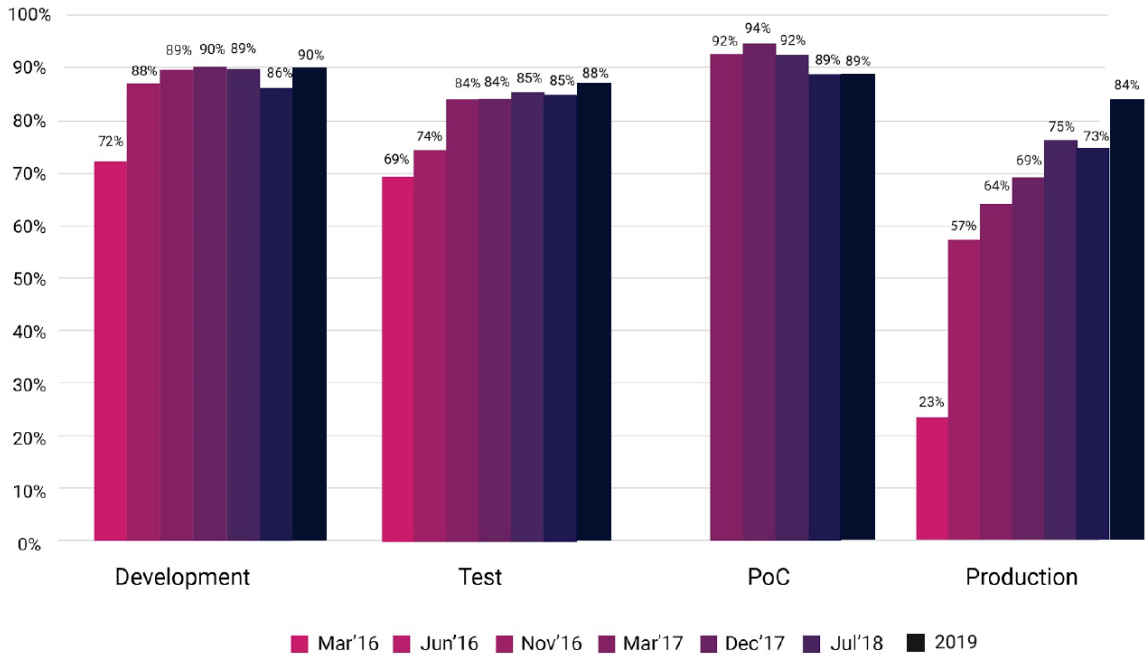

Figure 2 - Use of Containers since 2016[17]

Based on [14], almost 60% of enterprises worldwide are leveraging Kubernetes, and 27% are planning to do soon.

3- State-of-art technologies

The telecommunication services affected by the new IT revolution, the cloud will have existed everywhere in the network (core, edge, etc..), and most of these services are fitting in virtualized design using NFV paradigm.

SDN, on the other hand, is an excellent candidate for enhancing the Mobility Management process, as it helps in automate the whole networking process and move the networking infrastructure to a new era. There are many implementations and research for leveraging SDN with Mobility management.

a. NFV

i. Overview

There are many limitations to using closed hardware equipment to provide specific functions. This hardware can rapidly reach their end of life due to an unpredicted explosion in technological innovations[19].

The concept of network function virtualization (NFV) transforms the network architecture into a new era. NFV came with the idea of decoupling the dependency on the hardware platform. NFV terminology uses the IT virtualization technologies to build virtual network function (VNF) or chain these functions to form communications advanced network services (NSs) in a dynamic way[18].

Decoupling the software from the hardware let the software and the hardware to evolve independently from each other, and this gives the flexibility to assign the needed service (hardware or software) in a convenient place; finally, it provides the network with the dynamically and fixability in terms of operations and scaling[18].

There are many NFV applications and use cases [20]. NFV can apply to any functions in the mobile or fixed network (e.g., mobile core node (MME, S-GW, P-GW), switching/routing elements, tunneling elements, signaling elements like IMS, etc.) [19].

ii. NFV objectives and benefits

The main objectives and benefits of NFV introduced in [18] and [19]:

- By decoupling the Network Functions from the dedicated hardware and using commercial-off-the-shelf (COTS) hardware, NFV should provide a more efficient implementation than the legacy implementations.
- Assign the VNFs to the hardware in an elastic way led to more scalability in resource allocation in the best convenient place and improve the scale-up and scale-down operations.
- More efficient use of the hardware lead to reduce power usage and decrease equipment costs.
- Enhance operational efficiencies.
- Build open interfaces and standers between the NFs and the infrastructure.
- Software-based service deployment.
- A Single platform can support different applications, different uses, and a wide range of eco-system.

iii. NFV architecture

The European Telecommunications Standards Institute (ETSI) standardize the architecture of NFV in three layers, as in **Figure 3**, the layers are: network function virtualization infrastructure (NFVI), virtualized network functions (VNFs), and network function virtualization management and orchestration (NFV MANO) [18].

- NFVI transforms the physical hardware into a pool of virtualized resources by using virtualization technology. NFVI provides the Virtual machines (VMs) and the needed resources to the upper layer. NFVI Consist of Hardware (server-storage-network) and cloud OS (hypervisor and Management module (e.g., OpenStack)).
- VNF is the software implementation of the network functions on top of NFVI. Element Management System (EMS) is implemented in this layer, and each EMS is responsible for typical management for the NS. Mobility Management Entity (MME) in the LTE network is an example of NF.
- MANO is the central management entity for the whole system. MANO manage and orchestrate the life cycle of the hardware, software, and the VNFs in the NFV environment. MANO formed from three layers:
    - Virtualized Infrastructure Management (VIM): control NVFI resources and perform operations over it.
    - Virtualized Network Function Management (VNFM): control VNF life cycle management (service deployment, expansion, deletion)
    - Network Function Virtualization orchestration (NVFO): control Network services (NSs) life cycle management, manage the whole network service layout, and schedule resources across data centers [18].

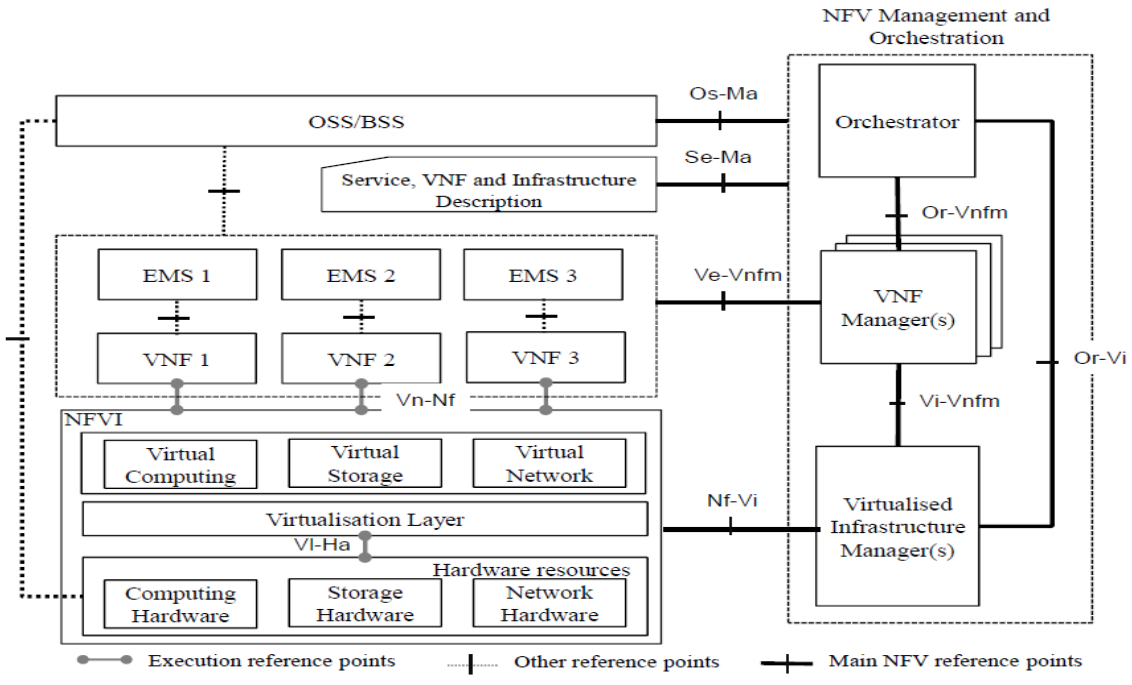

Figure 3 - NFV architecture[18]

iv. Network Service and Network Function

Network function (NF) is a well-defined functional block used to provide a particular service.

Network service (NS) is a set of network functions and the connections between these NFs to achieve end-to-end service hold in the NFV environment. The functionality of the NF builds based on the combination of NFs functionality. Mobile voice or data service is an example of NS [18].

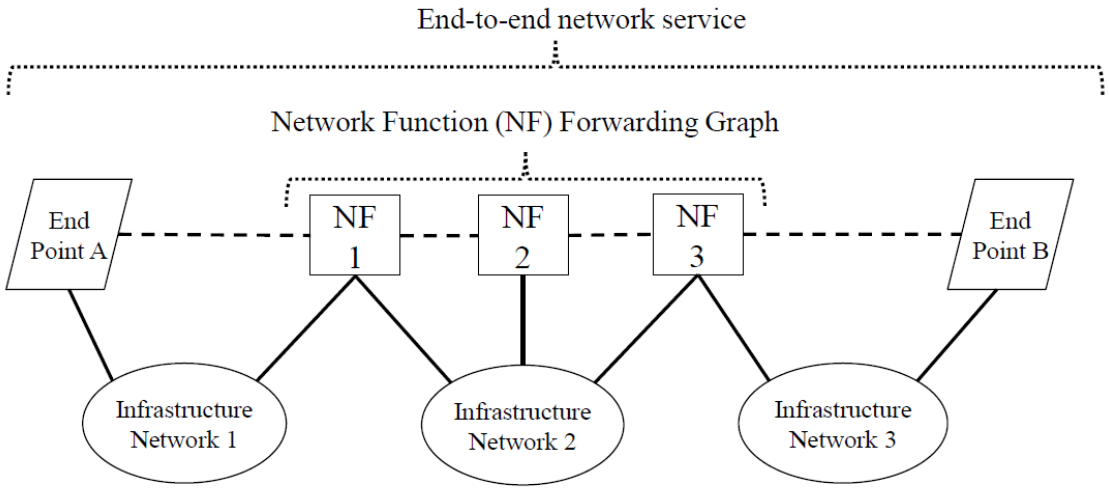

Figure 4 - network server architecture[18].

b. SDN

i. Overview

Due to the high increased number of mobile devices and the release of new technologies like server virtualization and cloud computing, the network architecture should adopt these changes and serve these technologies in a new innovative way.

Software-Defined Networking (SDN) transforms the network architecture into a new structure. SDN decouples the control plane and data plane in the networking devices. A centralized programable controlling unit called SDN Controller handles the controlling, signaling, and make decision roles, while the underlying infrastructure takes the part of forwarding data. This transformation provides the network with flexibility, scalability, and automation to adapt to business needs[21].

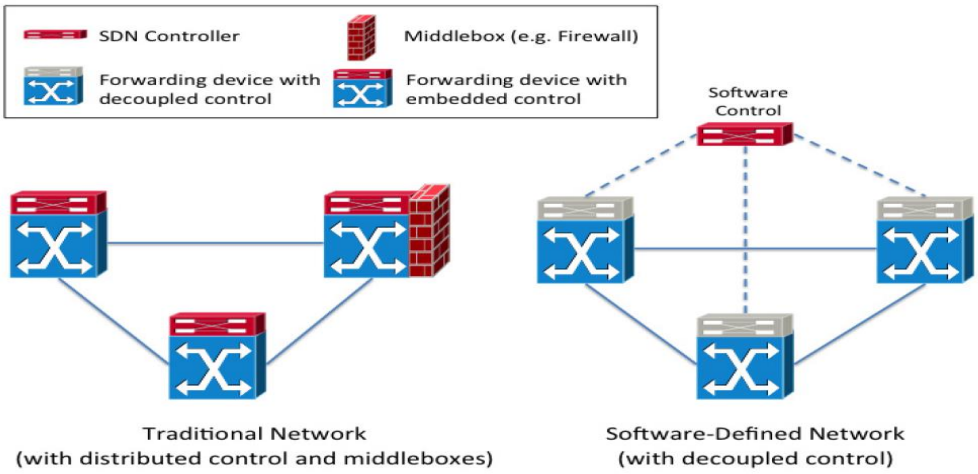

Figure 5 - Network architecture in SDN and traditionally based [22]

ii. SDN benefits

- Centralize the management process and the ability to use multiple vendors for networking devices.
- Enhance the automation and the management processes in the network.
- Deliver new solutions and services rapidly without the need to configure each device individually.
- Increase the reliability and the security of the network[21].

iii. Architecture

Three layers construct the SDN architecture: infrastructure layer, control layer, and application layer.

Starting from the bottom layer or the infrastructure, this layer is the data plane layer where the forwarding devices are working. It is responsible for forwarding data in the network and monitoring and gathering information and statistics locally. The middle layer is the control layer, this layer responsible for managing the control plane in the network. It defines the operations of the forwarding devices. The top layer is the application layer; this layer helps the control layer manage and configures the network, different applications, and features provided to the control layer by the application layer.

The communication between the infrastructure and control layers is done through southbound interfaces, and the control layer communicates with the application layer through northbound interfaces.

OpenFlow is the most popular protocol in the southbound interface. OpenFlow is responsible for installing the flow table entities in the forwarding devices, and those devices forward the data based on these entities[23].

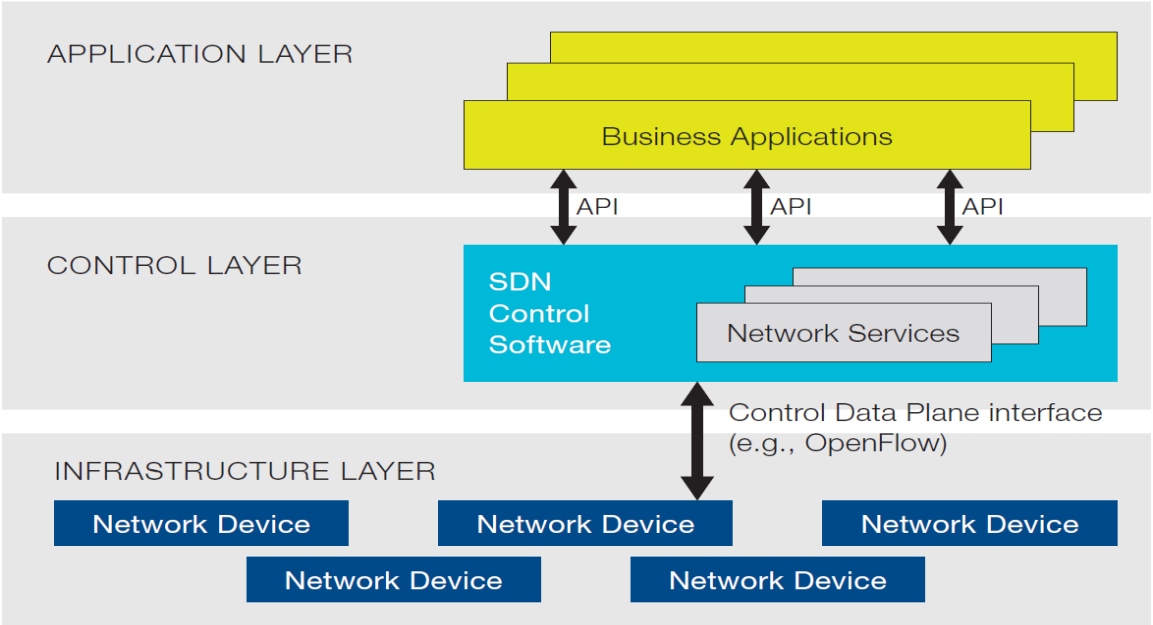

Figure 6 - SDN low-level design[21]

c. Cloud computing

The enterprises are working hard to integrate their resources in a cloud to have the agility, flexibility, scalability, and decrease infrastructure maintenance expenses[7].

Running a legacy application in a VM without any modification is inefficient and will not fully use cloud infrastructure to hold the application. Network Function Cloudification (NFC) solves this issue and provides the needed change to utilize the cloud's benefits fully. NFC takes into consideration provide connections between hosts, the separation between stateless application and centralized cloud storage [7]dynamically.

At the core of the cloud, virtualization technologies are applied. Using the cloud infrastructures increases resource availability and opens the possibility of use orchestration and management methods to automate the network procedures [19].

The cloud computing definition according to the National Institute of Standards and Technology (NIST) [24] is "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."

According to [24], cloud computing's main characteristics are On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, and Measured service. At the same time, there are three leading service models: Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS). And finally, the deployment models are private cloud, community cloud, public cloud, and hybrid cloud.

i. OpenStack

OpenStack [36] is an open-source cloud platform used to provide a virtualized cloud infrastructure by gathering a group of components over a pool of hardware resources. OpenStack can build infrastructure as a service IaaS, and it used to create a private and public cloud.

OpenStack is easy to control as the design could be customized based on need and could be integrated with many third-party technologies; it is also characterized by agility and cost-saving and support many eco-systems[40].

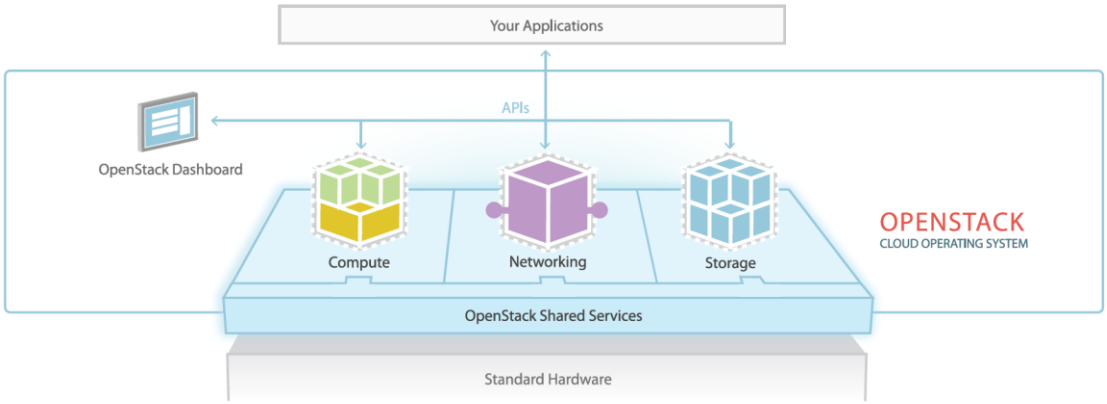

Figure 7 - OpenStack platform[40]

d. Containerization

i. Overview

The container definition in the IT world is "A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another" [10].

Comparison between VM and container in **Table 2** [35]:

	Virtual Machines	Containers
Guest OS	Each VM runs on virtual hardware, and Kernel is loaded into its own memory region	All the guests share the same OS and Kernel. Kernel image is loaded into the physical memory
Communication	Will be through Ethernet Devices	Standard IPC mechanisms like Signals, pipes, sockets etc.
Security	Depends on the implementation of the hypervisor	Mandatory access control can be leveraged
Performance	Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS.	Containers provide near-native performance as compared to the underlying Host OS.
Isolation	Sharing libraries, files, etc. between guests and between	Subdirectories can be transparently mounted and can

	guests hosts not possible.	be shared.
Startup time	VMs take a few mins to boot up	Containers can be booted up in a few secs as compared to VMs.
Storage	VMs take much more storage as the whole OS kernel, and its associated programs have to be installed and run	Containers take a lower amount of storage as the base OS is shared

Table 2 - Comparison between VM and Container

ii. Docker

Docker in Open-source containerization technology used to build and containerized applications[10].

Docker is beneficial in building cloud-native microservice applications efficiently and lightly [33].

It can offer most of the requirements for building a microservice application:

- Docker helps to increase the automating process's speed, by creating and launching a Docker container is based on script.
- Docker container is an isolated, portable box.
- Enhance resources utilization.
- Offered security at different levels[34].

e. Kubernetes

i. Overview and reasons to select Kubernetes

Docker has many limitations, especially when talking about deploying containers over multiple hosts and simultaneously managing numerous containers. That spot the light on the need for an upper-layer management system like Kubernetes[33].

"Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications."[11]. It is aimed to remove the overhead of building infrastructures and focus on the architect and deploy the applications. It can manage and run any code that runs inside containers in the PODs—the logging, monitoring, and alerting system chosen by the user.

Pod is the smaller elements that Kubernetes deal with, and it could contain one or more containers.

Kubernetes remove the boundary of building a lock-in infrastructure and create a cloud-native application that could be run on any cloud **Figure 8** example for that, and it supports most of the workloads[33].

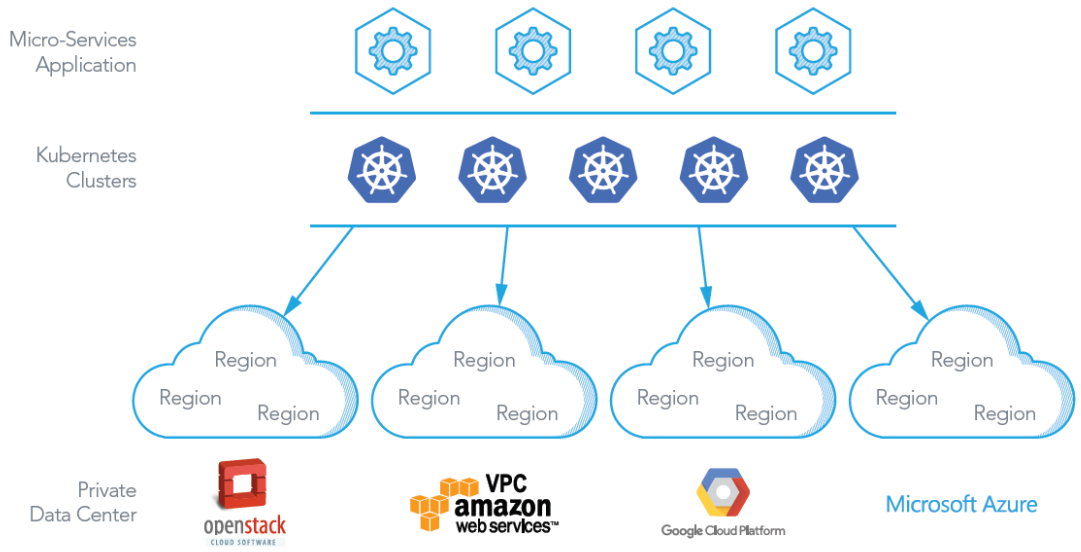

Figure 8 - Kubernetes over multiple clouds [33]

ii. Kubernetes Architecture

There are two types of nodes in Kubernetes that form the Kubernetes cluster, control node, and worker node.

The control node is responsible for the whole controlling process; it is considered the Kubernetes platform's brain, all the node's decisions.

The control node consists of a group of components:

- kube-apiserver: responsible for exposes the Kubernetes API.
- Etcd: highly-available key value to store all cluster data.
- kube-scheduler: select a specific node that should run the pods.
- kube-controller-manager: the main controlling component, it has many roles like check the node's availability, maintaining the correct number of pods, and some other functions.
- cloud-controller-manager: only existed if cluster builds based on cloud provider. It is used for links the cluster into the cloud provider's API.

On the other hand, the real applications and containers work on the worker node.

The worker node consists of three main elements:

- kubelet: this component should exist on all worker nodes; it is responsible for ensuring that the containers in a Pod are running and healthy.
- kube-proxy: it works to maintain network rules on node
- Container runtime: this component runs the container; the most famous example for container runtime is Docker.

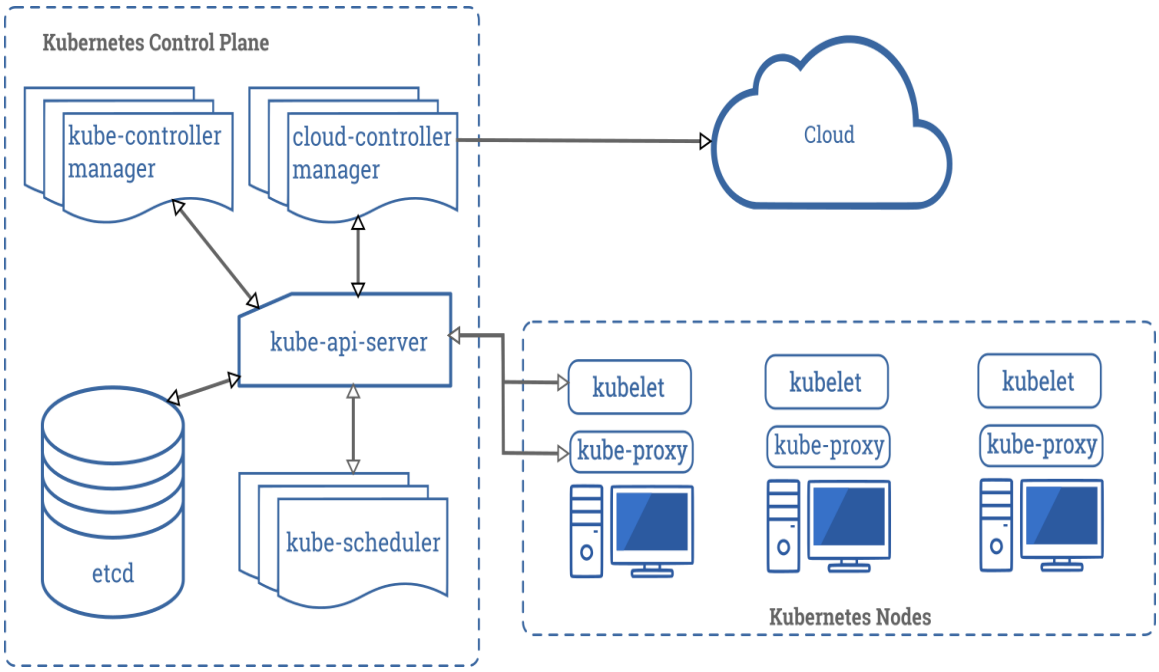

Figure 9 - Kubernetes architecture[11].

iii. Kubernetes networking

Kubernetes networking considers a kind of complex issue. There are five main scenarios for Kubernetes networking, as in **Figure 10**.

The communication between two containers is solved by the communication between Pods and the localhost. The outworld can communicate with pods through Kubernetes services.

The Kubernetes service is: “An abstract way to expose an application running on a set of Pods as a network service.”, there are four types of Kubernetes service, and each type help to solve kind of networking case:

- ClusterIP: In this mode, the service reachable inside the cluster, and it is the user-internal IP.
- NodePort: Use to make the service reachable from the outside world.
- LoadBalancer: Use cloud service provider load balancer to expose the service externally.
- ExternalName: This type maps the service to the contents of the external URL[11].

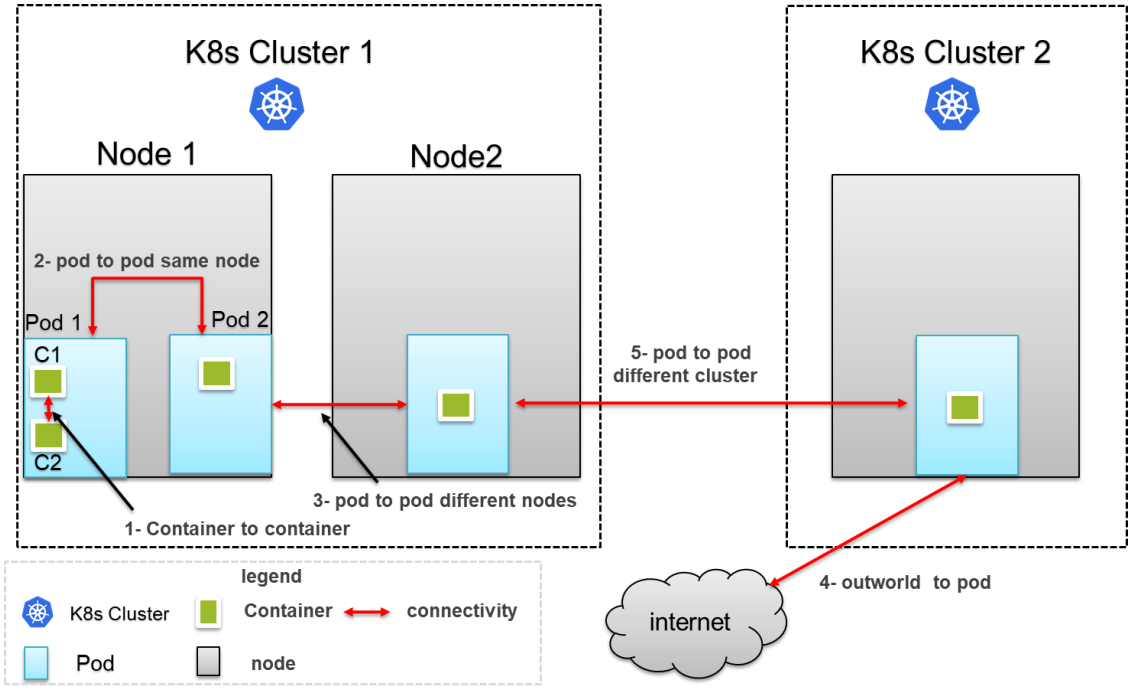

Figure 10 - Kubernetes networking scenarios

Pod to pod communication is the main focus, each Pod has its own IP address, and there are many solutions to solve Pod-to-Pod communication.

Container Network Interface (CNI) is a cloud-native project used to configure the container interfaces; according to the official CNCF website and Kubernetes website, more than 25 CNI solutions are serving the cloud-native design.

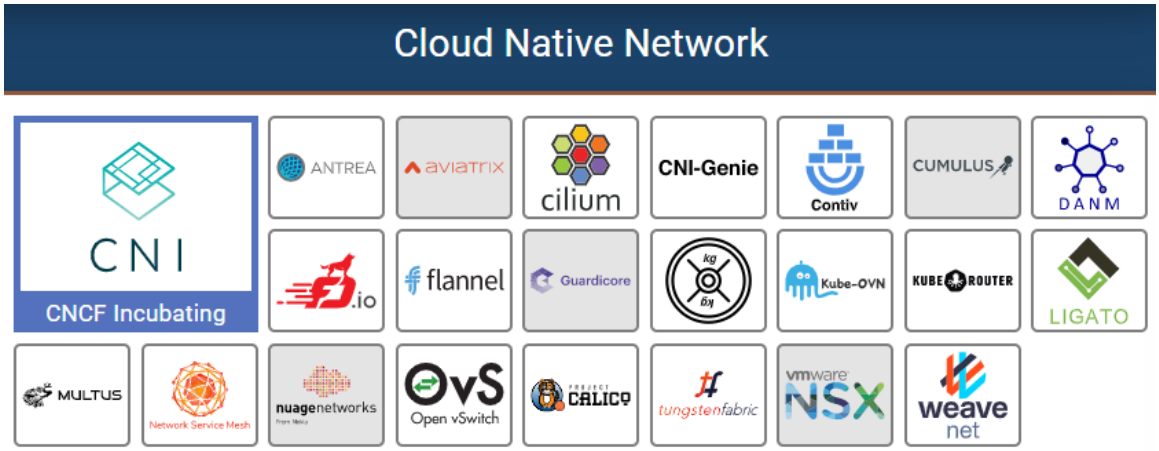

Figure 11 - Cloud-native networking solutions[9]

4- Mobility Management

a. Introduction

Mobility management aims to provide continuous service to the roaming devices when moving from one point of attachment to another.

Many mobility management solutions are introduced in the last decades; the categorization follows many types: the operational layer, the granularity of the service, pure end-to-end solution, etc..., as in **Figure 12**[3].

Most of the real-life solutions for mobility management are L2 solutions, and they lack seamless mobility management on the IP level this gap could be eliminated by using IP mobility management solutions.

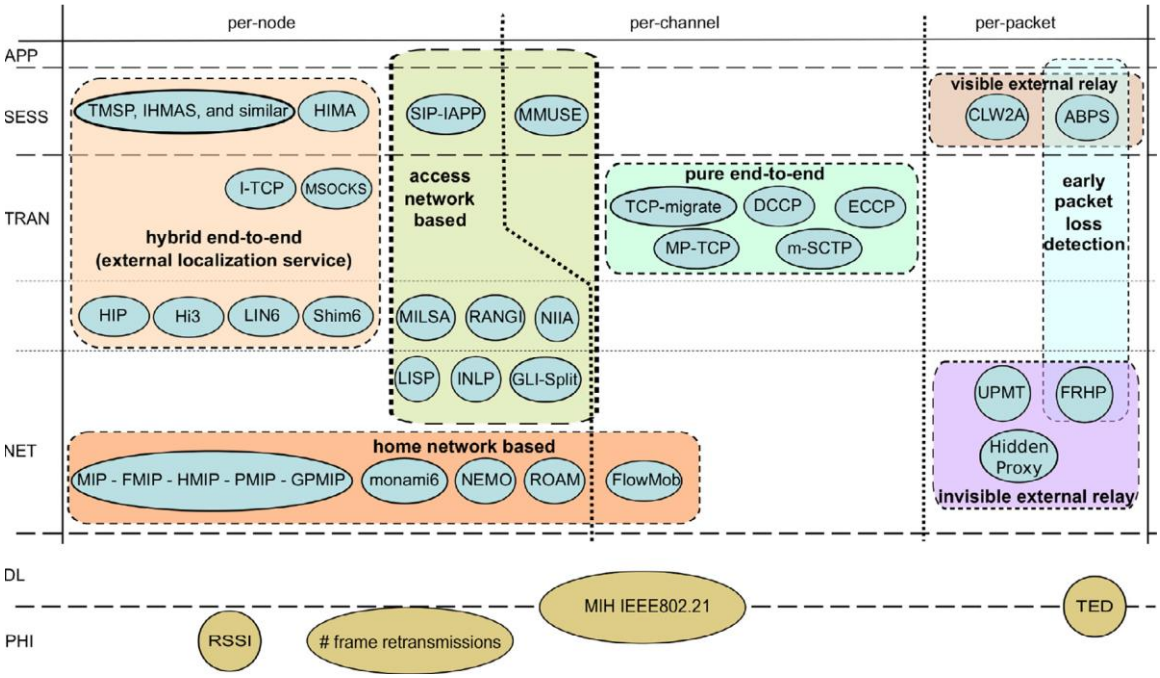

Figure 12 - Mobility Management approaches[3]

b. Mobility requirement

Essential functions, performance, and deployment requirements are needed from any mobility solution.

The essential functions of mobility management are to provide Handover Management, which aims to keep the communication up during MN's movement. Also, the Location Management function is mandatory, and this function should provide the MN current location and keep tracking the location during movement. Besides, supporting current services and applications and take care of security perspectives are basic functions, too. [2][29][32]

For the performance requirements, handover latency, packet loss, signaling overhead, and throughput are the most critical mobility management [2]

Some deployment requirements should take into consideration when deploying a mobility management solution. The mobility solution should integrate into the current infrastructure smoothly; it should avoid adding a third-party device in the network and minimize the application change [2].

c. Mobility management solutions in mobile network and WAN

In mobile networks, the core nodes control the handover process in interaction with the Radio Access Network (RAN) network [3].Diving into details is out of our work scoop, however it worth mentioning some examples.

In the GPRS network, the handover mechanism for the MN when moving from one BTS to another under the same SGSN is called radio handover; this kind of handover is layer 2 and does not affect the IP routing. In general, all the BTSs in the same routing area (RA) are connected to the same SGSN. In GPRS, the packet rerouting is the IP handover method, and that happens when the mobile node moves from one SGSN to another, a new GTP tunnel created between the new SGSN and the GGSN[30].

The core mobility management node in the LTE network called Mobility Management Entity (MME), and the handover is managed by Universal Terrestrial Radio Access Network (EUTRAN) [3]; in LTE set of various technologies interact together to support the handover process, the MN and the interconnected wireless network work together to provide the best possible handover process, the mobility management process places in L2 (data link layer), L3 (network layer), and cross-layer (L3 + L2), the l2 mobility occurs when the MN moved between different access nodes however the point of attachment to IP network doesn't change. At the same time, the l3 handover involves the change of IP addresses[38].

In [5] a cloud-native MME architecture is proposed, the MME VNFs built based on microservices architecture using Kubernetes platform, they proved that the design is scalable and the scaling is seamless and straightforward using the auto-scaling mechanism, besides simplicity in the deployment. The measurement results show that the cloud-native design decreases the processing resources' usage by 27% compared to legacy MME. The cloud-native MME provides 7% higher MME throughput.

In a wide area network (WAN) when the MN find a new access point (AP) has better signal strength, it breaks the connection to the previous one. It sends a reconnection request to the new AP, the new access point create the connection with the MN and inform the previous AP about transferring the MN to the new AP, and finally, the MN moved to the new Ap, knowing that this is a layer 2 handover[3].

d. IP mobility management

Ip mobility management is a perfect candidate to provide seamless mobility to the moving devices.

When the MN moves between two network domains, the mobility called macro-mobility, and the moving between two subnets in the same domain is called micro-mobility[32].

The first IP mobility problem in the network layer was that the IP address refers to the node and its location, which is solved in the MIP proposal[30].

MIPv4 has many drawbacks. According to **Figure 13,** the world moves toward IPv6 contents, so MIPv6 should be used instead. The current design of MIPv6 lacks the flexibility that could be provided by cloud-native design.

We will refer to the moving device as a Mobile Node (MN) and the destination device as a correspondent node (CN).

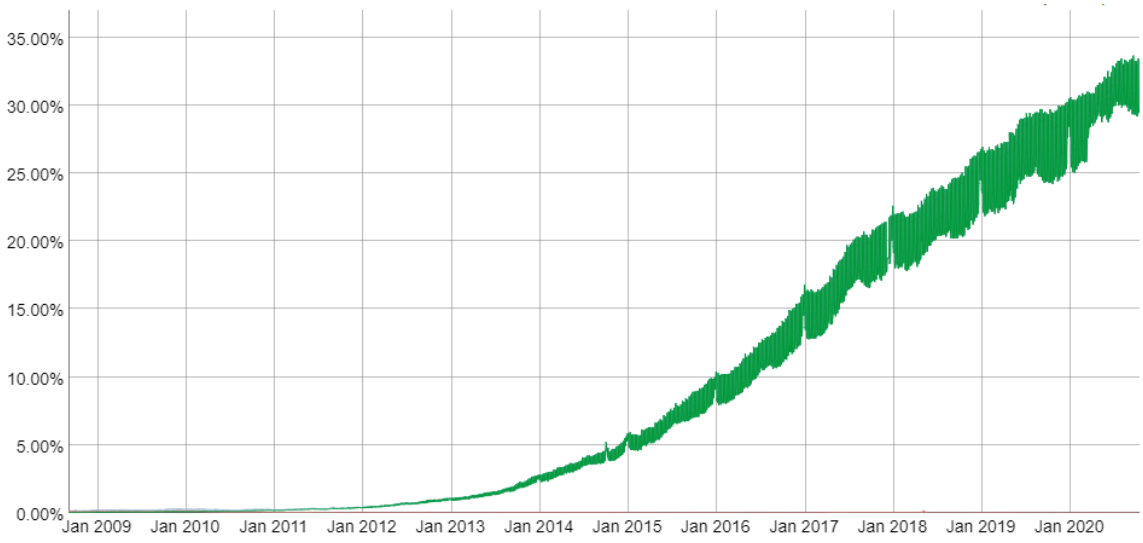

Figure 13 - IPv6 Adoption based on Google (the percentage of users that access Google over IPv6) [37]

i. Host-based vs. Network-based

Mobile IP (MIP) and its two versions (MIPv4 and MIPv6) are the most popular IP Mobility protocols. The MIP is a host-based mobility protocol. In MIP, the host can utilize two addresses: a permanent address called Home of Address (HoA) and a temporary address called Care-of Address (CoA). In host-based mobility management, the modification is done from the host side, and a new CoA assigns to the MN after each attachment to a new IP subnet.

In network-based mobility management, all the modifications are done from the network side without any change from the MN side; using network-based mobility management reduced the signaling overhead in the wireless channel, leading to efficient use of wireless resources enhancing handover performance[29].

ii. MIPv4

In MIPv4, a permanent IP address called Home of Address (HoA) identified the MN, and each time the MN moves to a foreign network, a temporary IP address called Care-of Address is assigned to the MN. The home network refers to the first network that the MN is attached to it, and the foreign network is the network that the mobile node moves to it. Two main entities introduced in the MIPv4, Home Agent (HA) in the Home Network (HN) and Foreign Agent (FA) in the Foreign Network (FN)[2][31].

The handover management in MIPv4 started when the MN moved to a new location, the Mn obtaining new CoA in a foreign network and resister this CoA with HA, The HA tunnel all the

packets from the CN toward the MN. The MN sends the packet directly to the CN, creating a triangle routing problem[31] [32].

In addition to triangular routing, the MIPv4 is also affected by high signaling load and high handoff latency[32].

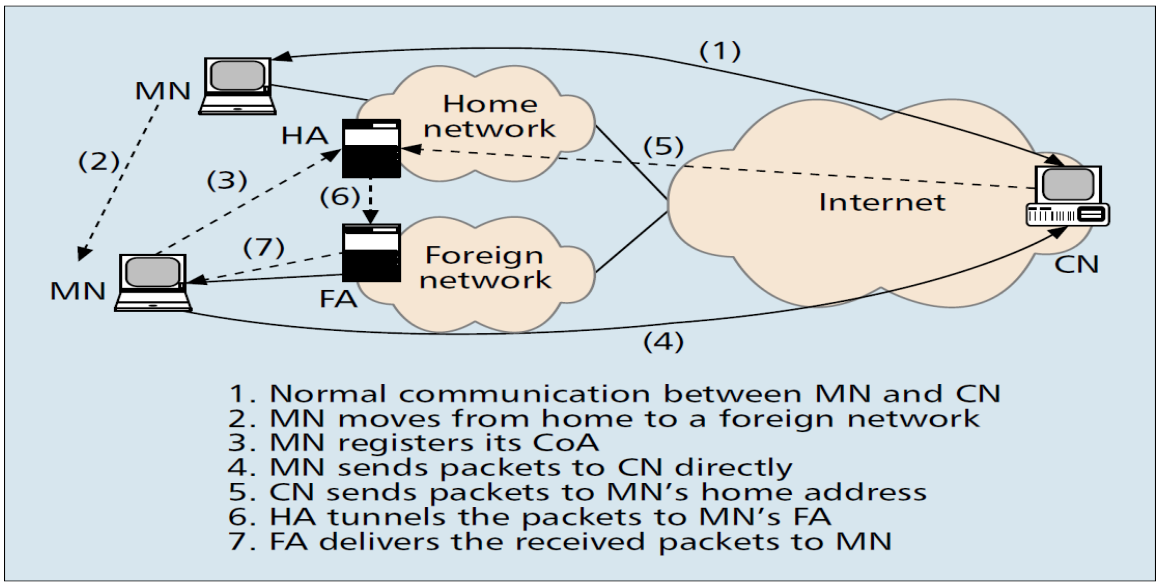

Figure - 14 MIPv4 architecture[2]

iii. MIPv6

MIPv6 is an enhancement of MIPv4. The Route optimization used by MIPv6 is supported by default. It solves the triangular routing problem and improves security[32]. The foreign agent's role is eliminated and replaced by any access router (AR), which can apply MIPv6 in any location without a particular need from the router[1].

MIPv6 architecture is described in **Figure 15**.

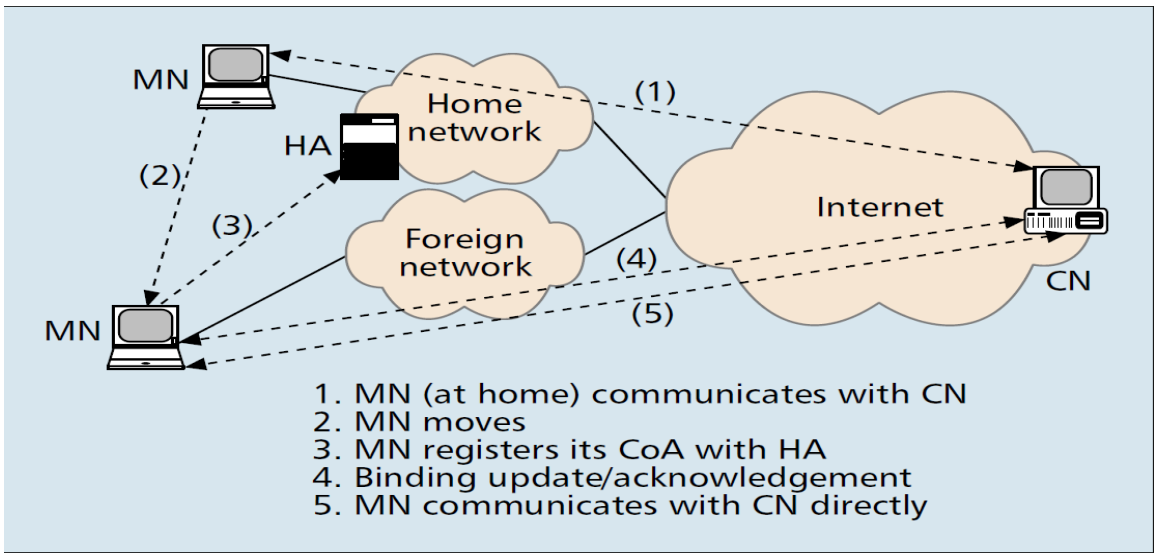

Figure 15 - MIPv6 architecture[2]

The HoA is the IP used to identify the MN in the home network and the traffic routed to this IP when the MN in its home network. When the MN changes its point of attachment to a new location away from the home network (foreign network), the MN obtains new IP called CoA. The MN get the CoA using the auto-configuration mechanism provided by IPv6. The MN received packets at CoA while it's located in this FN.

After moving to FN, the MN sends Binding Update(BU) message to HA to register the CoA, and the HA replies with Binding Acknowledgement (BA) message.

MN and CN communicate with each other in two modes. The first mode is bidirectional tunneling. In this mode, the CN directs the packets to HA, and the HA tunnel the MN packets. The second mode is the route optimization mode, and here the MN registers its current binding information at the CN (including its CoA). The CN direct the packets to the MN using CoA as a destination, each CN save the Binding Cache (BC) and later on check the BC entries, and in case of find MN record in this BC the CN use new header type called "Type 2 Routing Header", this header allow the packets to be routed directly to MN. MN adds a new IPv6 "Home Address" destination option to carry its home address.

In MIPv6, a new extension header called "Mobility Header" was used to manage the signaling Between MN, CN, and HA. This header carries the Binding Update (BU), Binding Acknowledgement (BA), Binding Error, and many other signaling messages.

HAs and CNs maintain the Binding Cache. The BC contains both "correspondent registration" entries and "home registration" entries

Each Binding Cache entry correspondent contains HoA, CoA, and some other controlling values.

While the HA maintains Home Agent List beside the binding cache[1], this list contains information about all other HAs on the same link, and it is used by the dynamic home agent address discovery mechanism[1].

The MIPv6 signaling messages are illustrated in **Figure 16**.

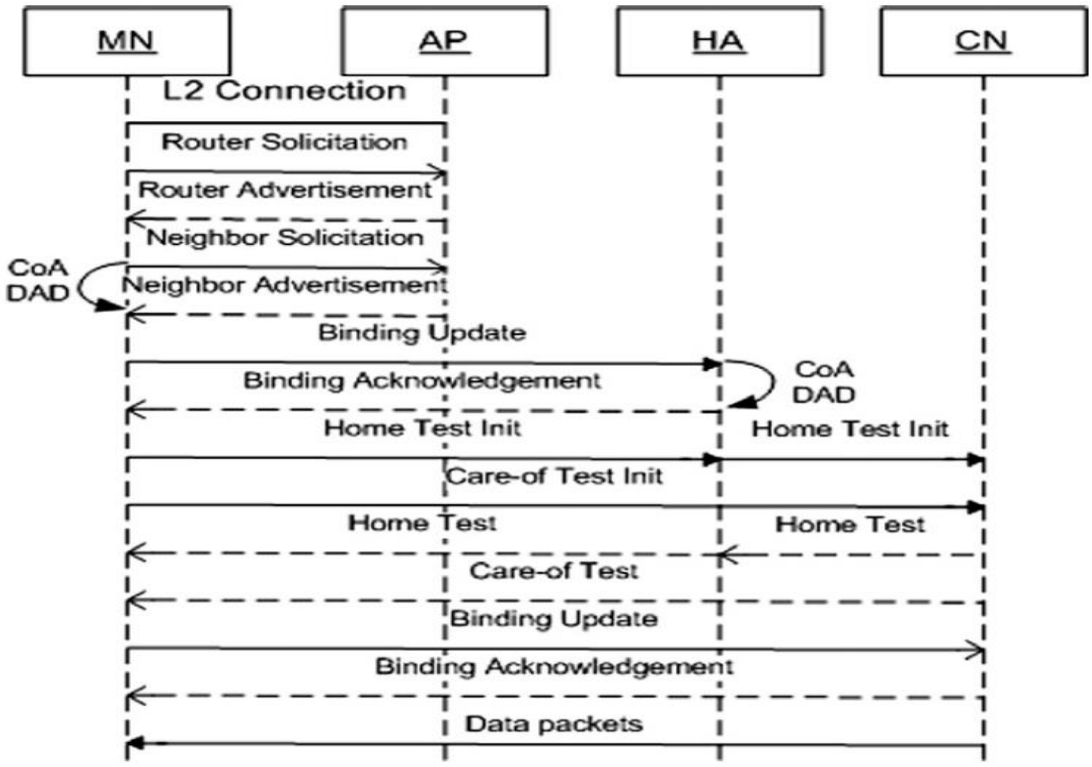

Figure 16 - MIPv6 message flow[39]

iv. Network-Based Localized Mobility Management

In local mobility, mobility management solutions aim to provide IP mobility within an access network. **Figure 17** sums up this issue. When the mobile node moves between two access points under the same access router, L2 mobility called intra-link happened. When the MN moved between two access nodes under two different access routers, local mobility occurs. The local mobility does not cover the moving between two access networks, and here the global mobility takes place and operates.

When global mobility occurs, three main problems are introduced:

- Update latency: some packets may drop if the CN and the global mobility anchor point have the same distance toward MN.
- Signaling overhead: the high amount of signaling messages when the MN moving to a new access network will affect the wireless bandwidth.
- Location privacy: the mapping between MN and its geographical location based on the subnet prefixes could expose the user location[46].

Consequently, Network-Based Localized Mobility Management (NETLMM) protocols should solve global mobility problems introduced before beside the following goals summarized in[8]:

- “Handover performance improvement
- Reduction in handover-related signaling volume
- location privacy

- Limit overhead in the network
- Simplify mobile node mobility management security by deriving from IP network Access and/or IP movement detection security
- Link technology agnostic
- Support for unmodified mobile nodes
- Support for IPv4 and IPv6
- Reuse of existing protocols where sensible
- Localized mobility management independent of global mobility management
- Configurable data plane forwarding between local mobility anchor and mobile access gateway"

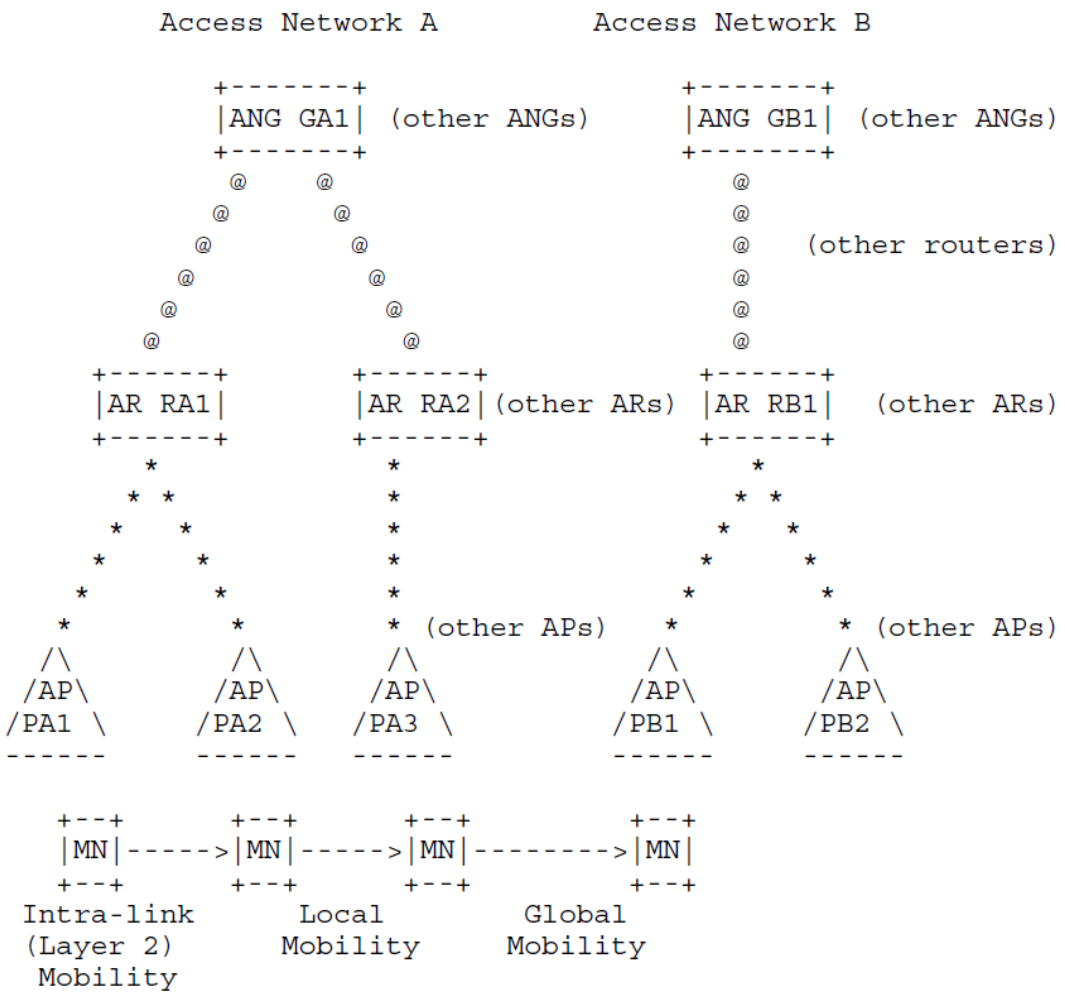

Figure 17 - Local and global mobility management [46]

v. PMIPv6

Proxy Mobile IPv6 (PMIPv6) [25] is a network-based mobility management protocol that works at the network layer. In PMIPv6, the network elements perform the mobility signaling instead of the MN.

The essential components of PMIPv6 are the Local Mobility Anchor (LMA) and the Mobile Access Gateway (MAG). The LMA works as the HA in MIPv6. It assigns a home network prefix (HNP) for the MN, tracks the MN location, and responsible for creating tunnels between the MAGs and LMA, Binding Cache Entries (BCE) stored in the LMA and it contains each registered MNs. In PMIPv6, all the packets between the MN and the CN are going through the LMA[25][27].

Usually, the MAG represented by an access router (AR), the MAG performs the mobility management signaling on behalf of the MN, it detects the MN movement and attachment, and forward data packets to the LMA through IP tunnel[25] [27].

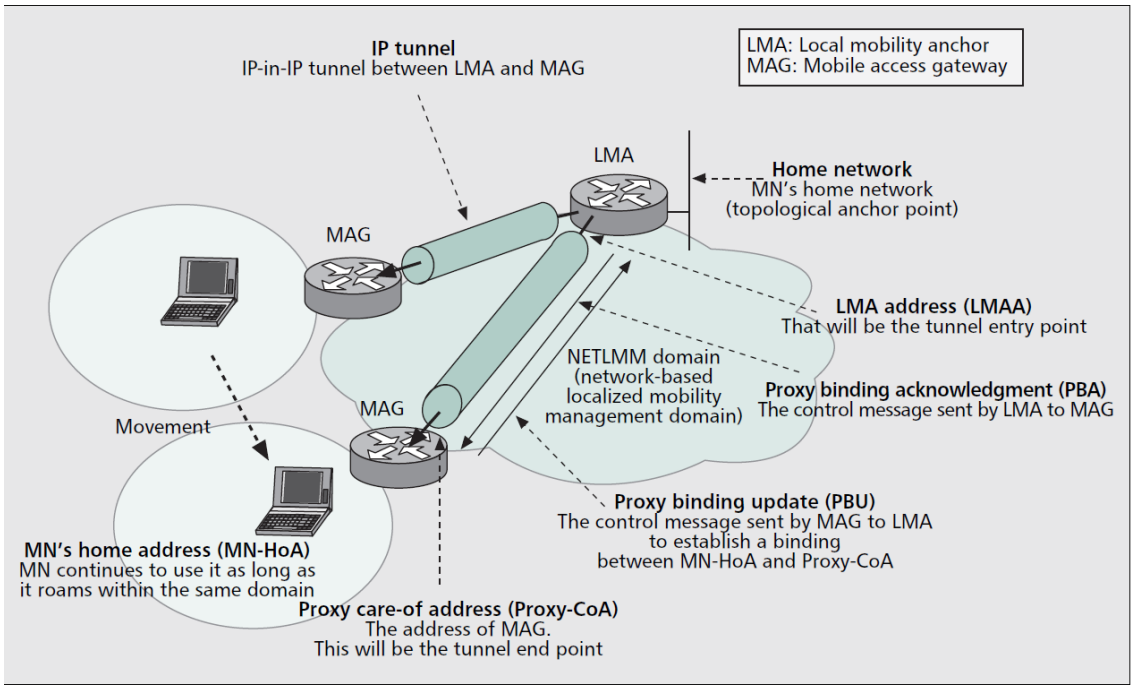

Figure 18 - PMIPv6 architecture[26]

PMIPv6 assume that policy store server like authentication, authorization, and accounting [AAA] server existed in the network.

When the MN attaches to the network for the first time, the MAG performs an authentication check with the AAA server using the MN identifier; after the successful authentication, the MAG obtain the MN's profile from the AAA, then the MAG sends Proxy Binding Update (PBU) message to the LMA informing it about the new attachment, the LMA updates its binding cache and sends back Proxy Binding Acknowledgement (PBA) to the LMA contains the MN's home network prefix (MN-HN) and Proxy-CoA which is the address of MAG-1. The bi-directional tunnel is established between the MAG and the LMA, and then the MAG sends router advertisement (RA) to the MN contains the MN-HNP. The data route will be from MN toward MAG, then to the LMA, and finally to the CN.

When the MN moves to a new location (new MAG), a de-attachment message sends from the old MAG to LMA via PBU message, and the LMA reply by PBA inform about the removal. All the previous procedure repeated between the new MAG and the LMA is different because the PBU contains the new Proxy-CoA of the new MAG[26][28].

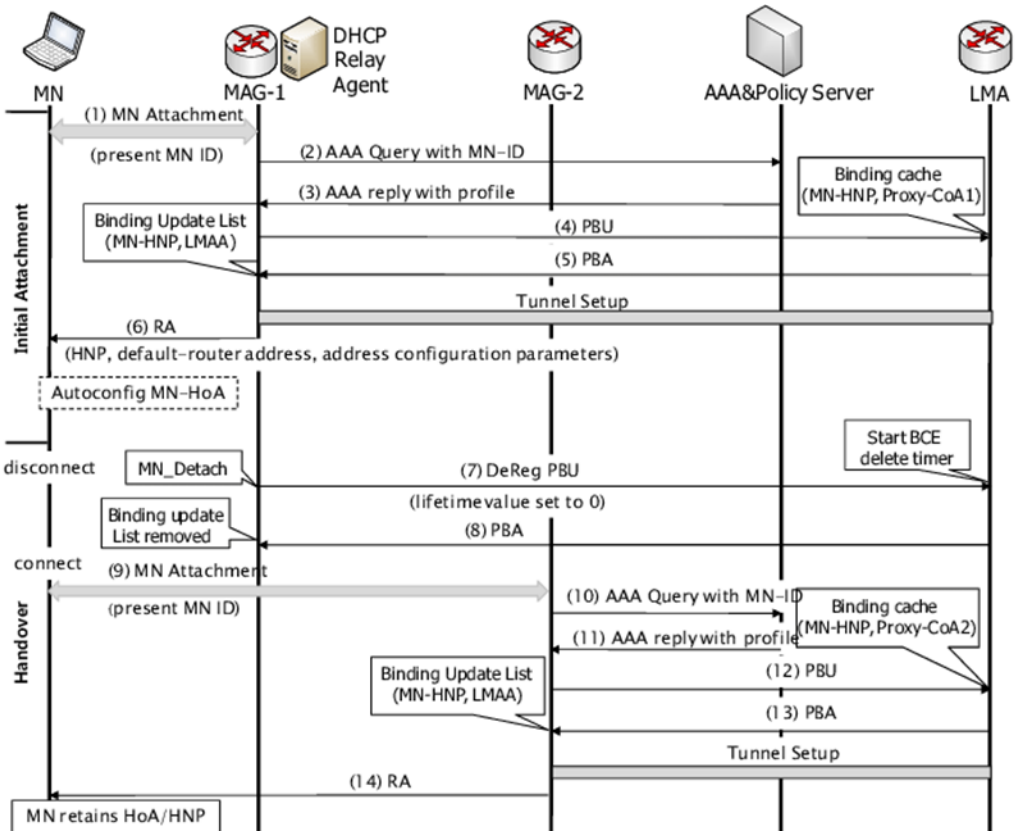

Figure 19 - PMIPv6 messages flow[28]

PMIPv6 has drawbacks because of tunneling overhead and overloaded LMA in the PMIPv6 domain.

Many promising approaches integrate PMIPv6 with SDN to solve the PMIPv6 limitations[27][41][42].

vi. Interactions between the MIPv6 and PMIPv6

The interaction between MIPv6 and PMIPv6 is possible in some scenarios. Some considerations should be managed for these solutions not to affect each other.

One scenario is implementing the MIPv6 as a global mobility management protocol and implementing PMIPv6 as a local mobility management protocol and no special requirement for this scenario.

Another scenario is to move from access network support network-based mobility solution like PMIPv6 to access network support host-based mobility solution (mainly MIPv6), for this scenario three issues should be identified, firstly, two BCEs should be created for the same MN in the LMA and HA as the structure is different, secondly, when the MN moved from MIPv6 foreign network to PMIPv6 home domain, the MN de-registration message shouldn't delete the BCE in the PMIPv6, thirdly, the race and the messages sequence between BU and PBU is an obvious

issue, it is essential in this scenario to treat the two protocols as independent in HA/LMA implementation.

Finally, there is a scenario where some MNs served by MIPV6 and the others served by PMIPv6[6]. In this scenario, a central node work as HA and LMA exists, and this scenario could be a combination of the first two[6].

vii. Distributed Mobility Management

One step forward in Mobility management development is the Distributed Mobility Management (DMM). The idea of DMM is based on distributing the mobility functions over the network, so instead of having a centralized mobility management entity serving all MNs in the network, multiple mobility functions will be distributed, and each MN will be served by the nearest mobility management function.

Using DMM distributes the load over the network, eliminates the single point of failure of the anchoring point (LMA in PMIPv6 and HA in MIPv6), and reduces the latency as the nearest anchor serves the MN[44].

DMM could be partially distributed. In this case, the data plane is distributed. To apply the partially distributed mobility management, the control plane and the data plane should be separated in the protocol structure, and that is not applicable in MIPv6 and PMIPv6 in their basic design as all the data and signaling messages go through HA/LMA, the data plane, and control plane have different properties, separating the data plane and the control plane and distributed the data functions based on that is the idea of partially distributed mobility management.

On the other hand, in fully distributed mobility management, the mobility control plane and the data plane could be distributed[4].

The distribution could be at many levels, the mobile core level, the access network level, and the host level:

- The mobile core level means distributing the core element in the mobility management, in case of MIPv6, it is distributing the HAs, and in case of PMIPv6, the MAs are distributed. With core distribution, the load on the anchor point is divided into the network. The single point of failure is eliminated. Each anchor will cover a certain geographical area; moving between these areas should be managed. The anchors' communication should happen to inform about the movement.
  distributing the MAG/AR/FA could be a core distribution, or these functions could be handled by the access layer
- Distributing the access network happened in the access points level; in general, these APs have the l2 capability. Adding the l3 capability will provide this level to handle the l3 mobility management, and they could work as MAG or FA.
- The host-level distribution is more like a peer-to-peer solution; when the MN find the CN, they will communicate directly; an information server like DNS server is needed. The MIPv6 in route optimization mode provides host-to-host communication[4].

An example of DMM of MIPv6 described in **Figure 20**, multiple HAs are distributed over the network, and each MN will be served by the nearest HA [44].

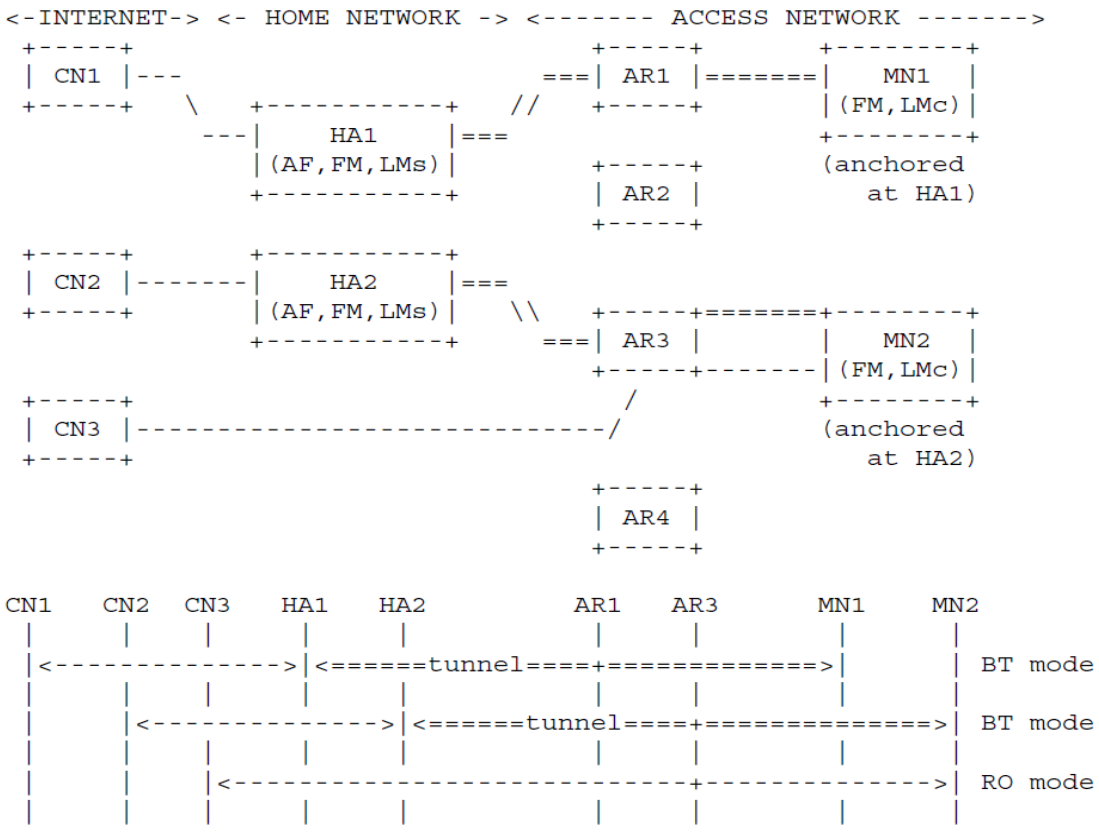

Figure 20 - Distributed Mobility Management in case of MIPv6[44].

5- Cloud-native MIPv6 architecture proposal

a. Overview and the proposal design

The overview of the high-level architecture for cloud-native MIPv6 is illustrated in **Figure 21**. The design follows the same structure as of the MIPv6 architecture described before in section **4-d.iii**. However, the HA is carried within the cloud. In this design, we do not expect any change of the MIPv6 structures and messages; the mobility process started when the MN moved to a foreign network, the HA responsible for all mobility operation. The HA is built inside a container based and cloud-native architecture.

Kubernetes has very powerful mechanisms of self-healing and detecting errors, these mechanisms are Readiness and Liveness probes which are used to ensure that traffic does not reach a container that is not ready for it and restart the containers in case of failure. In addition, one big advantage of using Kubernetes is the Horizontal Pod Autoscaler, this autoscaler monitor the PODs loads and increase or decrease the resource accordingly, this scaling could be adding new copies of POD or increase the resources of the POD, as a result the HA is always available, has the needed resources, and handled in a flexible manner.

The following improvements are provided from this design:

- In case of any failure, restoring the HA is fast and easy.
- The redundancy mechanisms are very advanced and convenient.
- High level of availability.
- The scaling mechanism is mature and automated.
- The resource usage is more efficient.
- This design could be applied to any cloud anywhere.

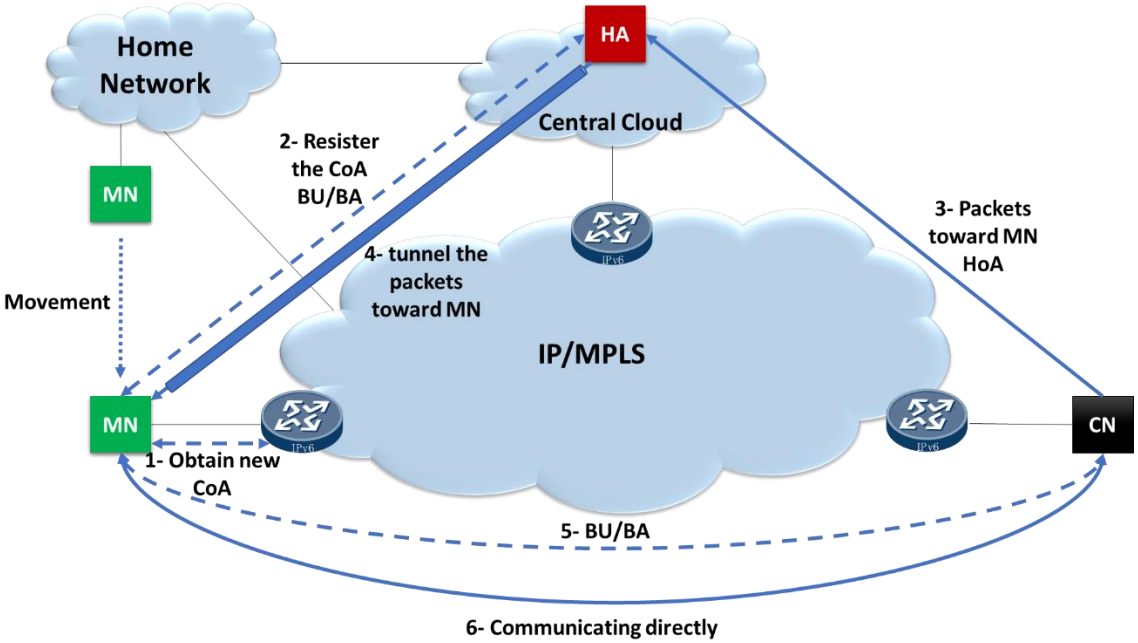

Figure 21 - High-level architectural proposal of cloud-native MIPv6

The low-level architecture design for the proposal is described in **Figure 22**. In the central cloud, we have the following layers:

- Physical hardware layer: the physical resources that form the cloud maintained in this layer; basically, the hardware consists of servers, storage, and networking devices. Nothing happened to the packets in this layer and no need for unique solutions rather than legacy connectivity. Later on, this layer will form the cloud infrastructure.
- Cloud infrastructure layer: In our case, it is the OpenStack layer. This layer responsible for creating and manage the VMs that will be used later from the upper layer. Networking solution should be treated here; OpenStack mostly used VXLAN technology for the communication, which will add l2 header to the packets.
- Containerization layer: Kubernetes operate in this layer, and it is responsible for creating the PODs, which will carry the HA service; dealing with the networking in this layer is a challenge; we will introduce some proposals in section **6-b**. In general, the Kubernetes networking solution should support the cloud-native design, support IPv6, and provide the required networking infrastructure to support the HA appropriately.

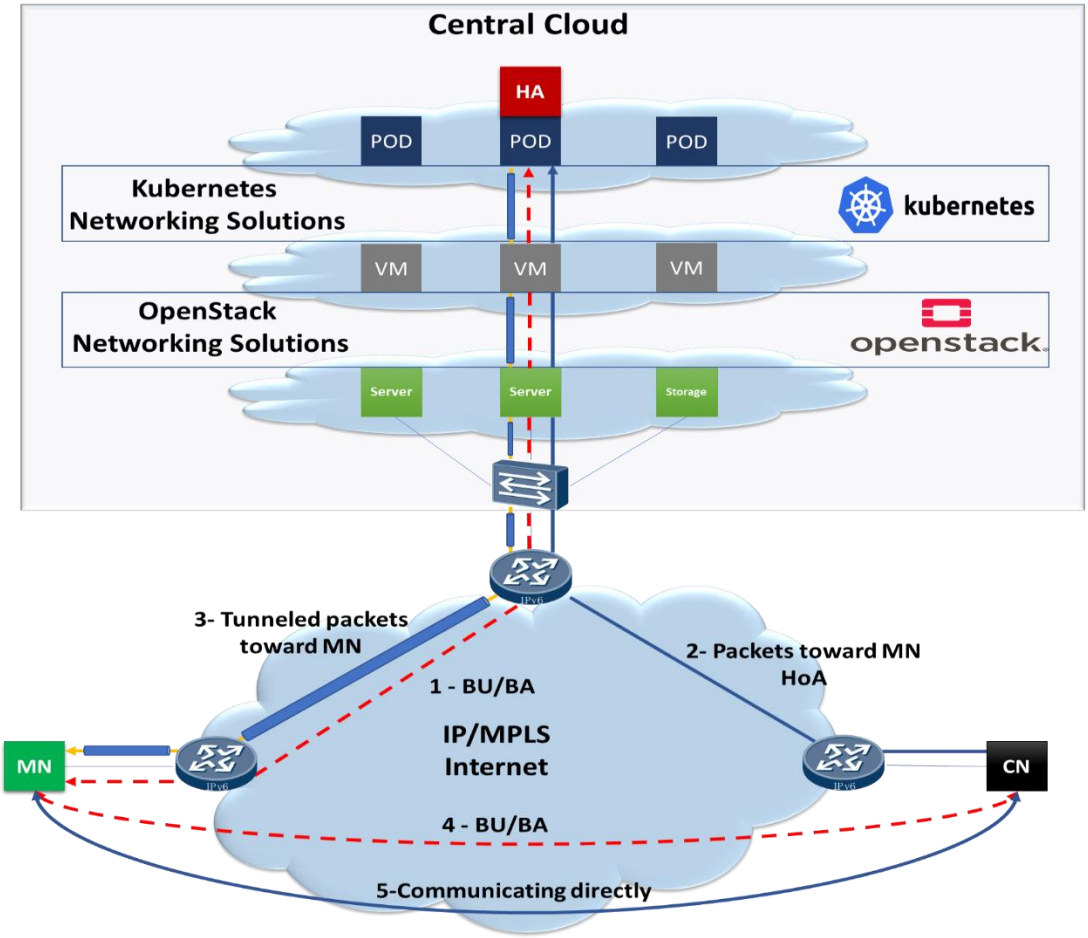

Figure 22 Low-level architectural proposal of cloud-native MIPv6

Reaching this design will give high flexibility in dealing with MIPv6 architecture as all the cloud-native and cloud techniques could be used to increase the reliability, availability, and performance.

The message flow of cloud-native MIPv6, illustrated in **Figure 23**:

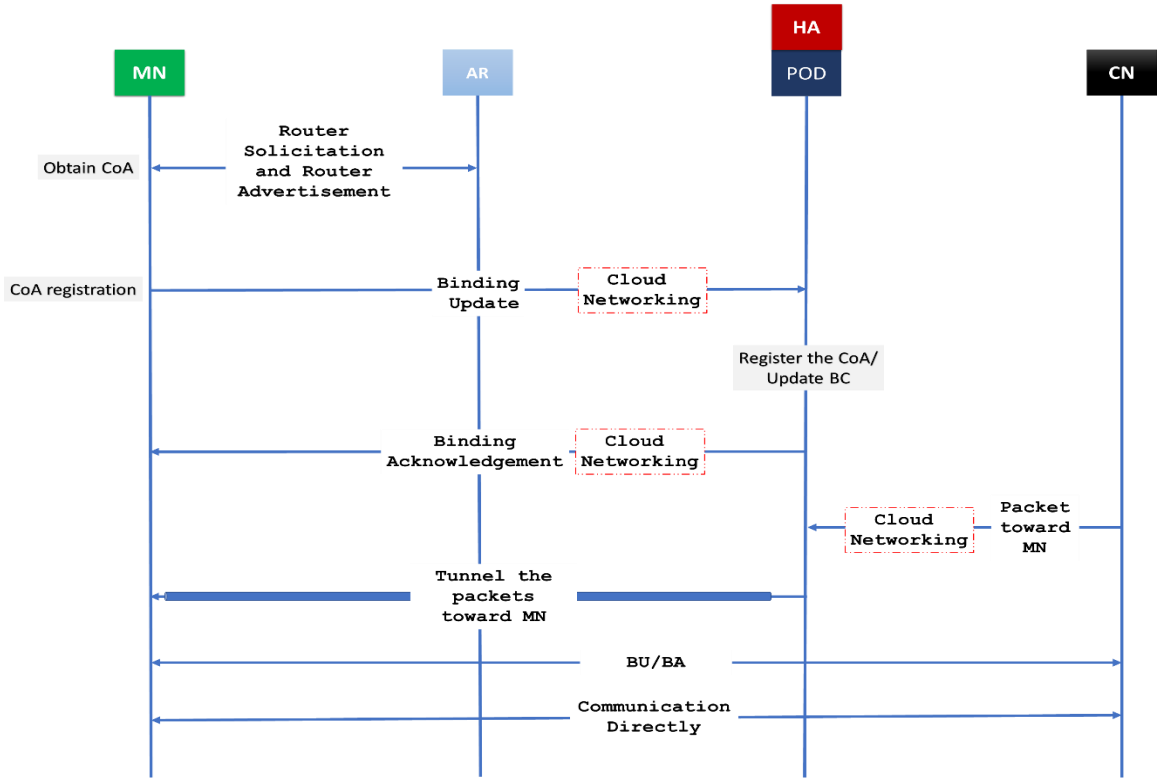

Figure 23 - Cloud-native MIPv6 message flow

b. Advance design involving SDN

In this case, the tunneling process is triggered by the home agent binding cache **Figure 24**; however, it is done by the SDN controller. The HA keeps the binding cache and all mobility information.

When the MN moves to a new foreign network, it obtains a new CoA and sends a binding update to the home agent. When the HA receives the BU, it sends back a binding acknowledgment to the MN, and update the binding cache by resister the new CoA.

Simultaneously, a third party application reading the binding cache, when the HA registers a new CoA in the binding cache, it triggers an application. This application communicates with the SDN controller through API on the northbound interface and it is responsible for adding new roles about the new information to the SDN controller.

The SDN controller creates a new IP-in-IP tunnel, and it adds a new flow to the responsible router. This flow contains this information: if packets routed toward MN HoA tunneled it directly to the already created tunnel without going to the HA, this will decrease the tunneling header through the cloud.

Merging Cloud-native design of MIPv6 with SDN architecture reduce the load over the HA, provide more flexibility to the network, and provide end-to-end automation possibility.

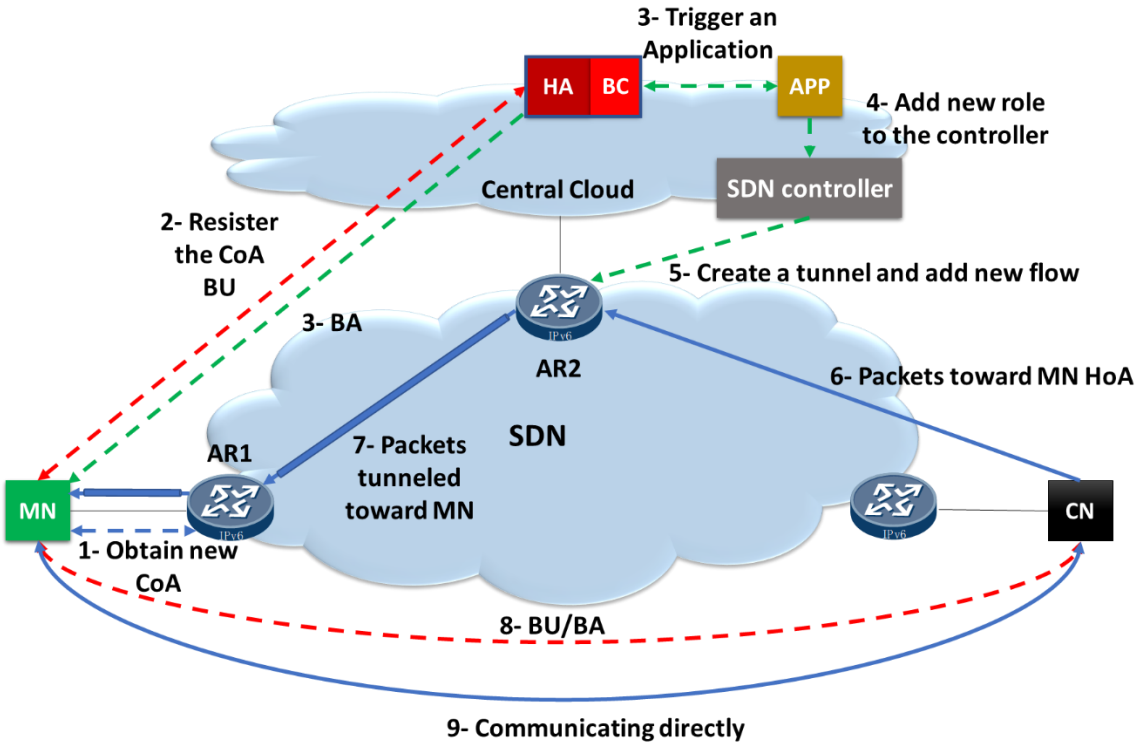

Figure 24 - Cloud-native MIPV6 in SDN environment

The messages flow in this case described in **Figure 25**:

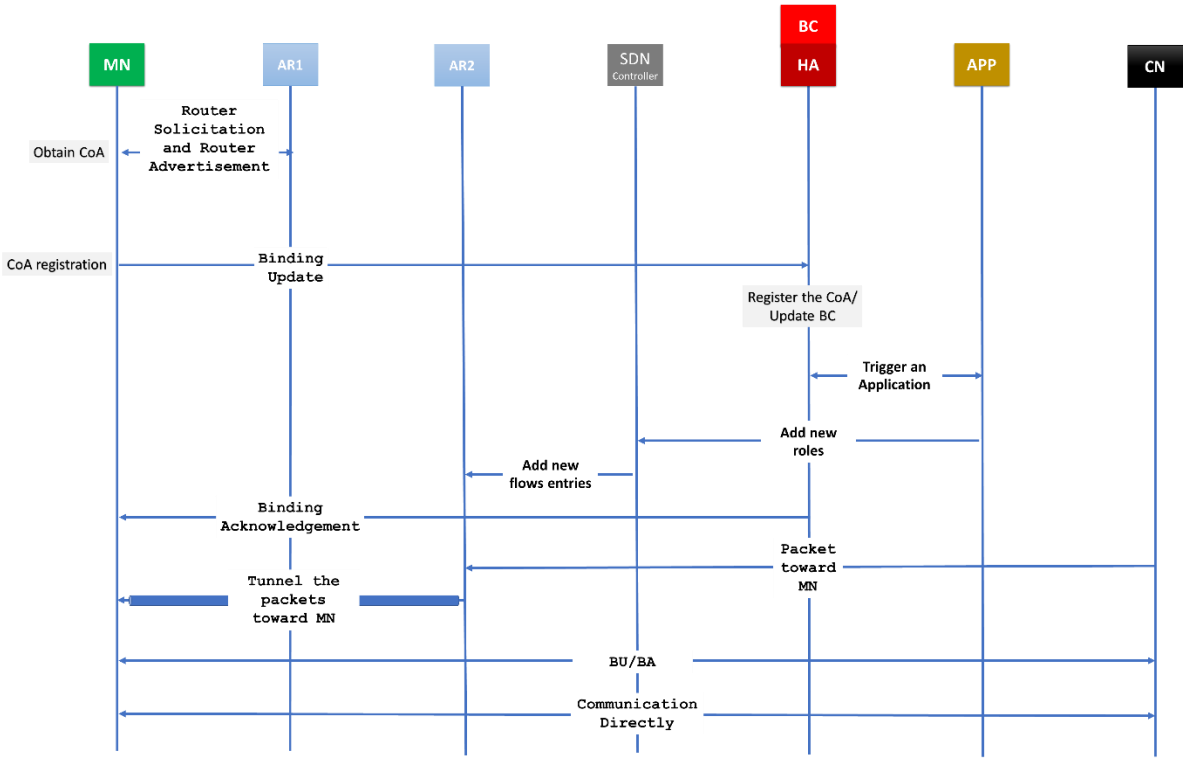

Figure 25 - Message flow of Cloud-native MIPV6 in SDN environment

6- Testbed

a. Cloud-native design

To test the MIPv6 as a cloud-native application, we built a container carrying UMIP service.

UMIP is "open-source (GPLv2) Mobile IPv6 stack for the GNU/Linux Operating System"[43].

The environment described in **Figure 26** consists of one OpenStack VM carrying Docker container and the MIPv6 service deployed inside the container.

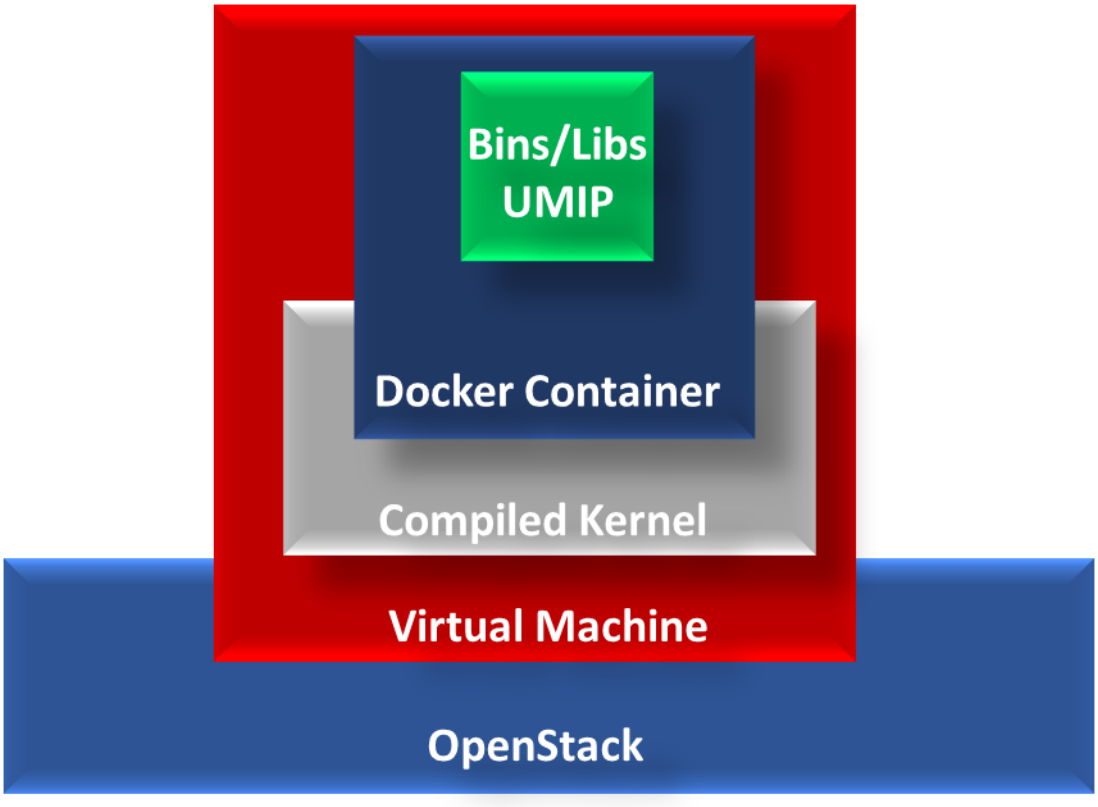

Figure 26 – High-level concept design of implementing MIPv6 service inside Docker container

We will exclude the part of preparing OpenStack and the VM as it is not our focus.

MIPv6 take part in two position Kernel space and userspace. The kernel space responsible for IPv6 header extension handling while the userspace responsible for all MIPv6 logics[45].

Kernel space preparation:

some modules should be loaded to the kernel to support MIPv6.

The steps for doing that:

1- download the source code of the kernel, in my experiment, I choose the kernel version 4.15:

```
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.14.15.tar.gz
```

extract the file:

```
unxz -v linux-4.14.15.tar.gz
tar xvf linux-4.14.15
```

2- Configure the Linux kernel features and modules:

```
cd linux-4.14.15
cp -v /boot/config-$(uname -r) .config
```

3- install the needed tools and dependencies:

```
sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
```

4- configure the kernel:

```
make menuconfig
```

In this step a list of modules should be mark as "load" during configuration , the list is:

General setup

Networking support [CONFIG_NET]

--> Networking options

--> Transformation user configuration interface [CONFIG_XFRM_USER]

--> Transformation sub policy support [CONFIG_XFRM_SUB_POLICY]

--> Transformation migrate database [CONFIG_XFRM_MIGRATE]

--> PF_KEY sockets [CONFIG_NET_KEY]

--> PF_KEY MIGRATE [CONFIG_NET_KEY_MIGRATE]

--> TCP/IP networking [CONFIG_INET]

--> The IPv6 protocol [CONFIG_IPV6]

--> IPv6: AH transformation [CONFIG_INET6_AH]

--> IPv6: ESP transformation [CONFIG_INET6_ESP]

--> IPv6: IPComp transformation [CONFIG_INET6_IPCOMP]

--> IPv6: Mobility [CONFIG_IPV6_MIP6]

--> IPv6: IPsec transport mode [CONFIG_INET6_XFRM_MODE_TRANSPORT]

--> IPv6: IPsec tunnel mode [CONFIG_INET6_XFRM_MODE_TUNNEL]

--> IPv6: MIPv6 route optimization mode [CONFIG_INET6_XFRM_MODE_ROUTEOPTIMIZATION]

--> IPv6: IPv6-in-IPv6 tunnel [CONFIG_IPV6_TUNNEL]

--> IPv6: Multiple Routing Tables [CONFIG_IPV6_MULTIPLE_TABLES]

--> IPv6: source address based routing [CONFIG_IPV6_SUBTREES]

6- install the kernel modules:

```
sudo make modules_install
```

7- install the Linux kernel and update the GRUB:

```
sudo make install
sudo update-initramfs -c -k 4.14.15
sudo update-grub
```

After the reboot, the needed modules should be loaded to the kernel:

```
ubuntu@mipv6-compiled-module-salah:~$ cat /boot/config-4.14.15 | grep MIP6
CONFIG_IPV6_MIP6=y
ubuntu@mipv6-compiled-module-salah:~$ cat /boot/config-4.14.15 | grep
INET6_XFRM
CONFIG_INET6_XFRM_TUNNEL=y
CONFIG_INET6_XFRM_MODE_TRANSPORT=y
CONFIG_INET6_XFRM_MODE_TUNNEL=y
CONFIG_INET6_XFRM_MODE_BEET=y
CONFIG_INET6_XFRM_MODE_ROUTEOPTIMIZATION=y
```

Prepare the container:

Install Docker:

```
sudo apt-get install docker.io
```

run new container :

```
sudo docker run -itd --privileged --cap-add=ALL  -v /dev:/dev -v
/lib/modules:/lib/modules ubuntu:bionic bash
```

notes: for the correct deployment, the following point should be considered:

- the container should work in the privileged mode (--privileged)
- all the capabilities should be added (--cap-add=ALL)
- mount host /lib/modules into the container (-v /lib/modules:/lib/modules)

User space preparation inside the docker container :

1- install the tools and dependency:

```
apt-get install autoconf automake bison flex libssl-dev indent ipsec-tools radvd
```

2- download UMIP package

3- Compile and install the package:

```
autoreconf  -i
CPPFLAGS='-isystem /usr/src/linux/include/' ./configure –enable-vt
make
sudo make install
```

4 – run the package and check that the daemon is running inside the container:

```
root@bd47d409bcca:/umip# mip6d
root@bd47d409bcca:/umip# ps -ef | grep mip
root      6884     1  0 15:46 ?        00:00:00 mip6d
```

b. Advanced design

For large-scale design, we suggest using Kubernetes as a container orchestrator platform. We study some Kubernetes networking solutions to provide appropriate support for the mobility design **5-a**. We sum up the scheme in **Table 3**:

The implementations	Summary
Flannel[48]	- Simple and easy way to configure a layer 3 IPv4 network fabric designed for Kubernetes. - Easy to deploy. - Support many backends. - It does not fit our study because it does not support IPv6 yet.
Network Service Mesh[46]	- It adds flexibility, simplicity, and new features to Kubernetes networking. - It maps the concept of a service mesh to L2/L3 payloads. - It works by using a simple set of APIs designed to facilitate connectivity. - Deployed and tested and could be used with other CNI for our purpose.
Multus[50]	- Multus is a container network interface (CNI) plugin for Kubernetes that enables attaching multiple network interfaces to pods. Typically, in Kubernetes each Pod only has one network interface.
DANM[49]	- DANM provides multiple interfaces to the Kubernetes pods either through its own CNI, or through delegating the job to any of the popular CNI solution e.g., SR-IOV, Calico, Flannel, etc. in parallel - a CNI plugin capable of provisioning IPVLAN interfaces with advanced features
- Multus and DANM are very useful because they add new interfaces to the Pod, which can be used for mobility purposes; we assume that these CNIs will be involved in our future experiments and design.	

Table 3 – Kubernetes networking solutions.

The detailed architecture of advance design illustrated in **Figure 27**:

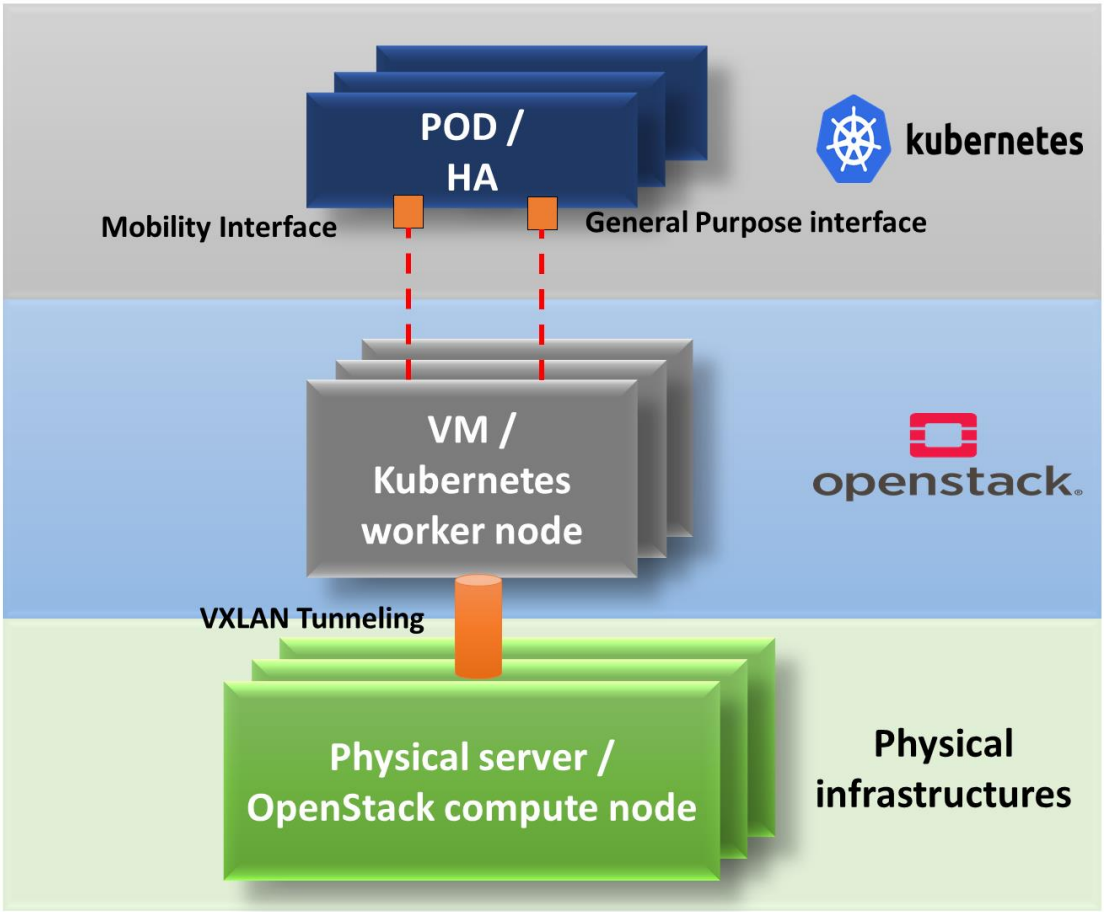

Figure 27 – Testbed architecture of the advanced design.

Conclusion

In this work, we introduced the evolution and the overall idea of the cloud-native approach; we started by studying the microservices development and the cloud-native transformation steps and providing statistics showing the importance and the high adaptation of cloud-native architecture in the new era of technology.

Then we discussed some hot topics and technologies that affect the network structures these days, including NFV, SDN, cloudification, and containerization. We summarized the benefits of using these technologies. Also, we covered the architectural and the essential terms in each one; besides, we introduced OpenStack, the most popular cloud platform in the telecom world, and Kubernetes as a containers orchestration platform, and how these platforms will involve in the cloud-native design.

The second part of this work is the IP Mobility Management evolution. We covered the overall idea of mobility management and mobility management requirements. We studied the IP mobility management solutions and the protocols beyond that, including MIPv4, MIPv6, PMIPv6, and the interactions between these protocols. We presented one more evolution jump approach in mobility management called Distributed Mobility Management before moving to cloud-native mobility management.

We took MIPv6 as the main research topic and introduced architectural proposals. The first proposal is building the MIPv6 protocol as a cloud-native design. This design involves using OpenStack and Kubernetes platforms. We summarized the advantages of this design and illustrated the message flows. The second proposal involved SDN in automating the network process, improving the performance, and reducing tunneling overhead. Moreover, the most serious problem of MIPv6 is the Single-Point-of-Failure of the HA is eliminated in our cloud-native designs, besides increasing the reliability, availability, and performance of the protocol.

Finally, we tested the implementation of MIPv6 as a cloud-native application. The experiment was done over OpenStack VM carrying Docker container, we deployed MIPv6 service inside a container, and introduced the steps and the needed change to build this environment, furthermore in the experiments part we evaluate some networking solutions that could be used later in advance experiments.

Future work

This work is prototyping and proof of concept of Cloud-native IP Mobility management, which will open the horizon for many experiments and tests.

First of all, a further real test could be done using UMIP over Docker and emulating a whole mobility scenario with configuring all nodes. Other MIPv6 packages could also be tested.

Moreover, fit the previous design in a high-scale environment, including automating the process using Kubernetes. Later on, an external orchestrator could be involved, distributing the HA functions over multiple pods, and managing this distribution could an experiment of DMM MIPv6.

Last but not least, test network-based solutions like PMIPv6 in a cloud-native environment.

Finally, Involving the SDN controller to automate the networking process could be done in further tests.

Acknowledgment

The full credits go to my buddy, my mentor, my supervisor Ákos Leiter, and my wise adviser and my supervisor, professor László Bokor, as I could not do anything without them.

References

[1] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. Number 6275 in Request for Comments. IETF, 2011-07. Published: RFC 6275.

[2] D. Le, X. Fu and D. Hogrefe, "A review of mobility support paradigms for the internet," in IEEE Communications Surveys & Tutorials, vol. 8, no. 1, pp. 38-51, First Quarter 2006, doi: 10.1109/COMST.2006.323441.

[3] Ferretti, Stefano, Vittorio Ghini, and Fabio Panzieri. "A survey on handover management in mobility architectures." Computer Networks 94 (2016): 390-413.

[4] Chan, H. Anthony, et al. "Distributed and dynamic mobility management in mobile internet: current approaches and issues." JCM 6.1 (2011): 4-15.

[5] P. C. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma and A. A. Franklin, "A cloud native solution for dynamic auto scaling of MME in LTE," 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, 2017, pp. 1-7, doi: 10.1109/PIMRC.2017.8292270.

[6] Giaretta, Gerardo. "Interactions between proxy mobile IPv6 (PMIPv6) and mobile Ipv6 (MIPv6): Scenarios and related issues." IETF RFC6612 (2012).

[7] Cziva, Richard, Simon Jouet, and Dimitrios P. Pezaros. "Gnfc: Towards network function cloudification." 2015 IEEE conference on network function virtualization and software defined network (NFV-SDN). IEEE, 2015.

[8] Kempf, J. "RFC 4831: goals for network-based localized mobility management (NetLMM)." proxy MIP (2007).

[9] Cloud Native Computing Foundation (CNCF), Official Website, https://www.cncf.io/, last accessed: 2020.09.29.

[10] Docker, Official Website, https://www.docker.com/, last accessed: 2020.09.29.

[11] Kubernetes, Official Website, https://kubernetes.io/, last accessed: 2020.09.29.

[12] Dragoni, Nicola, et al. "Microservices: yesterday, today, and tomorrow." Present and ulterior software engineering. Springer, Cham, 2017. 195-216.

[13] Astyrakakis, Nikolaos, et al. "Cloud-Native Application Validation & Stress Testing through a Framework for Auto-Cluster Deployment." 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). IEEE, 2019.\

[14] suse.com, "Containers, Kubernetes and the Road to Cloud Native Application Delivery", A pragmatic guide for business and IT leaders, White paper, "https://www.suse.com/media/white-paper/containers_kubernetes_and_the_road_to_cloud_native_application_delivery_wp.pdf".

[15] redhat.com, "The path to cloud-native applications", 8 steps to guide your journey, E-Book, "https://www.redhat.com/cms/managed-files/mi-path-to-cloud-native-apps-ebook-f12255cs-201805-en_0.pdf ".

[16] ETSI, NFVISG. "Network functions virtualisation (NFV); Network Operator Perspectives on NFV priorities for 5G." (2017).

[17] cncf.io," CNCF SURVEY 2019", Deployments are getting larger as cloud native adoption becomes mainstream," https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf ".

[18] NFV, GS. "Network Functions Virtualisation (NFV); Architectural Framework." NFV ISG (2013).

[19] Chiosi, Margaret, et al. "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action." SDN and OpenFlow world congress. Vol. 48. sn, 2012.

[20] ETSI, GSNFV. "Network functions virtualisation (nfv); use cases." V1 1 (2013): 2013-10.

[21] ONF," Software-Defined Networking:The New Norm for Networks", ONF White Paper, April 13, 2012.

[22] Nunes, Bruno Astuto A., et al. "A survey of software-defined networking: Past, present, and future of programmable networks." IEEE Communications surveys & tutorials 16.3 (2014): 1617-1634.

[23] Braun, Wolfgang, and Michael Menth. "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices." Future Internet 6.2 (2014): 302-336.

[24] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).

[25] Gundavelli, S., Leung, K., Devarapalli, V., Chowdhury, K., & Patil, B. (2008). Rfc 5213: Proxy mobile ipv6. IETF, Aug.

[26] Kong, Ki-Sik, et al. "Mobility management for all-IP mobile networks: mobile IPv6 vs. proxy mobile IPv6." IEEE Wireless communications 15.2 (2008): 36-45.

[27] Tantayakul, Kuljaree, Riadh Dhaou, and Beatrice Paillassa. “Impact of SDN on mobility management.” 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2016.
[28] Kang, Ju-Eun, et al. “Seamless handover scheme for proxy mobile IPv6.” 2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications. IEEE, 2008.
[29] Eddy, Wesley M. “At what layer does mobility belong?.” IEEE Communications Magazine 42.10 (2004): 155-159.
[30] Reinbold, Pierre, and Olivier Bonaventure. “IP micro-mobility protocols.” IEEE Communications Surveys & Tutorials 5.1 (2003): 40-57.
[31] Perkins, C. “IP Mobility Support for IPv4, RFC 3344.” IETF, Aug (2002).
[32] Akyildiz, Ian F., Jiang Xie, and Shantidev Mohanty. “A survey of mobility management in next-generation all-IP-based wireless systems.” IEEE Wireless communications 11.4 (2004): 16-28.
[33] Platform9.com,Why Kubernetes matters, white paper, https://platform9.com/wp-content/uploads/2018/08/why-kubernetes-matters.pdf.
[34] Jaramillo, David, Duy V. Nguyen, and Robert Smart. “Leveraging microservices architecture by using Docker technology.” SoutheastCon 2016. IEEE, 2016.
[35] Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs Containerization to Support PaaS. 2014 IEEE International Conference on Cloud Engineering. doi:10.1109/ic2e.2014.41
[36] OpenStack, Official Website, https://www.openstack.org/, last accessed: 2020.10.16.
[37] Google.com, https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption, captured in17/10/2020.
[38] Payaswini, P., and D. H. Manjaiah. “Challenges and issues in 4G Networks Mobility Management.” arXiv preprint arXiv:1402.3985 (2014).
[39] Tuncer, Hasan, Sumita Mishra, and Nirmala Shenoy. “A survey of identity and handoff management approaches for the future Internet.” Computer Communications 36.1 (2012): 63-79.
[40] OpenStack, Official Website, OpenStack: An Overview, https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/marketing/presentations/OpenStack-General-Datasheet-for-US-A4-size.pdf.
[41] S. Kim, H. Choi, P. Park, S. Min, and Y. Han, “OpenFlow-based Proxy mobile IPv6 over software defined network (SDN),” in 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), 2014, pp. 119–125.
[42] Syed M. Raza, Dongsoo S. Kim, DongRyeol Shin, and Hyunseung Choo: Leveraging Proxy Mobile IPv6 with SDN, ACM IMCOM 2014
[43] UMIP website , http://umip.linux-ipv6.org/, last accessed: 2020.10.21.
[44] Liu, D., et al. “RFC 7429: Distributed mobility management: Current practices and gap analysis.” (2015).
[45] Korhonen, Jouni. “Mobile IPv6 in Linux Kernel and User Space.” Proceedings of Seminar on Network Protocols in Operating Systems.
[46] Kempf, J. “Problem statement for network-based localized mobility management (NETLMM) IETF rfc-4830.” (2007).
[47] Network service mesh, Official Website, https://networkservicemesh.io/, last accessed: 2020.10.26.
[48] Flannel documentation on github ,https://github.com/coreos/flannel/, last accessed: 2020.10.26.
[49] DANM documentation on github, https://github.com/nokia/danm, last accessed: 2020.10.26.
[50] Multus documentation on github, https://github.com/intel/multus-cni, last accessed: 2020.10.26.