Softwarization of Large-Scale IoT-based Disasters Management Systems

Carla Mouradian

A Thesis

In

The Concordia Institute

For

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montreal, Quebec, Canada

October 2018

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Carla Mouradian

Entitled: Softwarization of Large-scale IoT-based Disasters Management Systems

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Information and Systems Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Mojtaba Kahrizi — Chair

Dr. Al Leon Garcia — External Examiner

Dr. Yann-Gaël Guéhéneuc — External to Program

Dr. Chadi Assi — Examiner

Dr. Rachida Dssouli — Examiner

Dr. Roch Glitho — Thesis Supervisor (s)

Approved by

Dr. Chadi Assi — Chair of Department or Graduate Program Director

Nov. 30, 2018
Date of Defence

Dr. Amir Asif — Dean, Gina Cody School of Engineering and Computer Science

ABSTRACT

Softwarization of Large-scale IoT-based Disasters Management Systems

Carla Mouradian, Ph.D.
Concordia University, 2018

The Internet of Things (IoT) enables objects to interact and cooperate with each other for reaching common objectives. It is very useful in large-scale disaster management systems where humans are likely to fail when they attempt to perform search and rescue operations in high-risk sites. IoT can indeed play a critical role in all phases of large-scale disasters (i.e. preparedness, relief, and recovery). Network softwarization aims at designing, architecting, deploying, and managing network components primarily based on software programmability properties. It relies on key technologies, such as cloud computing, Network Functions Virtualization (NFV), and Software Defined Networking (SDN). The key benefits are agility and cost efficiency. This thesis proposes softwarization approaches to tackle the key challenges related to large-scale IoT based disaster management systems.

A first challenge faced by large-scale IoT disaster management systems is the dynamic formation of an optimal coalition of IoT devices for the tasks at hand. Meeting this challenge is critical for cost efficiency. A second challenge is an interoperability. IoT environments remain highly heterogeneous. However, the IoT devices need to interact. Yet another challenge is Quality of Service (QoS). Disaster management applications are known to be very QoS sensitive, especially when it comes to delay.

To tackle the first challenge, we propose a cloud-based architecture that enables the formation of efficient coalitions of IoT devices for search and rescue tasks. The proposed architecture enables the publication and discovery of IoT devices belonging to different cloud providers. It also comes with a coalition formation algorithm. For the second challenge, we propose an NFV and SDN based - architecture for on-the-fly IoT gateway provisioning. The gateway functions are provisioned as Virtual Network Functions (VNFs) that are chained on-the-fly in the IoT domain

using SDN. When it comes to the third challenge, we rely on fog computing to meet the QoS and propose algorithms that provision IoT applications components in hybrid NFV based - cloud/fogs. Both stationary and mobile fog nodes are considered. In the case of mobile fog nodes, a Tabu Search-based heuristic is proposed. It finds a near-optimal solution and we numerically show that it is faster than the Integer Linear Programming (ILP) solution by several orders of magnitude.

Acknowledgements

A dissertation is not the outcome of the efforts of entirely one individual. Many people have contributed to its development. At this time, I take the opportunity to acknowledge those who have made some impact on my doctoral journey and accomplishment.

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Roch Glitho for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His constructive feedback and guidance helped me in all the time of research and writing of this thesis. Beside my supervisor, I would like to thank the rest of my thesis committee: Prof. Chadi Assi, Prof. Rachida Dessouli, Prof. Yann-Gaël Guéhéneuc, and Prof. Alberto Leon-Garcia, for their insightful comments and encouragements, but also for the questions which incented me to widen my research from various perspectives.

I thank my colleagues for their help, cooperation, and encouragement. Special thanks to Mohammad Abu Lebdeh and Abbas Soltanian for the stimulating discussions, for the days we were working together before deadlines, and for all the fun we have had in the last few years.

My very special thanks to Elaheh Narjes Tahghigh, my best friend. You were always there with me through the toughest moments. Your friendly advice, your soothing words, and your big heart helped me face all the obstacles and continue with my work. I am truly lucky to count you amongst my friends.

I owe a special thanks to my brother and sister, my father Hrayr, and my mother Ani who helped me throughout my life during this study and have provided me through moral and emotional support. Words cannot express how grateful I am to my mother for all of the sacrifices that she has made on my behalf.

At last, I do not know how to begin with saying thank you to my dearest husband and my best friend, George Zabounian. I love you for being so understanding and for putting up with me through the toughest moments of my life. Thanks for being extremely supportive throughout this entire process and for the countless sacrifices you have made to help me get to this point. I dedicate this Ph.D. thesis to my two lovely children, Lila and Sevag who are the pride and joy of my life. I love you more than anything and I appreciate all your patience and support during mommy's Ph.D. studies. Thanks for always cheering me up.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
BS	Base Station
CPU	Central Processing Unit
CRASAR	Center for Robot Assisted Search and Rescue
CSP	Constraints Satisfaction Problem
EMS	Element Management System
ETSI	European Telecommunications Standards Institute
FIFO	First in First out
GA	Genetic Algorithm
GAE	Google App Engine
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Markup Language
IaaS	Infrastructure as a Service
ILP	Integer Programming Language
IoT	Internet of Things
LAN	Local Area Network
MANET	Mobile Ad-hoc Network
MANO	Management and Orchestration
MEC	Mobile Edge Computing
MR-MT	Multi-Robot Multi-Task
NAT	Network Address Translation
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator
NIST	National Institute of Standards and Technology
NP	Nondeterministic Polynomial time
NSGA	Non-Dominated Sorting Genetic Algorithm

O&M	Orchestration and Management
OCFA	Optimal Coalition Formation Algorithm
OSS/BSS	Operational Support System/Business Support System
PaaS	Platform as a Service
PDF	Probability Distribution Function
PSO	Particle Swarm Optimization
PTCFA	Polynomial Time Coalition Formation Algorithm
QMOPSO	Quantum Multi-Objective Particle Swarm Optimization
QoS	Quality of Service
REST	REpresentational State Transfer
RFID	Radio-Frequency IDentification
RWP	Random Waypoint
SaaS	Software as a Service
SBC	Session Border Controller
SDN	Software Defined Networking
SenML	Sensor Markup Language
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SPEA	Strength Pareto Evolutionary Algorithm
UAV	Unmanned Aerial Vehicles
URI	Uniform Resource Identifier
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtualized Network Function
VNF-FG	VNF Forwarding Graph
VNFM	Virtualized Network Function Manager
WAN	Wide Area Network
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network

Chapter 1

1. Introduction

1.1. Overview

Internet of Things (IoT) is considered as part of the Internet of the future. IoT enables physical objects to interact with each other to share information and to coordinate decisions. Radio-Frequency IDentification (RFID) tags, sensors, and robots are examples of IoT devices. IoT can play a remarkable role and improve the quality of our lives in various domains. Examples of IoT applications include transportation, healthcare, and large-scale natural disasters where human decision making is difficult [1].

Many large-scale disaster management applications rely on IoT, for example in fire detection and fighting, and earthquake early warning and recovery. IoT cannot stop disasters from happening but it can be very useful for disaster preparedness (e.g., prediction) and disaster recovery (e.g., search and rescue tasks). For instance, sensors can withstand a harsh environment and contribute in disaster predictions. They may be distributed throughout a forest to monitor the environmental conditions or measure earth movements before and during earthquakes. On the other hand, robots can operate in dangerous environments and handle search and rescue tasks. In such tasks, the primary goal is to find victims as quickly as possible and to rescue them with utmost care.

Conventional methods employ human rescuers and dogs. However, the rescue teams often cannot reach the disaster sites in time due to collapsed building and destroyed roads. Robots can move quickly and find victims more accurately than their human counterparts. They can, for instance, penetrate rubbles to find people beneath them. The use of robots for search and rescue mission was first noticed during the rescue operations at World Trade Center in New York City on September 2001, where CRASAR (Center for Robot-Assisted Search And Rescue) rescue robots were used [2]. These make IoT a potential tool for large-scale disaster management applications. Hence, a number of different sensor and robot platforms have been designed for such applications.

Network softwarization emerged as a concept that drastically changes the way the network services are designed and operated, enabling network operators to deliver network services and applications with greater agility, flexibility, and cost efficiency. It relies on key technologies, such as cloud computing, Network Functions Virtualization (NFV), and Software Defined Networking (SDN). Cloud computing is an emerging paradigm with inherent benefits such as cost efficiency, rapid elasticity, and resource pooling. NFV is an emerging paradigm to decouple the network functions from the underlying hardware. SDN enables the dynamic orchestration and chaining of VNFs to provide a flexible management of the forwarding behavior of the VNFs.

1.2. Challenges and Thesis Contributions

Network softwarization can facilitate the provisioning of large-scale IoT-based disaster management systems by tackling several challenges. Some examples are as below:

- **Dynamic formation of an optimal coalition of IoT devices:** search and rescue tasks of disaster management applications typically involve IoT devices (i.e., robots) in the order of thousands to accomplish a task [3]. These robots have different capabilities and characteristics, and a certain task may need a coalition of robots to perform it. Forming dynamically an optimal coalition of robots with the required number and the required set of capabilities for search and rescue tasks is very challenging. In addition, finding the appropriate robots in a single business entity is not always possible. A single business entity may not provide all the capabilities and the number of robots required to perform a search and rescue task, and there may be need to use robots belonging to several business entities. Moreover, some tasks may require that the combination of a

given sensor and actuator should reside on the same robots or on different robots. Furthermore, describing these robots independently from their brands, technical constraints, and infrastructure provider is not straightforward. Meeting these challenges is critical for the cost efficiency of IoT-based disaster management applications.

- **Heterogeneity of IoT devices:** different types of IoT devices are used in such large-scale disasters. These IoT devices are usually heterogeneous, each with its own communication protocol and/or data formats. To enable interoperability across IoT devices and applications, gateways are needed to bridge the traditional communication networks and IoT devices domain. Such gateways are generally centralized and thus not practically feasible in the Mobile Ad-hoc Networks (MANET) setting of large-scale disasters where there is no centralized or fixed infrastructure. In addition, their capabilities do not scale when the number of applications and the corresponding workload of IoT devices changes dynamically. Moreover, they lack dynamicity and flexibility. For instance, when a new brand of IoT devices is added to the infrastructure the gateway needs to be upgraded on-the-fly such that it can serve the newly added IoT devices. In addition, when several applications use IoT devices with the same protocols and/or information models, the same gateway could be reused by these applications. Upgrading and reusing existing gateways is very difficult and expensive. Therefore, the IoT gateways architectures need to be rethought.
- **QoS of disaster management applications:** disaster management applications are known to be very QoS sensitive, especially when it comes to delay. Many service providers use cloud computing to deploy their applications. However, the fundamental limitation is the connectivity between the cloud and the IoT devices. Such connectivity is set over the Internet and is not suitable for latency-sensitive applications such as disaster management applications. Furthermore, cloud-based applications are often distributed and made up of multiple components. Consequently, it is not uncommon to sometimes deploy application components separately over multiple clouds (e.g., [4] and [5]). This worsens the latency due to the overhead induced by inter-cloud communications. The location of application components has a significant impact on the overall application execution cost and makespan. Thus, there is a need for efficient algorithms for application component placement.

Unfortunately, the solutions proposed so far do not address all these challenges. This Ph.D. thesis proposes softwarization approaches to tackle the architectural and the algorithmic challenges related to large-scale IoT-based disaster management systems. It makes three main contributions which are presented as follows. Each of our contributions corresponds to a challenge addressed by this thesis.

1.2.1. Cloud-based System for Disaster Management Applications [6], [7]

In the first contribution, we tackle the architectural and the algorithmic challenges for cost-efficient IoT-based disaster management applications provisioning. At the architectural level, we propose a cloud-based solution that allows selecting the most efficient group of robots for the search and rescue tasks of disaster management applications. In addition, the architecture allows publishing and discovering robots belonging to different infrastructures. A well-defined language to describe robot capabilities based on existing standards is also considered. This is important when considering robots with different platforms (e.g., different capabilities, sizes, and shapes). In addition, a proof of concept prototype to validate the feasibility of the approach is also developed. The new architecture enables flexible, elastic, and cost-efficient use of robots, benefiting the cloud advantages such as virtualization and scalability.

At the algorithmic level, the goal is to ensure that the optimal coalition of robots is selected dynamically with the required capabilities for resource efficiency. In addition, location constraints regarding the capability distribution of the robots are taken into consideration. This is necessary in order to ensure proper execution of the sub-tasks belonging to the search and rescue task. Extensive simulation experiments are also conducted, and the proposed algorithm is compared with other existing algorithms. The simulation results demonstrate that the proposed algorithm cannot only improve the solution but can also significantly reduce the processing times.

1.2.2. An Architecture for IoT Gateway based on NFV and SDN [8], [9], [10]

The second contribution is an architecture for IoT gateway based on NFV and SDN. Both centralized and distributed approaches are considered. For the centralized approach, the elastic scalability of the architecture is considered, which is crucial to adapt to the accelerated growth of the number of applications using the IoT devices. The architecture relies on a simple dynamic resource allocation algorithms to meet the growing demand of applications. Existing algorithms such as [11] and [12] are used as a basis. For the distributed approach, the proposed architecture

considers co-locating the gateway functions with the IoT devices and reusing already deployed gateways. It also considers handling the traffic and chaining between the gateway functions dynamically. For both approaches, a high-level description of the proposed architecture that is composed of two planes is provided, and a detailed description of each plane with their corresponding interfaces and procedures is presented. The proposed architectures are implemented as a proof of concepts in order to evaluate their viability and performance level. The performances results show advantages of using on-the-fly provisioning of IoT gateways and the possibility of reusing and updating a pre-existing gateway.

1.2.3. Application Component Placement Algorithm over Hybrid Cloud/Fog NFVI [13], [14], [15]

The third contribution is an application component placement algorithm over hybrid NFVI-based cloud/fogs. Our critical review of the existing cloud/fog systems [13] shows the need for a component placement algorithm for IoT applications over hybrid cloud/fog infrastructures. Fog computing helps in meeting the QoS. We consider both stationary and mobile fog nodes. The applications are considered as sets of interacting components that can be executed in sequence, in parallel, or by using more complex constructs such as selection and loops. The mobility of fog nodes is modeled using the Random Waypoint (RWP) model [16]. Based on the stationary distribution fog nodes' location, the expected makespan and cost for the constructs of sequence, parallel, selection, and loop are calculated. Next, the constructs' calculations are aggregated to obtain the application's makespan and execution cost. The problem is formulated as an Integer Linear Programming (ILP) problem and, regarding the complexity, Tabu Search-based heuristic is proposed to find the sub-optimal solution. The performance results show that the proposed algorithm reaches the optimality in several scenarios and reduce the execution time significantly compared to the ILP by many orders of magnitude.

1.3. Background Information

This subsection presents the background information that is relevant to our research domain. The background information covers four topics: Internet of Things, disaster management systems, network softwarization, and fog computing.

1.3.1. Internet of Things

The Internet of Thing (IoT) is considered as part of the Internet of the future. It is a novel paradigm that is gaining the attention of modern wireless telecommunications. It is present around us of a variety of things or objects such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. which are able to interact with each other and cooperate with their neighbors to achieve common goals [17].

IoT concept can be realized by several enabling technologies. One example is identification, sensing, and communication technologies such as RFID tags that are characterized by a unique identifier and sensor networks composed of several nodes communicating in a wireless fashion, etc. Another example is the middleware which is a software layer positioned between the technological and the application levels. IoT middleware's architectures proposed in the last years often follow the Service Oriented Architecture (SOA) approach. It simplifies the development of new services and the integration of legacy technologies with the new ones. It has five layers:

- Applications layer on top of the architecture, it exploits the functionalities of the other layers and provides it to the end-user.
- Service composition layer which provides the functionalities of the composition of different services by the objects to build specific applications.
- Service management layer provides functionalities such as object discovery, status monitoring, and service configuration. It enables the remote deployment of new services during run-time to meet the application requirements.
- Object abstraction which provides an abstraction of the heterogeneous objects by harmonizing the access to the different objects. This is done by offering common languages and procedures.
- Trust, privacy, and security management layer which provides functionalities related to the security and the privacy of the exchanged data.

IoT can be present in a variety of fields such as domotics, assisted living, e-health, enhanced learning, automation and industrial manufacturing, logistics, business management, and large-scale disaster management - to name few. Moreover, many standardization efforts are being carried for the IoT paradigm. For instance, IETF introduced the IPV6 over Low-Power Wireless

Personal Area Networks (6LoW-PAN) which defines a set of protocols that can be used to integrate sensors nodes into IPV6 networks.

1.3.2. Disaster Management System

Natural disasters such as earthquakes, wildfires, flooding, etc., happen daily worldwide and represent an important factor that affects human life and development. Such disasters, manmade and natural, are a cause of great economic and human losses each year throughout the world [18]. One example is the large-scale earthquake that hit Kobe, Japan, on January 17, 1995. Measuring 6.9 magnitudes, it was the deadliest to hit Japan in 47 years, with more than 5000 dead and more than 13,000 injured. It smashed more than 103,521 buildings, leaving a large number of bodies under debris. Damage from Kobe earthquake cost around $100 billion. An important issue that needs to be solved when a disaster occurs, is to preserve human lives. In this context, the first 72 hours after the disaster hit are the most critical [19]. This means that the search and rescue operations must be completed quickly and efficiently. The IoT can be very useful for disaster preparedness (e.g., prediction) and disaster recovery, in particular, it can participate in the following three phases of disaster management [19]:

- Pre-disaster preparedness: this phase is concerned with surveying-related events that precede the disaster. For instance, processing data of environmental conditions collected from different sensors, e.g., acceleration of physical objects, seismic waves, and threshold sensing, and setting up early warning systems.
- Disaster assessment: this phase provides situational awareness during the disaster in real-time and completes damage studies for logistical planning. For instance, damage-assessment maps are produced in this phase that allow responders to serve areas that experienced more damage first. In addition, the affected areas are monitored through drones to detect locations of possible human being presence under the ruins
- Disaster response and recovery: in this phase, the communications backbone is formed, and search and rescue strategies are implemented. These strategies are implemented according to the area size and conditions. Accordingly, whether to send first responders, rescue robots, or both to assist in recovery procedures is decided.

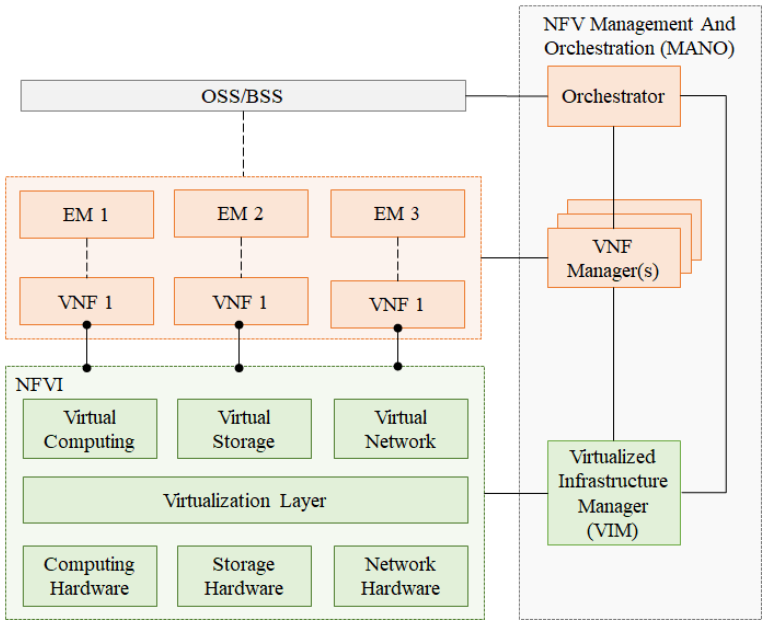


Figure 1.1 A high-level architecture of NFV

1.3.3. Network Softwarization

Traditionally, network operators deploy physical proprietary devices and equipment for each function that is part of a given service [20] hence they lack dynamicity and flexibility. In addition, it is difficult and expensive to upgrade or reuse existing functions or services. Network softwarization, in contrast, enables flexibility, adaptability, and reconfiguration of a network on the fly based on timely requirements [21]. It is the concept of designing, architecting, deploying, and managing network components, primarily based on software programmability properties. Network softwarization has shown huge potential in revolutionizing the way the network services are designed and operated enabling network services and applications with greater agility and cost efficiency. The network softwarization term was first introduced at the academic conference NetSoft 2015, the first IEEE conference on Network Softwarization. The main goal was to include a broader interest in NFV, SDN, and cloud computing.

NFV [20], by leveraging virtualization technology, offers a new way to design, deploy, and manage network service. It decouples the network functions from underlying hardware to run them as software instances (i.e., Virtual Network Functions [VNFs]) on general purpose hardware. The decoupling reduces operational expenditures by leveraging efficiencies that derive from virtualization in cloud computing such as elastic scalability, flexibility, and customization. The

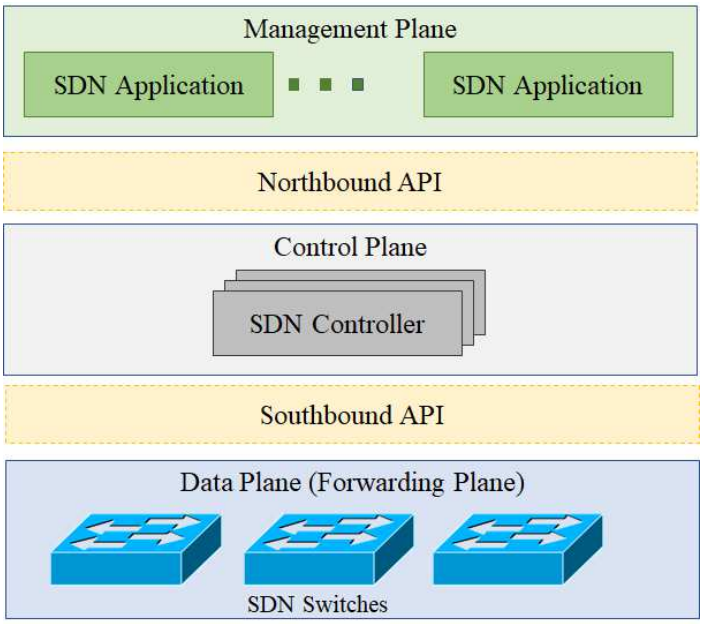


Figure 1.2 A high-level architecture of SDN

European Telecommunications Standards Institute (ETSI) has defined a reference architectural framework for NFV [22]. It is made up of a set of three main components as shown in Figure 1.1: VNFs, NFV Infrastructure (NFVI), and an NFV Management and Orchestration (MANO) framework. VNFs are the software implementation of given network functions. The NFVI provides hardware and software resources, including the computation, storage, and networking needed to deploy, manage, and execute VNFs. The NFV MANO framework enables the automated management of the VNFs by managing the NFVI and orchestrating the allocation of resources needed by the VNFs. It consists of three functional blocks: NFV Orchestrator (NFVO), VNF Manager (VNFM) and Virtualized Infrastructure Manager (VIM). The NFVO is responsible for the orchestration of the NFVI resources and the lifecycle management of the network services. The VNFM manages the lifecycle of the VNFs. The VIM is responsible for managing and controlling the NFVI. In NFV, an end-to-end network service is realized by an ordered set of VNFs that are deployed in the network and chained. This chain is called VNF Forwarding Graph (VNF-FG).

SDN [23] aims at splitting the control plane and the data plane in the network elements to provide a flexible management of the forwarding behavior of those elements. It can enable the easy on-the-fly chaining of these network functions. It also enables faster innovation, leading to

greater responsiveness and cost-effectiveness. The architecture of SDN contains three planes as shown in Figure 1.2: a management plane, a control plane, and a forwarding plane. The SDN application resides at the management plane. Its role is to communicate its requirements (e.g., the desired network behavior) to the control plane. This is done by defining a set of application policies and injecting them into the control plane. The control plane contains the SDN controller, whose responsibility is to translate the requirements of the SDN application to the forwarding plane. To that end, the SDN controller programs the forwarding plane by populating the SDN switches with well-defined flow entries (forwarding rules). The forwarding plane consists of forwarding elements such as switches and routers that allow traffic forwarding based on the flow entries that reflect the application's policies. SDN is highly complementary to NFV. Both are closely related technologies and are mutually beneficial (but not dependent). By using SDN, the network routers and switches can be dynamically programmed to steer the traffic through a set of VNFs.

Cloud computing has emerged as a viable delivery model for IT resources. It leverages virtualization technology to enable on-demand network access to a shared pool of configurable resources (e.g., networks, servers, storage, applications, and services). It has brought new business models and enormous benefits to enterprises. It comes with several inherent capabilities such as scalability, on-demand resource allocation, reduced management efforts, flexible pricing model, and easy applications and services provisioning [24]. Virtualization is a key enabling technology for cloud computing, allowing the abstraction of actual physical computing resources into logical units and enabling their efficient usage by multiple independent users. Virtualization can be performed at both node- and network- levels. Robot node-level virtualization is defined as the mechanisms that enable multiple applications to reside in and run concurrently on a single robot [25]. And, robot network-level virtualization is the dynamic formation of subsets of robot nodes, with each subset dedicated to a certain application at a given time [25].

Currently, the network operators are transforming their infrastructure to NFV and SDN enabled cloud infrastructures. NFV, SDN, and cloud together are driving the softwarization of networks toward a paradigm where software controls the treatment of flows in the network and deliver customized characteristics that meet the needs of each application. This allows reinventing future network architectures and facilitates infrastructure management.

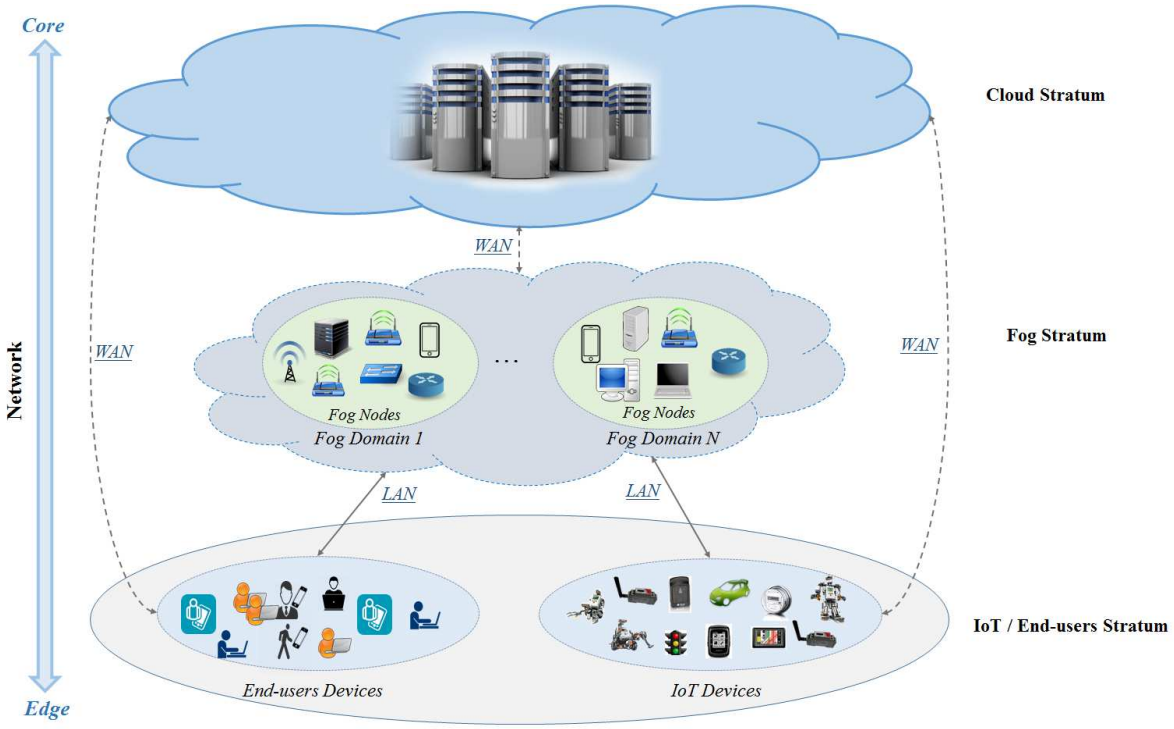


Figure 1.3 The fog system

1.3.4. Fog Computing

Fog is an extension of cloud computing paradigm from the core to the edge of the network. It is not a completely new concept. Cyber foraging, cloudlet, and Mobile Edge Computing (MEC) were introduced before the fog to bring computing closer to end devices [13].

Fog enables computing at the edge of the network, closer to IoT devices. It also supports virtualization. Fog is tightly linked to the existence of a cloud, i.e., it cannot operate in a standalone mode. This has driven a particular attention on the interactions between the cloud and the fog [26]. Moreover, fog extends the three-tier hierarchy of cloudlet to an n-tier one, offering more flexibility to the system [27][28]. Fog also provides life-cycle management of applications distributed over the n-tier hierarchy, through the "Fog Service Orchestration Layer" [27].

Figure 1.3 shows a fog system with a three-tier architecture. It has three strata: The cloud stratum, the fog stratum, and the IoT/end-users stratum. The fog stratum can be formed by one or more fog domains, controlled by the same or different providers. Each of these fog domains is formed by the fog nodes that can include edge routers, switches, gateways, access points, PCs, smartphones, set-top boxes, etc. The communication between the IoT devices and the fog nodes is

done through Local Area Network (LAN). Instead, the communication between the IoT devices and the cloud nodes requires connection over the Wide Area Network (WAN), through the fog or not.

1.4. Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 discusses the motivating scenario, requirements, and provides a critical review of the state-of-the-art. We organize the thesis into architectural contributions and algorithmic contributions. Accordingly, for the architectural contribution, Chapter 3 presents a cloud-based architecture for cost-efficient IoT-based disaster management application provisioning, Chapter 4 presents the proposed NFV-based centralized architecture for IoT gateway, and Chapter 5 presents an NFV and SDN –based distributed architecture for IoT gateway. For the algorithmic contributions, in Chapter 6 we present a coalition formation algorithm for multi-robot task allocation and in Chapter 7 we discuss the application component placement problem over hybrid cloud/fog NFVIs and present the proposed ILP model and the heuristic for that. Finally, we conclude this manuscript in Chapter 8 and provide future directions for this research work.

Chapter 2

2. Related Work

In this chapter, we first present a motivating scenario. Then, we present a set of requirements derived from the scenario. Finally, we survey and review the related work in light of these requirements.

2.1. Motivating Scenario

An earthquake early warning and recovery application can illustrate the motivation behind our work. The application is considered that consists of two phases: an early warning phase and a recovery phase. In the first phase, the application monitors the environmental conditions by collecting environmental data such as the acceleration of physical objects, wind speed, etc. In the second phase, the application monitors the affected areas and performs search and rescue tasks.

In the early warning phase, the monitoring of the disaster can be facilitated by the use of IoT devices. Data can be acquired by real-time GPS sensors, accelerometer sensors hosted in homes, business or schools [29]. In addition, the monitoring can be performed by using unmanned aerial vehicles (UAV) such as drones. Drones have several benefits such as small size, exposure to dangerous environments, and low cost of operation. Drones are usually equipped with camera

sensors which can capture images. The collected images are processed later to produce damage-assessment maps to allow responders to service areas that experienced more damage first.

In the recovery phase, after the disaster occurs, emergency rescue teams are created immediately, and troops are sent to assist the search and rescue operations. However, these teams have a hard time reaching the quake site because of the lack of information about the quake sites and blocked roads. To successfully complete this search and rescue operation is even beyond human capacity when the scale of a disaster is too large. IoT devices can be an alternative. Robots, for instance, have a wide range of capabilities. They could make a noticeable contribution in such large-scale scenarios. Some can be equipped with sound sensors to detect voices or other sounds of possible human presence through the ruins. They use specialized arms to sift through debris. Others carry thermal sensors to detect body heat. Some robots with hexapod legs can walk over the debris – where humans cannot – and pass a camera rod into the debris to assist in the search. With these possibilities, they outdo their human counterparts in of crisis situations on site. Moreover, drones can make finding the earthquake survivors faster and easier. They can fly and capture images to identify possible human presence under the ruins and inform rescue agencies to send first responders or rescue robots, or both in some cases.

This earthquake early warning and recovery application can be made up of several components, such as:

- ***Early Warner and Analyzer (EW)*** – process data of environmental conditions such as acceleration of the physical objects, images, seismic waves, etc., and accordingly detect prospective disaster.
- ***Warning Alert Issuer (WA)*** – warns the community with smartphones of imminent hazard (similar to Zizmos [30]), also it can be connected to television networks and broadcasting media to alert the citizens (similar to Grillo [31]).
- ***Map Producer (MP)*** – processes the data relevant to the earthquake to determine the earthquake epicenter location and direction of seismic waves and provides damage-assessment maps allowing responders to service areas that experienced more damage first.
- ***Victim Detector (VD)*** – monitors the affected areas and detect locations of possible human being presence under the ruins.

- ***Rescue Strategies (RS)*** – implements rescuing strategies according to the area size and conditions and decides whether to send first responders (***First Responders (FR)***), rescue robots (***Robots Dispatcher (RD))***, or both ***(Human-Robot Team (HR))*** to assist in recovery procedures.
- ***Historical Storage (HS)*** – collects historical information gathered about a particular seismic event.

Some of these components can be very latency sensitive such as Early Warner & Analyzer and Victim Detector, while others can be computationally intensive and delay tolerant such as Historical Storage. The following interactions can happen between the different components:

1- The ***Early Warner & Analyzer*** analyses and processes the data received by different sensors such as seismic sensors and camera sensors. For instance, it processes the images and the seismic waves automatically and identifies events of interest, such as potential disasters.
2- The ***Early Warner & Analyzer*** also sends the selected data to ***Historical Storage*** for long-term storage and analysis.
3- When the ***Early Warner & Analyzer*** detects a potential earthquake, it sends the information to the ***Warning Alter Issuer*** to alert the public.
4- It also sends the information (images, seismic data) to the ***Map Producer*** to find the epicenter location and produce assessment maps.
5- The produced maps are sent to ***Warning Alter Issuer*** so the latter can send it to different departments (e.g., fire department, public transportation, etc.).
6- The produced maps are also sent to ***Historical Storage*** for further analyses.
7- The ***Map Producer*** sends the information of the earthquake to ***Victim Detector*** to find victims.
8- The ***Victim Detector*** monitors different areas to detect locations of possible human being presence using camera sensors. When it finds victims, it sends the relevant images to the ***Rescue Strategies***.
9- The ***Rescue Strategies*** take an immediate life-saving decision and start the rescue mission. It instructs either ***First Responders (FR)***, ***Robots Dispatcher (RD)***, or ***Human-Robot Team (HR)*** to start the rescue operations.

Different IoT platforms have been designed for such large-scale disaster applications that typically involve IoT devices in the order of thousands to accomplish a mission. These IoT devices

might belong to different business entities. They can be provisioned through several infrastructures owned by different business entities. Each entity hosts IoT devices with different capabilities. In addition, the earthquake early warning and recovery application may have its own constraints in terms of communication protocols, data formats, etc. and the IoT devices may use different communication protocols and data formats. Moreover, after an earthquake, there is an increasing possibility of fire. For instance, the earthquakes in San Francisco in 1906, in Northridge in 1994, in Kesennuma City in Miyagi, Japan in 2011 were followed by devastating fires that lasted for several days. In such cases, a fire detection and fighting application might consider adding new types of sensors with different capabilities, communication protocols, and/or data formats to the infrastructure. These sensors allow the application to collect temperature, humidity, and CO2 data in order to evaluate the contour and the intensity of any fire it detects and to dispatch firefighter robots accordingly.

2.2. Requirements

According to the motivating scenario, the following requirements are derived. We categorize the requirements into architectural requirements and algorithmic requirements.

2.2.1. Architectural Requirements

In this subsection, we divide the requirements into general architectural requirements, requirements specific to the cloud-based architectures for IoT applications provisioning, and requirements specific to the IoT gateway.

A) General Architectural Requirements

The following requirements are identified as general requirements on the architectures.

1) **Elastic Scalability:** the architecture should function well and be scalable in terms of the number of IoT devices and number of applications. Accordingly, the system should scale down and up in an elastic manner.

2) **Extensible Architecture:** the solution should take future growth into consideration; extensibility can be through adding new architectural modules such as fault management which may enhance the performance of the system, or through modification of existing modules. The extensibility can also be through the support of future scenarios and new application domains.
3) **Heterogeneity:** the heterogeneity should be considered in terms of IoT devices. For instance, because of the diversity of IoT devices' vendors, one system may contain IoT devices belonging to different providers with each having its own interface and programming language. This means the overall solution should be applicable to a wide variety of heterogeneous IoT devices.
4) **Interoperability**: the proposed architecture should have an appropriate signaling and control interfaces, as well as appropriate data interfaces to enable interoperability at the level of providers and architectural modules (e.g., between the IaaS and the PaaS, between the IoT gateway and the application).

B) Requirements Specific to Cloud-based Architecture for IoT Applications Provisioning

The following requirements are considered to be important for cloud-based architecture for IoT applications provisioning.

1) **Network-level Virtualization:** the architecture should support network-level virtualization for the robots since we are dealing with dynamic environments. Also, some tasks that cannot be solved individually or can be solved more efficiently as a group may need collaboration between several robots.
2) **Task Delegations:** the architecture should be able to delegate tasks to robots that belong to other infrastructures. This is very important in some cases, such as when the capabilities or the number of the robots belonging to one infrastructure may not be sufficient for a given task which may result in incompletion of a task or completing it in a non-efficient manner.
3) **Publication/Discovery:** the architecture should support the possibility to publish and discover idle robots with their characteristics (static, dynamic, etc.). This should be speedy and ahead of time if possible due to the dynamic nature of tasks.

C) Requirements Specific to IoT Gateway

The following requirements are considered to be important for designing IoT gateways.

1) **Distributed:** due to the ad-hoc nature of disaster scenarios where there is no centralized or fixed infrastructure, the gateway should be deployed in a distributed manner.
2) **On-the-fly Upgradability:** the gateway should be upgraded on-the-fly when new IoT devices are deployed. For instance, in case of fire, new sensors should be added to the infrastructure to send data to the fire detection and fighting application. This leads to the need to upgrade the gateway such that it can serve the newly added IoT devices.
3) **Reusability:** several applications should be able to use the same gateway. This may be required when, for instance, the sensors used by the fire detection and fighting application use the same protocol as the sensors used by the earthquake early warning and recovery application.

   Standard Northbound and Proprietary Southbound Interface: the gateway should support standard northbound and proprietary southbound interfaces. For the standard northbound interface, an example could be the widely used Sensor Markup Language (SenML) [32] carried over HTTP. It is designed to encode sensor measurements and device parameters, and its extension could be used as a standard interface for robots. For instance, in [33], SenML is extended as a unified robots description model while in [34] SenML capabilities is extended to provide a uniform representation of sensors and robots and to control robots. On the other hand, having a proprietary southbound interface is necessary since devices can be vendor lock-in or because standard interfaces do not support the device's functionality.
4) **Provide Key Gateway Functionalities:** the architecture should provide at least some key gateway functions, such as protocol conversion, information model conversion, data aggregation, and metadata adaptation.
5) **Performance:** the architecture must ensure that the execution of gateway modules achieves performance similar to when they are executed in a traditional gateway. In particular, the performance metrics that require significant attention are latency and overhead.

2.2.2. Algorithmic Requirements

For the algorithmic requirements, we consider that a given system will meet a given requirement if the algorithm either has the requirement as the main objective or the requirement is part of the set of constraints the algorithm should satisfy. Or, if the requirement is factored in the

models and operations of the algorithms. We further discuss these aspects as we present each requirement.

A) General Algorithmic Requirements

The following requirements are identified as general requirements on the algorithms.

1) **Heterogeneity:** the heterogeneity should be taken into account in terms of IoT devices, fog nodes, and cloud nodes. Algorithms need to take this heterogeneity into account. The limitations of specific nodes need to be factored in the models and operations of the algorithms.
2) **QoS:** meeting the QoS requirements, such as delay, jitter, and throughput, is critical in disaster management systems. This thesis focuses on the delay requirements of disaster management applications. The algorithms should be able to minimize the time needed to perform the search and rescue tasks of disaster management applications and the total application execution time.
3) **Cost:** the algorithm should be able to minimize the cost which is a budget for resources consumption, including robot deployment cost, application component deployment cost, cloud/fog nodes cost, and bandwidth cost.

B) Requirements Specific to Coalition Formation of Robots

We defined the following requirements for coalition formation algorithms for robots.

1) **Resource Optimization:** the algorithm should be able to minimize the number of robots in a coalition performing a search and rescue task/sub-tasks in order to make robots available for other tasks/sub-tasks of the disaster management application.
2) **Capability Distribution of Robots:** some tasks can be tied by locational constraints regarding the capability distribution of the robots, while others may be executed without any locational constraints. For instance, a combination of sensors and actuators should reside on the same robot, or on different robots. The algorithm should ensure this in order for the proper execution of the search and rescue tasks/sub-tasks.

C) Requirements Specific to Application Component Placement over Hybrid Cloud/Fog

The following requirements are considered to be important for designing efficient application component placement algorithms. They should be factored in the model and operations of the algorithm.

1) **Non-deterministic Applications Graphs**: the algorithm should consider non-deterministic application graphs, with sub-structures as selections and loops, when making the placement decision.
2) **Mobility:** fog nodes can be mobile. Accordingly, the algorithm should be able to handle this mobility. An efficient algorithm should consider such mobility during the placement decision to avoid high cost or a long makespan as a result of assuming stationary nodes with predefined locations.

2.3. Related Work

In this section, first, the architectural related works are reviewed, then the algorithmic related works are reviewed.

2.3.1. Architectural Related Works

This subsection first, presents the related work for cloud-based architectures for IoT applications provisioning, and then, review the architectures for IoT gateways.

A) Cloud-based Architectures for IoT Applications Provisioning

We review the related work in this subsection of two areas. First, we discuss the use of robots in large-scale disasters. Second, we review architectures for robotic applications in the clouds.

1) Robots in Large-Scale Disasters

Kitano *et al.* [3] present a detailed analysis of search and rescue domains in large-scale disasters. They identify several research issues in search and rescue strategy, such as real-time planning and multi-agent planning, which may involve over 10,000 agents including humans and robots. As their main focus, they analyze the large-scale search and rescue domain and introduce the RoboCup-Rescue Simulation project. Among others who approach this topic, Messina *et al.* [35] propose a robot ontology for search and rescue domains. Chatterjee *et al.* [36] identify the needs for and benefits of developing a standard description and ontologies for rescue robot features and disaster scenarios. However, ontologies add overhead in developing and maintaining new components.

2) **Robotic Applications in Cloud**

In [25], the authors propose an architecture for robotic applications as cloud computing services. The proposed architecture provides support for heterogeneous robots and delegates tasks to robots belonging to other IaaSs. However, it does not perform network-level virtualization; neither does it consider large-scale disaster applications. Chen *et al.* in [37] propose an architecture that decouples robots' sensing and actuating capabilities to offer them as SOAP-based services. However, they provide a solution for one robot they designed; yet, supporting heterogeneous robots is not addressed. Du *et al.* [38] have improved the architecture presented in [37] by adding support for network-level virtualization. However, a mechanism to delegate tasks to robots belonging to other clouds is another important characteristic to take into account. Turnbull *et al.* [39] propose a cloud infrastructure for robots. Their robotic cloud receives images from a vision acquisition and performs some computation, and finally implements an algorithm to control the robot behavior. However, they do not consider heterogeneous robots as their solution applies only to one type of robot hardware (i.e., iRobot). In [40], Liu *et al.* present a cloud-enabled robotics system where robots offload their computationally intensive tasks to the cloud. Robot Operating System (ROS), a robotic middleware to develop robot software, is used as the robot platform. However, the authors do not discuss how to discover and publish robots.

In [41], the authors propose a Robot *as-a-Service* platform that provides easy access to heterogeneous robots. The proposed design consists of an OCCI extension that models cloud robotics *as-a-Service*. It also includes a gateway for hosting mobile robot resources. The proposed platform allows users to have a unified view of all robots. However, architectural modules that perform network-level virtualization are not discussed. Mohanarajah *et al.* in [42], the authors propose Rapyuta, an open source PaaS framework for robotic applications. Rapyuta computing environment allows robots to easily access the RobotEarth knowledge repository. The latter enables robots to benefit from the experience of other robots. Rapyuta allows robots to offload heavy computation to the cloud. It dynamically provides secure computing environments for the robots. These computing environments are tightly interconnected, allowing robots to share their services and information with other robots. However, creating teams of robots or coalitions is not discussed.

B) Architectures for IoT Gateway

In this subsection, we review the state-of-the-art for both traditional architectures for IoT gateways and NFV/SDN-based architectures.

1) **Traditional Architectures for IoT Gateways**

Several works have proposed IoT gateway architecture. Some designed their architecture without considering the use of NFV/SDN technology. For instance, Datta *et al.* [43] propose a smart M2M gateway architecture to manage the huge volume of M2M devices and endpoints. They extended the capabilities of CoRE Link to add additional resource types for SenML units. In [44], an architecture for an in-home IoT gateway is proposed. It consists of three subsystems: sensor node, gateway, and application platform. The architecture does not support standard or proprietary interfaces. A configurable, multifunctional and cost-effective architecture for smart IoT gateways is proposed in [45]. It is extensible since modules with different communication protocols can be plugged into the architecture. It also provides protocol conversion by granting a common frame structure for data communication. However, scalability in terms of the number of applications is not discussed. In addition, these gateways cannot be upgraded on-the-fly when introducing new types of IoT devices or new applications.

2) **NFV and/or SDN-based IoT Gateway Architectures**

Other works have investigated using NFV and/or SDN technology when designing IoT gateways. For instance, Li *et al.* [46] propose an IoT architecture based on SDN. Their proposed gateway allows introducing new applications through open programmable interfaces. It supports standard data formats using JSON and provides protocol conversion functionality as one of the gateway's key functionalities. However, the proposed gateway cannot be deployed over a MANET, as it does not have a distributed nature. In addition, it does not enable the same gateway to be used by more than one application. Ojo *et al.* [47] propose an SDN-IoT architecture coupled with NFV. Their goal is to address the scalability and the mobility issues in IoT networks. They replace traditional gateways with SDN gateways and implement the functionalities of the gateways as VNFs. The VNFs are SDN-enabled. This work supports heterogeneous IoT devices and provides key gateway functions. Also, the programmability feature of SDN allows the gateway to be updated dynamically. However, a drawback of is that the proposed architecture cannot be deployed over ad-hoc networks since it does not have a distributed architecture.

Table 2.1 Architectural related work evaluation

Related Works \ Requirements			Architectural Requirements													
			General Architectural Requirements				Requirements for Cloud-based Architectures			Requirements for IoT Gateway						
			Elastic Scalability	Extensibility	Heterogeneity	Interoperability	Network-level Virtualization	Task Delegation	Publication/Discovery	Distributed	Upgradability	Reusability	Standard NB and Proprietary SB Interfaces	Key Gateway Functionalities	Performance
Cloud-based Architecture for IoT Applications		**Mouradian *et al.*** [25]	x	✓	✓	✓	x	✓	x						
		Chen *et al.* [37]	x	✓	x	✓	x	x	✓						
		Du *et al.* [38]	x	✓	✓	✓	✓	x	✓						
		Turnbull *et al.* [39]	x	x	x	x	x	x	x						
		Liu *et al.* [40]	x	✓	x	✓	x	x	x						
		Merle *et al.* [41]	x	✓	✓	✓	x	x	x						
		Mohanarajah *et al.* [42]	x	✓	✓	✓	x	x	✓						
Architectures for IoT Gateway	Traditional IoT Gateways	**Datta *et al.*** [43]	x	✓	✓	✓				x	x	x	✓	✓	N/A
		Zhu *et al.* [44]	x	✓	✓	x				x	x	x	x	✓	N/A
		Guoqiang *et al.* [45]	x	✓	✓	x				x	x	x	x	✓	N/A
	NFV and/or SDN-based IoT Gateways	**Li *et al.*** [46]	x	✓	✓	✓				x	✓	x	✓	✓	x
		Ojo *et al.* [47]	✓	✓	✓	x				x	✓	x	✓	✓	x
		Salman *et al.* [48]	x	✓	✓	x				x	✓	x	✓	✓	x

Salman *et al.* [48] propose a global IoT architecture leveraged with SDN. The proposed architecture inherits management and programmability capabilities from SDN and mobility capabilities from the fog. The gateways in the proposed architecture are SDF gateways that ensure interoperability between different communication protocols and heterogeneous networks, thereby providing key gateway functions. Their architecture also supports a standard northbound interface i.e., REST, and its programmability feature allows dynamically updating the gateway. However, the SDF-gateway does not have a distributed architecture, and so it cannot be deployed over an IoT MANET.

2.3.2. Algorithmic Related Works

This subsection first, presents the related work for coalition formation algorithms for robots and then, review the algorithms proposed for application components placement over hybrid cloud/fog NFVIs.

A) Coalition Formation Algorithms for Robots

Liu and Chen [49] propose an algorithm based on Genetic Algorithm (GA) to form the best coalition of robots. Authors in [50] propose a modified version of Shehory and Kraus's algorithm. The major drawback of these works is that they optimize only one objective that is the overall utility and the coalition value respectively. However, there are other important objectives that need to be optimized, such as the time needed by robots to perform a given task. Agarwal *et al.* [51] propose an algorithm to form coalitions of robots for a set of tasks. The proposed algorithm tries to maximize the number of tasks completed and the system efficiency. Two multi-objective evolutionary optimization algorithms are introduced to solve this problem: A Non-Dominated Sorting Genetic Algorithm (NSGA-II) and a Strength Pareto Evolutionary Algorithm (SPEA-II). Unfortunately, factors such as the minimizing the number of robots in a coalition is not considered. Service *et al.* [52] propose a simultaneous descending auction-based approach to the task allocation that allows task preemption and does not exhibit any unnecessary task reassignment. However, their proposed algorithm does not take the capability distribution of the robots into consideration.

In [53], authors assign Unmanned Aerial Vehicles (UAVs) to search and prosecute missions. Their objective is to minimize the coalition size and accomplish the tasks in minimum time. However, the cost of UAV deployment is not considered. In [54], authors propose an ant-colony based algorithm. They consider fix number of robots for each task. However, it is not efficient to fix the number of robots required for each task since robots have different capabilities and different capability distribution. Authors in [55], propose an algorithm based on dynamic ANT coalition technique. Minimizing the number of robots in a coalition is not considered. In [56], Rauniyar and Muhuri modify the standard GA. They proposed an adaptive Random Immigrants Genetic Algorithm (aRIGA) and adaptive Elitism-based Immigrants Genetic Algorithm (aEIGA). However, they do not consider minimizing the time needed to perform a task.

B) Application Component Placement over Hybrid Cloud/Fog NFVI

In this subsection, we review the relevant literature on application component placement over hybrid cloud/fog NFVIs. In the first subsection, we review the proposed solutions for application component placement in hybrid cloud/fog systems where these components are not placed as VNFs. We then review the works to date on VNF-FG embedding that do not focus on hybrid cloud/fog systems. To the best of our knowledge, our works (i.e., [14] and [15]) are the only one that investigates the placement of application components as VNFs in hybrid cloud/fog NFVIs.

1) **Application Component Placement in Hybrid Cloud/Fog Systems**

Most of the proposed solutions for application component placement over hybrid cloud/fog systems consider stationary fog nodes, such as IP video cameras, access points, roadside units, etc. Few works have considered the mobility of fog nodes, such as their being located in a moving vehicle. In this subsection, we first review the proposed solutions that consider stationary fog nodes, and then we describe the proposed mechanisms that consider the mobility of the fog nodes.

a. Application Component Placement Considering Stationary Fog Nodes

Several different objectives have been considered in the literature for application component placement over cloud/fog infrastructures. Mahmud *et al.* [57] consider placing application components over cloud and fog nodes such that the user's Quality of Experience (QoE) is maximized. In contrast, Deng *et al.* [58] doing so with the objective of minimizing the power consumption of cloud and fog nodes while taking additional system constraints into consideration, such as the delay at the user's side. Many authors seek to minimize the application response time. Yin *et al.* [59] schedule the tasks over the cloud and the fog infrastructures with the objective of reducing the response time of the tasks under a specified threshold. Similarly, Pham *et al.* [60] schedule the tasks over the cloud/fog system. They aim at minimizing the execution time of a workflow consisting of several interacting tasks. In addition, they minimize the monetary cost of the rented cloud resources. However, they do not consider non-deterministic workflows.

Other objectives have been considered besides optimizing the response time. Agarwal *et al*. in [61] propose an algorithm that distributes the workload over the hybrid cloud/fog system while considering the throughput maximization and the response time minimization. Taneja *et al.* in [62] propose an algorithm for dynamically distributing application components across cloud/fog infrastructures so that the resources are utilized in an efficient manner and the application response

time is minimized. In [63], Skarlat *et al.* model the problem as an ILP and solve it using CPLEX solver. Their goal is to optimize the utilization of fog nodes while satisfying the application QoS in terms of execution time. Authors in [64] and [65] tackle the problem from the perspectives of mobile devices. Hassan *et al.* [64] propose the offloading of application tasks from mobile devices to cloud and fog nodes with the goal of minimizing the application execution time. Similarly, Bittencourt *et al.* in [65] schedule the workload offloaded by mobile users over cloud and fog infrastructures. They present different scheduling strategies to cope with applications with different objectives, such as a delay-priority strategy that prioritizes latency-sensitive applications.

Although these solutions address the problem of application component placement in hybrid cloud/fog systems, they only consider stationary fog nodes with predefined locations. This assumption makes their approach nonfunctional when the system includes mobile fog nodes such as vehicles, UAVs, personal cell-phone devices, or nomadic data centers [66].

b. Application Component Placement Considering Mobile Fog Nodes

Very few works have considered the mobility of fog nodes when placing application components over the cloud/fog system. Zhu *et al.* propose an algorithm to dynamically distribute the application tasks across stationary fog nodes (e.g., roadside units), mobile fog nodes (e.g., busses), and the cloud [67]. They model the problem as a Mixed Integer Linear Programming (MILP) problem with the objective of finding a balance between the application latency and quality loss. The proposed algorithm places individual tasks on cloud/fog nodes; however, in many real-world applications, there are interactions among an application's tasks that require an appropriate component placement mechanism. The authors in [68] propose a computation offloading mechanism for mobile devices by using reinforcement learning. The tasks are offloaded to mobile fog nodes and to cloud nodes such that the service response time and the energy consumption of mobile devices are minimized. The proposed mechanism handles the mobility of the fog nodes by migrating an offloaded task from one mobile fog node to another whenever needed; however, it does not perform mobility aware offloading.

2) **VNF-FGs embedding**

The problem of VNF-FGs embedding in NFV and cloud networks has been studied widely over the last few years. Various objectives have been considered, such as efficient infrastructure utilization [69][70], operational cost minimization [71], VNF instances minimization, [72][11],

and provider revenue maximization [73][74]. In the following, we explain the solution approaches these works have used.

Moens *et al*. in [69] model the problem of placing a batch of VNF-FGs using ILP, which minimizes the infrastructure utilization. Fang *et al.* propose an ILP and a heuristic to solve the VNF-FG placement problem. They consider a balanced utilization of the spectrum of fiber links and infrastructure resources [70]. Embedding VNF-FGs using a minimum number of VNF instances whilst meeting the end-to-end delay requirement was studied by Luizelli *et al*. [72]. Ghaznavi *et al.* solve the VNF placement problem with the objective of minimizing the operational cost of VNF placement while maintaining the QoS [71].

The authors in [73] and [74] maximize the provider's revenue. Sun *et al.* in [73] solve the problem by proposing online and offline methods. In the offline method, all requests are known in advance. The online method uses a prediction of future VNFs and their requirements. Mechtri *et al.* model the VNF-FG embedding problem as a weighted graph matching problem and propose an eigen-decomposition-based approach to solve it [74]. These works are not directly applicable when fog resources are also involved since they implicitly assume that all resources are provided by the cloud. Indeed, using the fog brings two challenges to the problem. 1) The fog nodes' existence in the problem introduces a new type of heterogeneity compared to cloud resources; they have limited resources but provide faster response time. An appropriate allocation mechanism is required to exploit such resources. 2) Similar to what has been discussed about component placement approaches, the fog resources can be mobile [75]. An efficient placement should consider such mobility to avoid high cost or a long makespan as a result of assuming stationary nodes with predefined locations.

2.4. Conclusion

In this chapter, we first presented a motivating scenario, from which we derived a set of architectural and algorithmic requirements. After that, we surveyed the related work. Table 2.1 and Table 2.2 provide a summary of the reviewed architectural and algorithmic papers, respectively. For each paper, we show the requirements which are met and the ones which are not met. As it can be seen, none of the reviewed works satisfy all our requirements.

Table 2.2 Algorithmic related work evaluation

Related Works / Requirements			Algorithmic Requirements						
			General Algorithmic Requirements			Requirements for Coalition Formation		Requirements for Application Components Placement	
			Heterogeneity	QoS	Cost	Resource Optimization	Capability Distribution	Non-deterministic Applications	Mobility
Coalition Formation Algorithms for Robots		**Liu and Chen** [49]	✓	x	✓	✓	x		
		Vig. And Adams [50]	✓	x	x	✓	✓		
		Agarwal *et al.* [51]	✓	x	✓	x	x		
		Service *et al.* [52]	✓	x	x	x	x		
		Manathara *et al.* [53]	✓	✓	x	✓	x		
		Qian and Cheng [54]	x	x	x	x	x		
		P.M and G. R. Suresh [55]	✓	✓	✓	x	x		
		Rauniyar and Muhuri [56]	✓	x	x	x	x		
Application Components Placement	Components Placement over Hybrid Cloud/Fog	**Mahmud *et al.*** [57]	✓	✓	✓			x	x
		Deng *et al.* [58]	✓	✓	x			x	x
		Yin *et al.* [59]	✓	✓	x			x	x
		Pham *et al.* [60]	✓	✓	✓			x	x
		Agarwal *et al.* [61]	✓	✓	x			x	x
		Taneja *et al.* in [62]	✓	✓	x			x	x
		Skarlat *et al.* [63]	✓	✓	x			x	x
		Hassan *et al.* [64]	✓	✓	x			x	x
		Bittencourt *et al.* [65]	✓	✓	x			x	x
		Zhu *et al.* [67]	✓	✓	x			x	✓
		Alam *et al.* [68]	✓	✓	x			x	✓
	VNF-FG Embedding	**Moens *et al*. in** [69]	x	x	x			x	x
		Fang *et al.* [70]	x	x	x			x	x
		Ghaznavi *et al.* [71]	x	✓	✓			x	x
		Luizelli *et al*. [72]	x	✓	✓			x	x
		Sun *et al.* in [73]	x	✓	✓			x	x
		Mechtri *et al.* [74]	x	x	✓			x	x

Chapter 3

3. Cloud-based Architecture for IoT Applications Provisioning

3.1. Introduction

Recently, different robot platforms have been designed for large-scale disaster management applications that typically involve robots in the order of thousands to accomplish a search and rescue mission. However, the cost-efficient provisioning of these applications remains a big challenge, as robots' resources are still seldom used in an efficient manner. Cloud computing can tackle this challenge. Virtualization is a key enabling technology for cloud computing, allowing the abstraction of actual physical computing resources into logical units and enabling their efficient usage by multiple independent users. Virtualization can be performed at both node and network levels. Robot node-level virtualization is defined as the mechanisms that enable multiple applications to reside in and run concurrently on a single robot [25]. And, robot network-level virtualization is the dynamic formation of subsets of robot nodes, with each subset dedicated to a certain application at a given time [25]. The virtualization in the IaaS is achieved by providing a coalition formation algorithm for Multi-Robot Task Allocation (MRTA) problems. The proposed algorithm is described in Chapter 4.

This chapter focuses on large-scale robotic applications as cloud computing services. It proposes an architecture that enables cost-efficient robotic applications provisioning for search and rescue tasks in large-scale disasters management applications with a focus on the IaaS aspects. The proposed architecture incorporates task delegation and network-level virtualization. Task delegation has a remarkable role because a single infrastructure may not provide all the capabilities, and/or the number of robots required to perform a task, and there may be need to use several infrastructures. The Publication and Discovery Engines are key modules of the architecture. They are based on presence technology. They allow different IaaSs to subscribe to the presence information of the robots which is defined as the state of the robots (idle/busy). In order to publish and discover heterogeneous robots, they need to be described. To that end, a well-defined robot description language based on existing standards is proposed. The description language extends SenML. SenML[1] an IETF standard [32], is a Sensor Markup Language that defines media types for representing simple sensor measurements and device parameters. The proposed architecture is implemented as a proof of concepts in order to evaluate its viability and performance level.

The rest of this chapter is organized as follows, we first introduce the overall system architecture including the business model, architectural principles, and architectural modules along with the interfaces and procedures. We then present a well-defined language for robot capabilities' description based on existing standards. After that, we describe the Publication and Discovery Engines. Next, we present the performance evaluation and finally, in the last subsection, we conclude this chapter.

3.2. Overall Architecture for IaaS for Robots

In this subsection, the business model is first introduced, and then the architectural principles are presented. After that, a detailed description of the architectural planes, including the related architectural modules, interfaces, and procedures is discussed.

[1] tools.ietf.org/html/draft-ietf-core-senml-09/

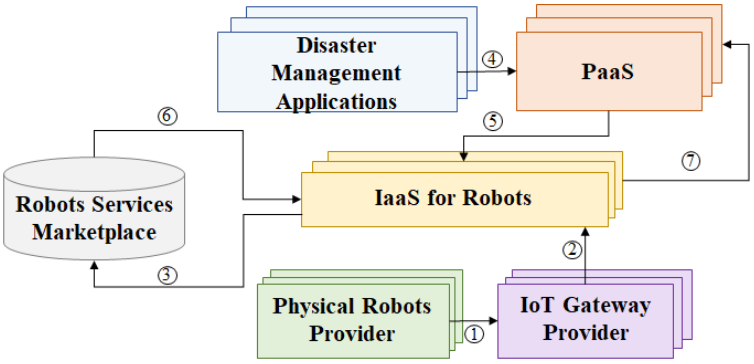


Figure 3.1 The proposed business model

3.2.1. Business Model

The related business model uses and extends the pay-as-you-go cloud model. The robots are provisioned as-a-Service. The specific actors and their relations are schematized in Figure 3.1. *Physical Robots Providers* represent the concrete pool of the heterogeneous physical robots. The *IoT Gateway Provider* provides the required communication gateways to interact with the robots (Figure 3.1, action 1). The virtualization of the robots is performed by the *IaaS for Robots Providers* (action 2). These providers publish their supported robots services in a common *Robots Services Marketplace* (action 3). The marketplace lists and indexes all the available robots for prospective use.

The *Disaster Management Applications* are provisioned as SaaS over the several *PaaS*s (action 4). *PaaS*s interact with the underlying *IaaS*s to settle the required runtimes for hosting and executing these applications. They allocate the necessary robots' services from the *IaaS for Robots Providers* and bound them to the applications (action 5). If the required robot service is not supported by the local *IaaS for Robots Providers*, the latter requests the *Robots Services Marketplac*e (action 6) to get it from another *IaaS for Robots Providers* and deliver it to *PaaS* (action 7).

3.2.2. Architectural Principles

The proposed architecture does not cover PaaS although the IaaS interacts with the PaaS to receive a task request from it. We define two principles for our design:

1- The proposed architecture is publication and discovery technology agnostic, in order to cater for possible future technologies.

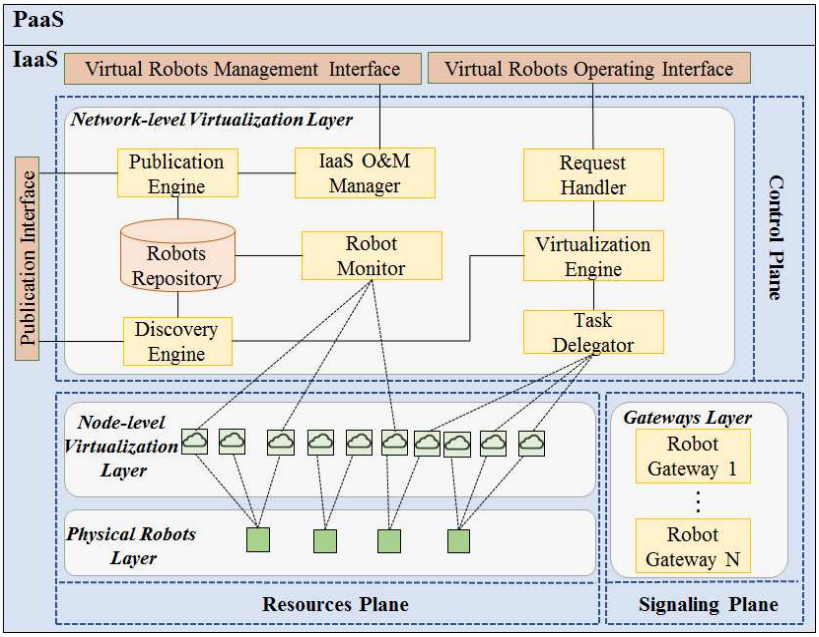


Figure 3.2 The proposed IaaS for robots architecture

2- The interaction interfaces of IaaS, between different layers of IaaS, Publication/Discovery Engines interfaces are REpresentational State Transfer (REST)-based. REST is selected because it is lightweight, standard-based, and can support multiple data representations (e.g., plain text, JSON, and XML).

3.2.3. Proposed Architecture for IaaS for Robots

A) Architectural Modules

The proposed IaaS for Robots architecture is shown in Figure 3.2. It consists of Resources Plane, Control Plane, and Signaling Plane.

1) The Resources Plane

The Resources Plane includes two layers: The *Physical Robots Layer* and the *Node-level Virtualization Layer.* The *Physical Resources Layer* involves the supported robots. It includes the physical heterogeneous robots with their various capabilities and characteristics. The *Node-level Virtualization Layer* contains the pool of the virtualized robots.

2) The Signaling Plane

The Signaling Plane contains a set of communication gateways called *Robot Gateways*. These gateways allow of hiding the heterogeneity and specificities of the robots in terms of user APIs,

communication protocols, and so on. Their role is to map between the *Network-level Virtualization Layer* (in the Control Plane) requests and the proprietary robots' APIs. It receives the task assignment request from the *Network-level Virtualization Layer* by standard interfaces and sends the request to robots, based on the interface supported by the desired robot. The gateways are (un)instantiated on-demand in accordance with the evolution of the applications' workload and the used robots. Their design is based on our work in Chapter 4 and 5 (i.e., [8] [9] [10]).

3) **The Control Plane**

The Control Plane includes the *Network-level Virtualization Layer*. It has the following modules:

- **The IaaS O&M Manager**: responsible for adding the supported robots to the IaaS or removing them from it. For instance, when a new robot service is added, this module parses the robot metadata (e.g., communication protocol, list of capabilities) and generates a descriptor based on a well-defined model.
- **Publication Engine**: stores locally the robots service descriptor and publishes it to different IaaSs through the remote marketplace.
- **Discovery Engine**: discovers robots services in local and different IaaSs. It runs on the local repository to get the local services descriptors and on the remote marketplace to get the available robots services descriptors from the remote marketplace.
- **Request Handler**: responsible for analyzing the upcoming requests from PaaS and providing a set of inputs, such as task requirements and constraints to the Virtualization Engine.
- **Virtualization Engine**: performs the network-level virtualization of the robots' capabilities. This is done by running an appropriate algorithm for coalition formation in multi-robot systems. The algorithm is designed and implemented as part of our work (i.e., [7], described in Chapter 6). It runs the algorithm on both local robots and those belonging to other IaaSs.
- **Task Delegator**: sends task assignment requests to local robots and to robots in other IaaSs. It may also receive task assignment requests from other IaaSs whenever needed.
- **Robot Monitor**: responsible for monitoring the robots in the *Physical Resources layer*. A robot basically sends a notification to this module when it finishes its sub-task or fails. Accordingly, the robots' availability is updated in the local *Robots Repository* and in the external *Robots Services Marketplace*.

Table 3.1 Static characteristics representation using extended SenML

	SenML	*JSON*	*Type*
Static Characteristics	Physical Char.	ph	Array
	Sensors	sen	Array
	Actuators	act	Array
	Personal Info.	info	Array

	SenML	*JSON*	*Type*	*e.g.*
Sensors	Sensor Name	sname	string	Camera, microphone
	Sensing Value Range	sval	string	(min, max)
	Sensing Unit	su	string	Hz for microphone

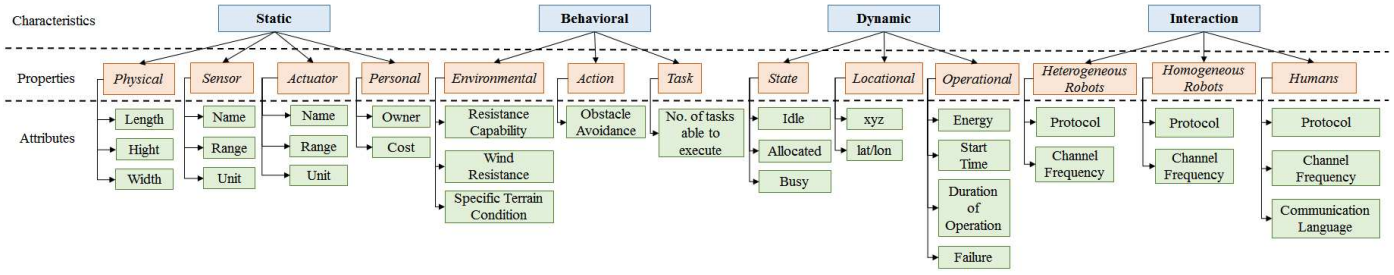


Figure 3.3 Extended SenML for unified robots description model

B) Interfaces

All the interfaces are designed according to the REST principle [76]. They all expose CRUD (i.e., Create, Read, Update, and Delete) operations.

- The *Virtual Robots Management Interface* is a management interface that allows administrators to add/remove robots to IaaS. It also allows an IaaS to delegate tasks to robots in other IaaSs.
- The *Publication Interface* allows IaaS to (un)publish its robots in the remote marketplace. Its detailed description is provided in Section 3.2.5 (B).
- The *Virtual Robots Operating Interface* exposes to the PaaS control operations to request robots' services from IaaS.

Finally, the proposed IaaS for robots reuses and adapts the regular control and signaling IaaS interfaces. The interface between the *Network-* and the *Node- level Virtualization Layers* is one example. *Robot Monitor* and *Task Delegator* modules interact with the local robot through this interface. For instance, the resources in the *Node-level Virtualization layer* side are used to reserve

robot resources when adding new robots, when creating a group of robots, and when sending a task to a specific robot or group of robots, or to modify the resources for an ongoing task.

C) Procedures

This section discusses four main functional procedures: Idle Robot Discovery, Selecting Robots for a Given Task, Task Assignment for the Selected Robots, and Notification of Finished Task. Idle Robot Discovery is used by a *Publication and Discovery Engines* to publish and discover robots along with their characteristics. Selecting Robots for a Given Task is performed by the *Virtualization Engine*, which runs a coalition formation algorithm for multi-robot systems. The Task Assignment for the Selected Robots is the procedure of assigning tasks to the selected robots. Notification of Finished Task, is sent by the robots when they finish their task.

3.2.4. Standard Description of Robots

The considered physical robots are developed with different platforms. More importantly, they have different capabilities, sizes, and shapes. Applications should be able to deal with robots' heterogeneity. To allow this, there is a need for a standard and well-defined description of robots to publish and discover robots with different characteristics and capabilities. Accordingly, a common model that unifies the robotic characteristics description is designed. The relevant literature considers developing ontologies for the standard description of heterogeneous resources (e.g., [35] for the specific case of robots). Although the semantics enable powerful and faithful modeling, its overhead in terms of processing and developing and maintaining new components is important. In our approach, we extend SenML to describe these robots. It is lightweight and can be parsed efficiently, which makes it more suitable for the robots' description.

The unified description model is implemented in the *IaaS O&M Manager*. It is this module that generates the robots' descriptor to be stored in the local *Robots Repository* and in the remote *Robots Service Marketplace*. We try to cover most of the characteristics of robots. We categorize robots' characteristics into static, behavioral, dynamic, and interaction characteristics. Each characteristic includes a list of properties and each property may include one or several attributes. Figure 3.3 shows the scheme of this model, with some examples of attributes for each property. Table 3.1, for instance, details the properties of Static characteristic and its Sensor attribute.

Table 3.2 Examples of the API operations on the publication interface

REST Resource	Operation	HTTP Action and Resource URI
Robot Presence Information	Create: PUBLISH presence information for newly added robots to the IaaS (joining publication)	POST: /robots
	Update: re-PUBLISH presence information of a robot, update an already created resource. (state change publication)	PUT:/robots/{robotid}
	Read: SUBSCRIBE for presence information of a robot	POST:/robots/{robotid}?fromuri={subscriberuri}
	Read: SUBSCRIBE to a list of robots	POST:/robots?fromuri={subscriberuri}
	Delete: un-SUBSCRIBE from presence info of a robot	DELETE:/robots/{robotid}/{subscribeid}
	Delete: un-SUBSCRIBE from presence info of list of robots	DELETE:/robots/{subscribeid}

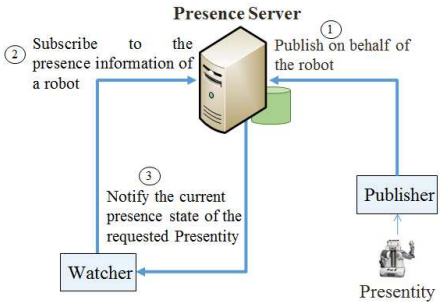


Figure 3.4 Presence technology-based architecture for Publication/Discovery

3.2.5. Publication/Discovery Engine

The publication/discovery mechanisms are based on the presence technology [77]. The presence service is chosen as it allows the discovery to be speedy and ahead of time. It allows each IaaS to publish its local robots whenever they change their state. This guarantees that the relevant robots are already discovered when IaaS receives a new search and rescue task. In the proposed architecture, the presence server is provided by a third-party tier (i.e., the *Robots Services Marketplace*). The *Publication and Discovery Engines*, as part of IaaS, are the clients that interact with the presence server.

A) Architectural Modules

Figure 3.3 shows a presence-based architecture for publication and discovery. The *Presentity*

represents a robot. It is the source of the presence information to be stored and distributed by the presence server. The *Watcher* represents the *Discovery Engine*. It subscribes to a *Presentity* (i.e., robot) to receive its presence information along with its characteristics from the *Presence Server*. The *Publisher* represents the *Publication Engine*. It publishes the robots' presence information along with their characteristics on behalf of the robots in the *Presence Server*. It uses the SenML-based descriptors stored in the *Robots Repository*. This design allows each IaaS to publish its local robots whenever they change their state.

B) Interfaces

The *Publication Interface* is defined as a REST interface. We re-use the interfaces defined in [78] and modify them according to our architecture. Table 3.2 details the list of the *Publication Interface* operations. These operations allow the *Publication Engine* to publish and update the presence information of its robots and allow the *Discovery Engine* to (un)subscribe to the presence information of robots or list of robots belonging to other IaaSs.

C) Procedures

We define the following procedures: Subscription, Joining Publication, State Change Publication, and Notification of State Change. In Subscription, each IaaS through its *Discovery Engine* (i.e., the Watcher) subscribes to the presence information of the robots (i.e., the Presentity) belonging to other IaaSs in order to receive notification when a robot changes its presence information. The Joining Publication is the procedure for robots to publish their presence with all their characteristics (static, behavioral, etc.) to the Presence Server when they are first purchased and joined an IaaS. In the State Change Publication procedure, when robots change their states, they publish their presence to the Presence Server along with only their dynamic characteristics. The last procedure is the Notification of State Change, where the Presence Server notifies the current presence state of the requested Presentity to the Watcher. Figure 3.5 represents an end to end scenario, where both overall architecture procedures and presence service procedures are demonstrated.

3.3. Performance Evaluation

This section discusses the performance evaluation performed to validate our proposed architecture. We first discuss the implementation alternatives. Then, we describe the prototype and the setup related to our experimentations. After that, we describe the evaluation of the architecture along with the metrics and the obtained measurements.

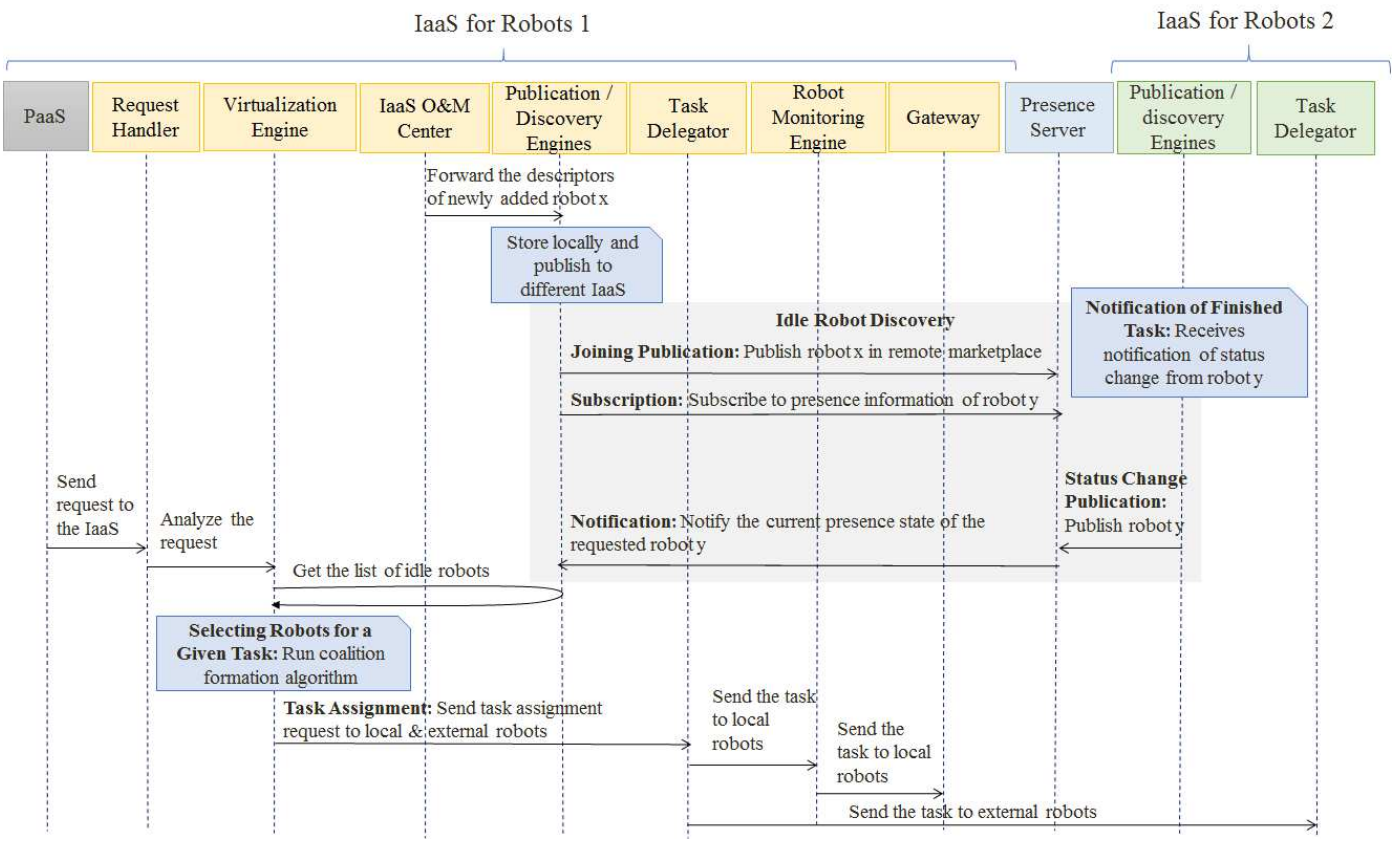


Figure 3.5 Illustrative sequence diagram for end to end scenario

3.3.1. Implementation Alternatives

During or after a disaster, the telecommunication infrastructure is most likely to crash because of physical destruction or congestion of the network components [79]. This could block out the system and therefore, prevents the dispatching of the robots. In order to maintain the communication between the application deployed in the PaaS and the robots, a potential solution is to diversify the routing path. Such as establishing several network connections between PaaS and IaaS, using either different type of technologies or following different redundant physical paths. The US Department of Information Technology and Telecommunications has recommended developing ducts in order to expand route diversity [80]. Another alternative is to route the communication over ad-hoc networks. These networks can be provided by the tactical radio networks of civilian or military responders. Such as the tactical communication system proposed by CISCO for disaster situations [81] and the emergency communication system based on Software Defined Radio (SDR) network [82][83]. The robots can be deployed using a helicopter such as T-Rex miniature helicopter used by CARSAR to deploy unmanned

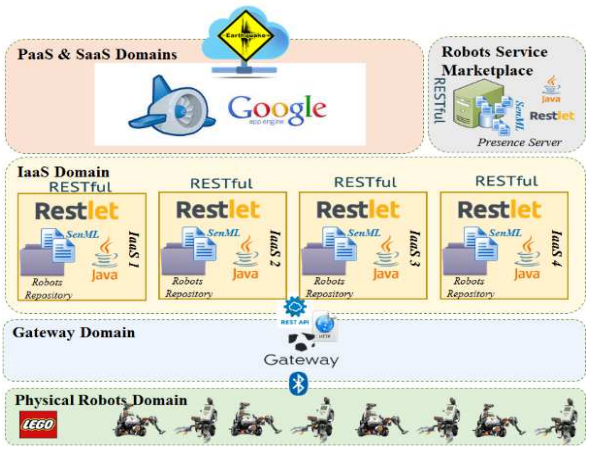


Figure 3.6 Prototype architecture

aerial systems to the areas affected by Hurricane Katrina [84].

We assume that the request sent by the PaaS to the IaaS can be done via the Internet if the Internet access is still available. Otherwise, the request is sent through ad-hoc networks. For the ad-hoc network creation, several approaches have been proposed in the relevant literature. Such as by using communication tags proposed by Miyama *et al.* [85], or by establishing a Base Station (BS) within a safety zone around the disaster area proposed by Sugiyama *et al.* in [86]. The presence technology ensures that, when an IaaS receives a request, it does not need to communicate with other infrastructures to get their robots information. It ensures that the IaaS has the latest information about robots. However, since the communication path may be destroyed between all or some of the IaaSs, the remote *Robots Services Marketplace* may not receive the Notification of State Change for all robots, hence it may not have the latest information regarding the robots. This can be solved by integrating data replication strategies such as those used for fault tolerance systems [87] [88].

3.3.2. Proof of Concept Prototype

The prototype implements a firefighting task of the fire detection and fighting application explained in the motivating scenario in Chapter 2. Earthquakes are often followed by fire with devastating consequences especially in townscape environments (e.g., Kesennuma City in Miyagi, Japan, 2011). The prototype architecture is depicted in Figure 3.6. In the *Physical Robots Domain*, the considered robots are LEGO Mindstorms NXT[2]. Two types of robots are used: one with arms

[2] lego.com/en-us/Mindstorms/

and a movement motor and another with light sensors, kicking arms, and a movement motor. They carry plastic balls as water extinguishers. In the *IaaS Domain*, four distinct infrastructures are implemented. The inter-domain architectural modules of IaaS (e.g., *Request Handler*, *Task Delegator*) are implemented as RESTful Web services using Java Restlet framework. The rest of the modules (e.g., *Virtualization Engine)* are developed as regular Java tools. The local *Robots Repositories* are simple OS folders that store the SenML-based descriptors of the supported robots' services. In the *Gateways Domain*, an appropriate *Robot Gateway* is settled to map between the IaaS HTTP Java REST and LeJOS NXJ Java API commands that implement the Lego Communication Protocol (LCP). The *Robots Services Marketplace Domain* provides the presence server. It is implemented as RESTful Web Service using Restlet framework. It also includes a storage folder of the SenML descriptors of the published robots' services.

In the *PaaS Domain*, Google App Engine[3] (GAE) is used. It hosts and executes the fire detection and fighting application. Internet connection is assumed available between GAE and IaaS. A Network Address Translation (NAT) server is developed to redirect the requests coming from GAE to *IaaSs*. In the *SaaS*, the fire suppression sub-task requires the light sensor capability to detect the balls and the kicking arms and movement capabilities to handle and move them. We implemented all the procedures described in Sections 3.2.3 (c) and 3.2.5 (c).

3.3.3. Experimentation Setup

Four machines belonging to the same LAN are used, each host one IaaS for Robots. The first machine executes *IaaS1* and *NAT Server*. The second executes *IaaS2* and *Presence Serve*r. The third and the fourth respectively execute *IaaS3* and *IaaS4*. One of the machines has two interfaces: one with a public IP to communicate with the application and the other with a private IP (the LAN interface). The other machines have only a private IP. All machines run on Windows 7 Professional and have an Intel® Core ™i7-2620 CPU with 2.70Hz and 8 GB of RAM. We have used three LEGO Mindstorms NXT robots [89], one in each IaaS. The communication with the robots is done through the Gateway via Bluetooth.

To properly evaluate it, the prototype is compared it with a peer-to-peer (P2P) overlay network. Consequently, an overlay node corresponding to each IaaS is implemented. An overlay node is

[3] appengine.google.com/

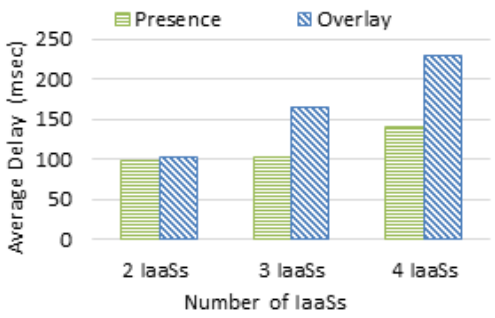


Figure 3.8 Idle Robot Discovery Delay (IRDD)

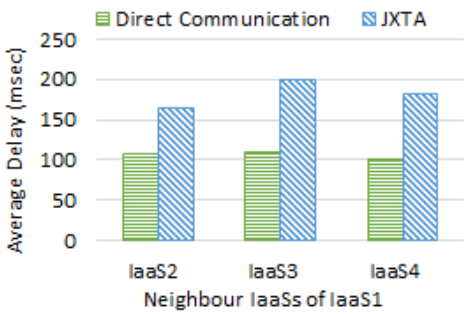


Figure 3.7 Task Assignment Delay (TAD)

implemented using the JXTA protocol (JXSE 2.6). The publication and the discovery procedures are carried out by the JXTA advertisement. The task assignment procedure is mapped to the JXTA messages that are exchanged through JXTA bidirectional pipes.

3.3.4. Measurements and Results

A) Performance Metrics

The performance metrics according to which we evaluate the system performance are:

- *Idle Robot Discovery Delay (IRDD)* (msec) - the time difference between the moment an IaaS for Robots subscribes to the presence information for the Presentities in the other IaaSs for Robots, and when it receives notification of the current state of the requested Presentities.
- *Task Assignment Delay (TAD)* (msec) - the time difference between the moment an IaaS for Robots sends a task assignment request and when other IaaSs for Robots receive this request.

B) Results and Discussion

Test case 1 - Idle Robot Discovery Delay (IRDD): Figure 3.7 shows the average time for IRDD using a different number of IaaSs for Robots. As is noticed, for any number of IaaSs, the average IRDD for presence-based publication/discovery is less than the delay for P2P overlay-based publication/discovery. This is because overlay networks have additional costs caused by the communication overhead. They add an intermediate level between the IaaSs. It is also observed that the average IRDD using P2P overlay increases as the number of IaaSs increase since IaaS1 should discover robots in more than one IaaS. The overlay nodes add additional overhead due to the processing of each packet. This shows the viability of using a presence technology-based publication/discovery

Test case 2 - Task Assignment Delay (TAD): Figure 3.8 shows the average delay for TAD. In this test case, the average delay is calculated for the *Task Delegator* in the three IaaSs required to receive the task assignment request from the *Task Delegator* in IaaS1. It is observed that the average delay for direct communication remains almost the same for the three IaaSs. The involved IaaSs communicate with each other directly, i.e., point-to-point. So, all IaaSs have the same delay. Moreover, this delay is far less compared to P2P overlay-based task assignment. To be received by each IaaS, the task assignment request needs to go through the overlay that adds overhead and increases latency in the system. This shows the viability of the proposed method for task assignment.

3.4. Conclusion

In this chapter, we investigated the problem of robotic applications provisioning in a cost-efficient manner. We proposed an IaaS for Robots architecture that enables network-level virtualization and task delegation to robots in different IaaSs. We also proposed a presence technology-based publication/discovery and provided an extension to SenML to describe heterogeneous robots. The experiments' results show the feasibility of our proposed architecture.

Chapter 4

4. NFV-based Centralized Architecture for IoT Gateway

4.1. Introduction

Research on sensor network virtualization [90] has become prominent in recent years. Virtualization technology abstracts sensor resources as logical units and allows for their efficient and simultaneous use by multiple applications, even if they have conflicting requirements and goals. New applications can be deployed in the same WSN with minimal efforts. More importantly, reusing the same sensors' capability by multiple applications transforms WSN into a multi-purpose sensing platform in which several virtual WSNs (VWSNs) are created on-demand, each tailored for a specific task or objective. Actuators are often incorporated in WSNs to make more powerful applications, thus the concept of virtualized wireless sensor and actuator network (VWSAN). Gateways are required for the interactions between applications and heterogeneous, multivendor VWSANs. They are generally complex. Furthermore, it is difficult and expensive to upgrade them when new-brand sensors and actuators/robots are deployed. In addition, their capabilities do not scale when the number of applications and the corresponding workload in

VWSANs change dynamically. NFV can aid in overcoming the aforementioned challenges. On-the-fly, dynamic, scalable, and elastic provisioning of network services are among its benefits.

This chapter presents an NFV architecture for VWSAN. The firmware/hardware used to provide VWSAN Gateway functionalities are replaced by VNFs deployed in an NFV infrastructure. We enable a granular provisioning of NFV, such as decomposing the gateway into fine-grained modules – e.g., protocol converter, information model converter, etc. – to be implemented as VNFs. More importantly, granular NFV is best suited for virtualized WSANs, wherein the dynamic growth in the number of applications and addition of new-brand sensors require a rapid introduction of new VNFs and update of existing VNFs. VNFs are instantiated on-the-fly and chained to realize a service in VWSAN. The architecture introduces a new business actor - the VWSAN Gateway Provider – in addition to the traditional actors, meaning the Application Provider and the VWSAN Provider. This new actor plays a dual role. On the one hand, it provides the VNFs, chained to make on-the-fly gateways. On the other hand, it operates and manages the infrastructure in which the VNFs are executed. We acknowledge that the introduction of this new actor does bring a host of additional security and trustability challenges. We consider these challenges outside the scope of this work. More and more standardization work will certainly be required to enable secure and trustable interactions between different NFV actors, as the business model opens up. The proposed architecture is implemented as a proof of concepts in order to evaluate its viability and performance level.

The rest of this chapter is organized as follows: in the first subsection, we present our proposed NFV-based architecture for virtualizing WSAN gateways. The architectural principles are discussed first, followed by architectural modules and interfaces, VNF migration and scalability issues, and control plane. Next, we present the performance results. Finally, we conclude this chapter.

4.2. Overall Architecture for Virtualized WSAN Gateway

The overall architecture for NFV-based VWSAN Gateway is shown in Figure 4.1. It comprises several Application Domains, a VWSAN Gateway Provider Domain, and VWSAN Provider Domains. In this section, we first describe the architectural modules and interfaces, followed by a discussion of the VNF migration process and the scalability issues. We then present the control plane and finally, we provide an illustrative scenario.

4.2.1. Architectural Principles

We defined the following set of principles:

1- Granular provisioning of network functions. We aim to use highly granular VNFs for virtualized WSAN gateway functions. Examples include protocol conversion and information model conversion. The protocol converter decodes a packet received in one protocol and encodes it in another protocol. The information model conversion converts data from one format to another. We do acknowledge the fact that converting a protocol X (or an information model X) into a protocol Y (or an information model Y) is not always feasible. Consequently, the Gateway Provider provisions the related VNFs only when the conversion is feasible.

2- The VWSAN Gateway Provider maintains a centralized store of VNF images. VNFs are dispatched on-demand to the VWSAN provider's domain. This principle is in accordance with the ETSI, that VNFs must be deployed throughout the networks where they are most effective and highly customized to a specific application or user [22].

3- The interaction interfaces between different domains are REST-based. Similar to the previous chapter (i.e., Chapter 3), REST is selected because it is lightweight, standard-based, and can support multiple data representations (e.g., plain text, JSON, and XML).

4.2.2. Proposed Architecture for NFV-based Virtualized WSAN Gateway

A) Architectural Modules

Each Application Domain contains an application that requires the services of one or more VWSAN providers. The Application contains two modules: *Infrastructure Agent* and *Sensor/Actuator Agent*. The *Infrastructure Agent* is responsible for the signaling procedure. It communicates with the VWSAN Provider Domain to negotiate the use of VWSAN infrastructure. The *Sensor/Actuator Agent* is responsible for gathering measurements from the sensor and sending commands to robots. The VWSAN Gateway Provider Domain consists of the following entities:

- **Core Layer:** contains VNFs and their corresponding Element Management Systems (EMS), where each EMS is responsible for monitoring the resource utilization of its corresponding VNF [22].
- **NFV Infrastructure (NFVI):** provides hardware and software resources, including computation, storage, and networking needed to deploy, manage, and execute VNFs.

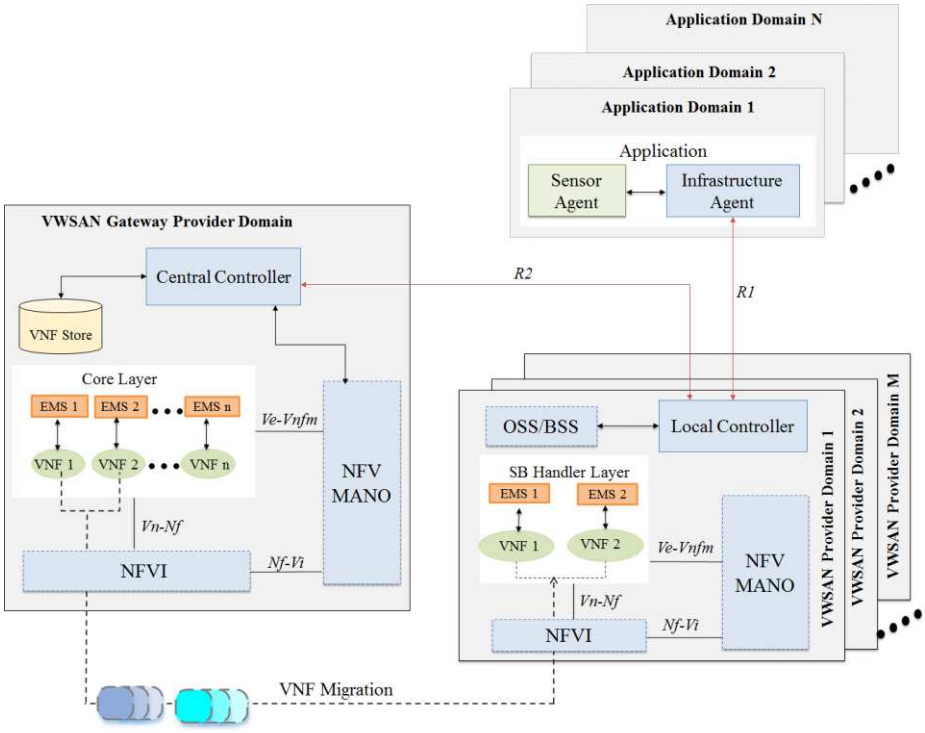


Figure 4.1 The proposed NFV-based IoT Gateway

- **NFV Management and Orchestration (MANO):** responsible for orchestration and lifecycle management of physical/software resources, and the lifecycle management of VNFs. (instantiation, update, migration, and termination).
- **Central Controller:** performs functions as part of the signaling procedure that occurs during service negotiation (this is described later).
- **VNF Store:** a repository that contains VNFs of various gateway modules. It provides VNFs that match the requirements of an end-to-end service.

Each VWSAN Provider Domain comprises the following modules:

- **Southbound (SB) Handler Layer:** contains VNFs that have been migrated from the VWSAN Gateway Provider Domain and their corresponding EMSs.
- **NFVI:** (explained in the previous section).
- **NFV MANO:** performs the typical orchestration and management functions for the execution of migrated VNFs.
- **Operational Support System/Business Support System (OSS/BSS):** provides the description of VWSAN (e.g., sensor/robot brands).

Table 4.1 Resources on the VWSAN Provider Domain and VWSAN Gateway Provider Domain

Domain Name	*REST Resource*	*Operation*	*Http Action & Resource URI*
Resources on VWSN Provider Domain	List of applications service requests	Create: add application information (protocol used, data format, latency, etc.)	POST: /ApplicationsServiceRequests
	Specific application's service request	Update: Change information of specific application	PUT: /ApplicationsServiceRequests /(RequestId}
		Delete: Delete specific application information	DELETE: /ApplicationsServiceRequests /(RequestId}
	Notification of service availability	Create: Send notification to VWSN domain by the gateway domain about the availability of requested VNFs.	POST: /ServiceAvailabilityNotification
Resource on VWSN Gateway Provider Domain	Request for VNFs	Create: send request from VWSN domain to gateway domain for VNFs with specific information (protocol, data model, etc.)	POST: /VNFsRequest
	Specific request for VNFs	Update: Change information of specific request for VNFs.	PUT: /VNFsRequest/{VNFsRequestId}
		Delete: delete information of specific request for VNFs.	DELETE: /VNFsRequest/{VNFsRequestId}

- **Local Controller:** interacts with the *Infrastructure Agent* and the *Central Controller*.

B) Interfaces

The NFV modules interact with each other through the interfaces defined by ETSI [22]. They include: (1) Vn-Nf which represents the execution environment provided by NFVI to Core Layer and to SB Handler Layer. (2) Nf-Vi which is used for assigning virtualized resources in response to resource allocation requests (e.g., allocating VMs on hypervisors), it is also used by NFVI to communicate status information about virtualized and hardware resources to the MANO. Nf-Vi is also used to configure hardware resources. And (3) Ve-Vnfm which carries out all operations during a VNF life cycle. It is also used for exchanging VNF configuration information.

4.2.3. VNF Migration and Scalability Issues

A) VNF Migration

In the architecture, VNFs are migrated on-demand from VWSAN Gateway Provider Domain to VWSAN Provider Domain. The architecture supports two approaches for migration. In the first approach, VNFs are instantiated and chained in VWSAN Gateway Provider Domain. Then, using live migration, running VMs are sent from the VWSAN Gateway Provider Domain to VWSAN

Provider Domain. In the second approach, VNFs are migrated from the VWSAN Gateway Provider Domain to VWSAN Provider Domain, where they are instantiated and chained.

B) Scalability

The architecture relies on dynamic resource allocation algorithms to meet the growing demand of applications. These algorithms enable vertical scaling and/or horizontal scaling. Existing algorithms such as [11] and [12] could be used as a basis. We consider the design of these algorithms as items for future work.

4.2.4. Control Plane

The control plane consists of signaling procedure and control interfaces; R1 and R2. In a typical end-to-end service, the application sends a query to sensors to receive measurements and deploy robots. Before the service begins, a signaling procedure is conducted, in which different business players (i.e., Application Domain, VWSAN Provider, and VWSAN Gateway Provider) engage in service negotiation and exchange the necessary parameters to obtain the appropriate VNFs.

A) Signaling procedure

Signaling is initiated when an application requires services from VWSAN Provider Domain. The *Sensor/Actuator Agent* instructs the *Infrastructure Agent* to start the service negotiation. The *Infrastructure Agent* creates a service request that includes a description of the northbound interface used by the application (i.e., communication protocol, information model, etc.) and QoS parameters associated with the service delivery (i.e., latency, throughput, etc.) and sends it to the *Local Controller*. Upon receipt of the service request, the *Local Controller* communicates with the *OSS/BSS* to obtain information on parameters specific to the VWSAN (e.g., type of sensors/robots). It then creates a VNF request containing parameters of the service request as well as parameters specific to the VWSAN and sends it to the *Central Controller*. Based on these parameters, the *Central Controller* searches for appropriate VNFs in VNF Store.

If the VNFs are found, the *Central Controller* instructs *NFV MANO* of VWSAN Gateway Provider Domain to instantiate and migrate the VNFs to VWSAN Provider Domain. The *Central Controller* also receives a notification from *NFV MANO* of VWSAN Gateway Provider Domain when the VNFs are ready for use in VWSAN Provider Domain. The *Central Controller* then forwards the notification to the *Local Controller*, which sends a notification about service

availability to the *Infrastructure Agent*. The latter notifies the *Sensor/Actuator Agent* to start the service. It is important to note that, when the required VNFs are not found in the VNF Store, a service unavailability notification is sent to the *Infrastructure Agent*, to either cancel the negotiation or resume signaling after a certain time period.

B) Control Interfaces

R1 is used for the interactions between *Infrastructure Agent* and *Local Controller*. R2 is used for the interactions between *Local Controller* and *Central Controller*. R1 and R2 are based on REST paradigm. The required information is modeled as resources and each resource is uniquely identified by the Uniform Resource Identifier (URI). Table 4.1 summarizes the proposed REST interface for the interactions between different domains. It defines resources on VWSAN Provider Domain, used to reserve resources when it receives a service request from Application Domain with a description of parameters. They also allow the Application Domain to modify parameters and delete resources of specific applications. Furthermore, they allow VWSAN Gateway Provider Domain to send notifications to VWSAN Provider Domain about the availability of the requested VNFs. The resources defined on VWSAN Gateway Provider Domain allow it to receive VNF requests from VWSAN Provider Domain. They also allow the VWSAN Provider Domain to update or delete information (e.g., sensor/robot brand) about specific VNF requests.

4.2.5. Illustrative Scenario

In Figure 4.2, we illustrate an end-to-end scenario, wherein an application (e.g., earthquake early warning and recovery) queries the sensors owned by VWSAN Provider 1 and collect their measurements, and another application (e.g., fire detection and fighting) needs to be notified when a fire occurs and deploy robots. Before using VWSAN Provider Domain's service, the signaling procedure starts. The northbound interface description sent to the *Local Controller* for both sensors and robots is SenML over HTTP. Since the current SenML implementation only supports sensor measurements, we have used the extended capabilities of SenML proposed by Datta et al. in [91] and [92] to send robot commands from the application.

Upon receiving the description from *Infrastructure Agent*, the *Local Controller* obtains a description of the sensors (i.e., SunSpot) and the robots (i.e., Lego Mindstorms) from *OSS/BSS*. The signaling procedure continues as described in Section 4.2.4 (B) for both applications. For VNF migration, the second approach (see section 4.2.3 (A)) is used; the VNFs are instantiated, chained,

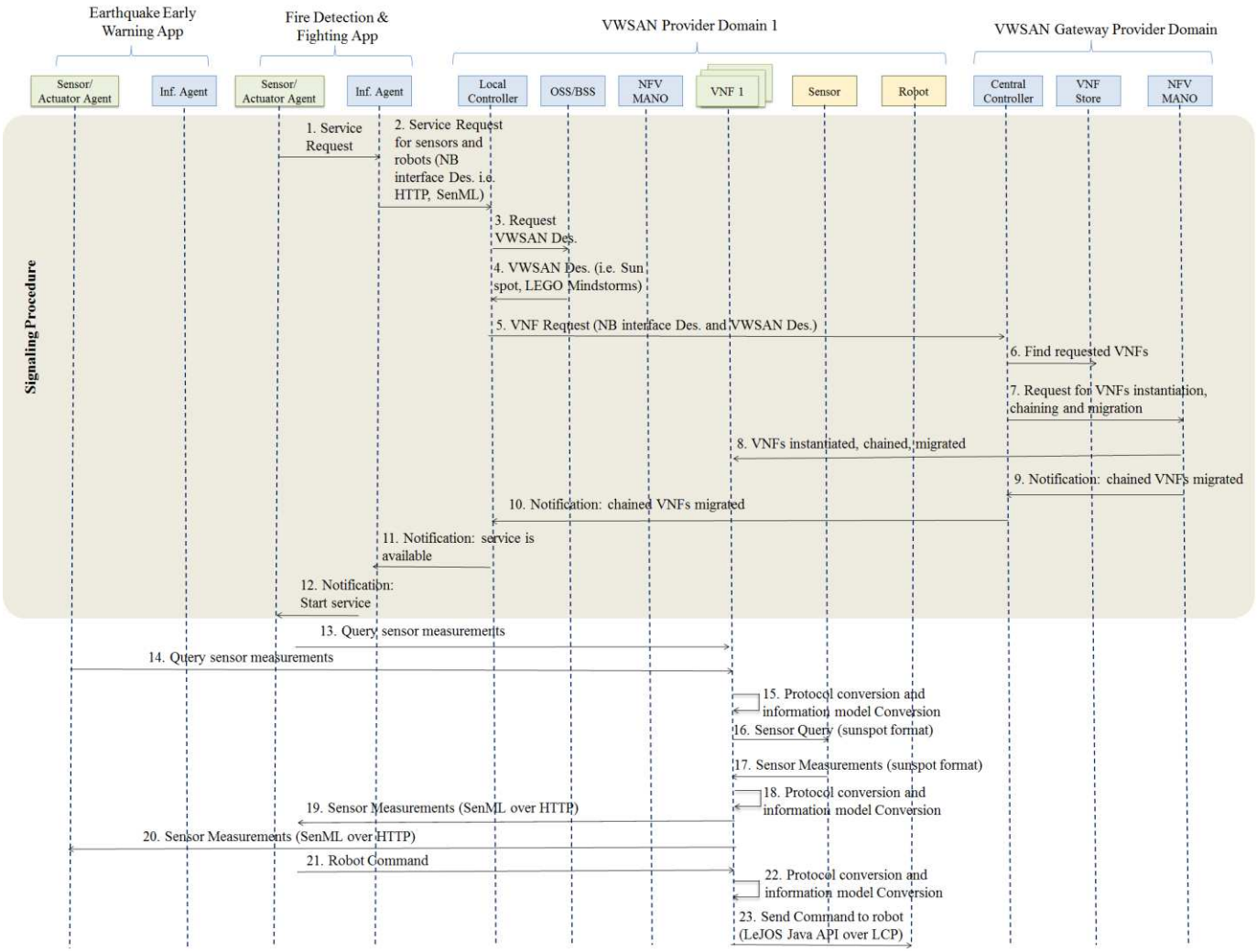


Figure 4.2 Sequence diagram for an end to end scenario

and then migrated to VWSAN Provider Domain. After service negotiation, the *Sensor/Actuator Agent* sends a query to the sensors through the VNFs. Upon receiving the query, SunSpot sensors send their raw measurements over CoAP protocol. These measurements are processed by protocol conversion (encoded in HTTP protocol) followed by information model conversion (mapped to SenML format), in order to enable the applications to interpret the measurements. If the fire detection and fighting application receives notification of fire, it sends actuating commands to the robots in the SenML format through HTTP, where the commands are mapped to LeJOS Java API and to Lego Communication Protocol (LCP). The end-to-end service is completed when the robots are deployed.

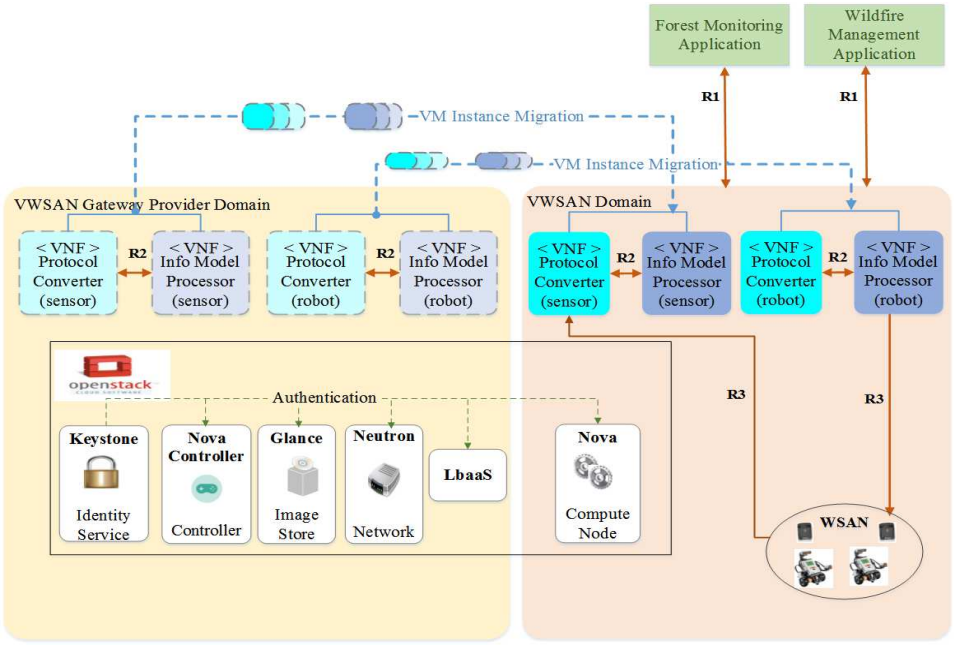


Figure 4.3 Prototype architecture

4.3. Performance Evaluation

In this section, we first present the prototype that we built followed by the experimentation setup. After that, we discuss the measurements and the results.

4.3.1. Proof of Concept Prototype

For the prototype, we implemented the scenario described in Section 4.2.5, in which the earthquake early warning and recovery application is interested in collecting environmental data to monitor different earthquake-prone areas and a fire detection and fighting application that needs to be notified when a fire occurs and deploy robots in order to suppress it. We assume both applications are interested in collecting the temperature data.

The applications were created using java dynamic web application and hosted on Tomcat8 server. We used OpenStack [4] Icehouse to build our private cloud. OpenStack is a free, open-source software for creating private and public clouds. Figure 4.3 depicts our prototype architecture. We used a multi-node OpenStack with two compute nodes. We considered each compute node as a domain: One as VWSAN Provider Domain and the other as VWSAN Gateway Provider Domain.

[4] https://www.openstack.org/

In our prototype, we assume the two domains are in the same data center. In order to provide live migration, both compute nodes share the same storage. This allows the migration of only the memory footprint of the VM. If each domain were in a separate data center, we would assume a provision for live migration among them. The VNFs are instantiated in VWSAN Gateway Provider Domain and migrated to VWSAN Provider Domain after being chained. For simplicity's sake, we assume that the VNFs are chained in a static way in VWSAN Gateway Provider Domain.

In the node representing VWSAN Gateway Provider Domain, all necessary components of OpenStack were installed. NFS (Network File System) server was also configured in this node, allowing servers to share directories and files with each other over a network. The two nodes representing VWSAN Provider Domain contains only Nova. The fourth node is configured as Neutron and LBaaS (Load Balancing as a Service) was installed on it, which is a service of Neutron, allowing to load balance traffic for services running on VMs in OpenStack. We used OpenStack4j API, as an open-source OpenStack client, allowing the provision and control of an OpenStack system as a controller. Because all domains are in the same data centers, the controller can control all domains. Each VNF runs a Linux Ubuntu V14.04 on 1 VM and is equipped with 1 VCPU and 2GB RAM. The VNFs communicate with each other through a REST interface (R2), using the RESTlet framework [13]. Communication between VWSAN Provider Domain and Application Domains is also achieved via REST interface (R1).

4.3.2. Experimentation Setup

The applications and the domains controller run on a PC with Intel® Xeon® CPU clocked at 2.67 GHz and a 6GB RAM with 64-bit Windows 7 Enterprise. This PC uses JVM version 1.8.0_51. We used four PowerEdge™ T410s, which are Intel® processor-based servers – two as nova compute nodes, one as the Nova controller, and one as the network node. Two Java Sun SPOT sensors, two Advanticsys sensors, and one LEGO Mindstorms NXT robot were used. Each sensor executes the environment monitoring task. We implemented a simple gateway that runs on a laptop with Intel® Core ™i7-2620 CPU with 2.70Hz and 8 GB of RAM. This gateway exposes the robots' and the sensors' capabilities as APIs. For example, in order to send a command to the robot, the protocol converter and information model conversion convert the REST request received at its northbound interface to LeJOS Java API commands that implement the LCP. The gateway then wraps the request to either Bluetooth or USB communication channel and sends it to the robot.

4.3.3. Measurements and Results

A) Performance Metrics

The performance metrics according to which we evaluate system performance are:

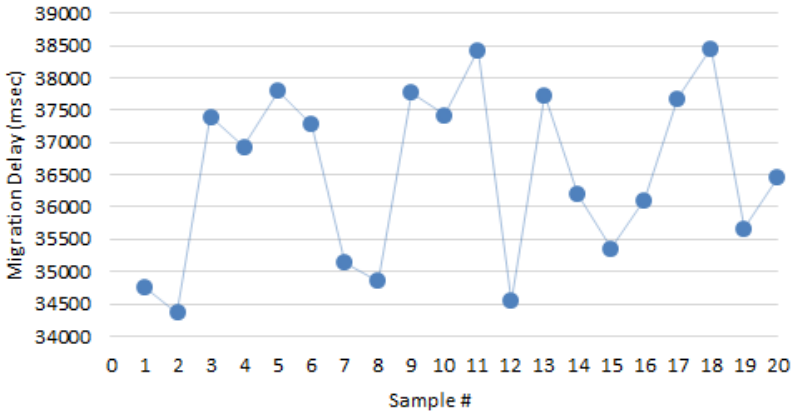
- *Service Provisioning Time* – the time between the moment the VM instantiation starts in VWSAN Gateway Provider Domain and when the VMs are migrated to VWSAN Provider Domain, including the chaining time of VMs, while also calculating the downtime duration of the VMs.
- *End-to-End (E2E) Delay* - time between the moment sensors send a measurement and when robots are deployed. We calculated E2E delay for non-virtualized and virtualized environments.
- *Scalability* – the ability of the system to handle the growing amount of loads without suffering significant degradation in the performance. We considered the response time of the system as a metric to evaluate the scalability of our architecture. Response time is the time period from when measurements are sent by the sensors to when these measurements are received by the VNFs.

B) Results and Discussion

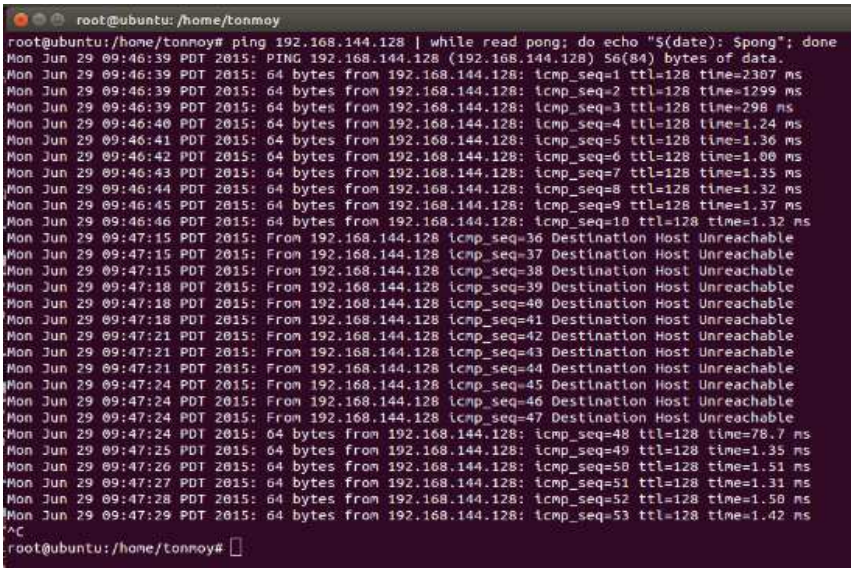
Test Case 1: Service Provisioning Time

Figure 4.4 (a) depicts the live migration delay of chained VMs, based on shared storage in a virtualized environment. We studied 20 tests and found a maximum delay of 38.4 sec and a minimum delay of 34.3 sec. We observed that the delay fluctuates between samples. This is because the time needed to instantiate VMs and migrate them in OpenStack is inconsistent. One of the limitations of OpenStack is the time needed to start a new VM, which could cause a prolonged delay in service provisioning time. As reported in [93], VM instantiation delay can sometimes reach up to 60 sec.

Although the live migration of VMs allows transferring VMs to other physical servers without shutdown and ensures high availability with non-stop services, VMs still face some period of downtown, depending on the memory state of the VM. In this experiment, we tested ping on the VMs during live migration. We started pinging before the live migration starts and it lasted until the migration ends. We noticed that during the migration, some ping requests were lost. Figure 4.4 (b) shows the process of pinging the VMs and Figure 4.4 (c) shows the downtime of the VMs considering 5 samples.



(a)



(b)

Sample	Protocol Conversion Downtime (sec)	Info Model Processor Downtime (sec)
1	30	39
2	24	39
3	31	38
4	33	38
5	24	38

(c)

Figure 4.4 Results of service provisioning

(a) Live migration delay of VM
(b) Pinging the VM during live migration
(c) VM downtime during live migration

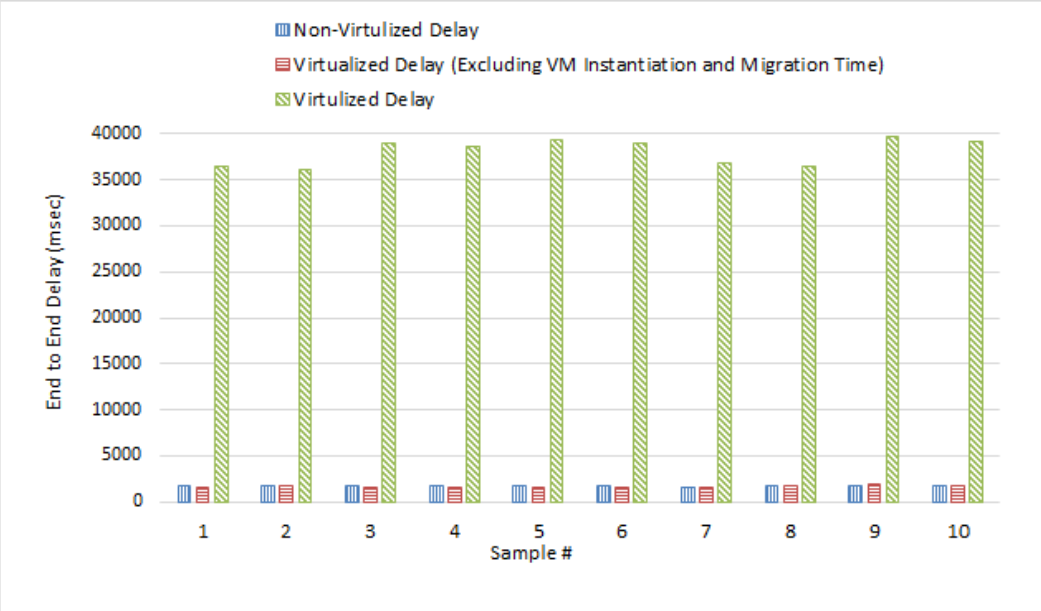


Figure 4.5 End to end delay (virtualized gateway vs. non-virtualized gateway)

Test Case 2: E2E Delay

Figure 4.5 illustrates a comparison of E2E delay for virtualized and non-virtualized environments, wherein each sample represents the average E2E delay for the first 10 measurements. In order to ensure an accurate comparison, we repeated the experiment 10 times. The average E2E delay for a virtualized environment is always higher than the delay for the non-virtualized environment. This is because the delay includes the time needed to instantiate and migrate the VMs. The maximum E2E delay for the virtualized gateway is around 39736 msec (sample 9), whereas it is 1784 msec (sample 10) for the non-virtualized gateway. Frameworks such as [94] can be integrated with OpenStack to overcome the performance gap between virtualized and non-virtualized environments.

We observe that the minimum E2E delay of the virtualized gateway, excluding VM instantiation and migration delay, (1603 msec) is close to the minimum E2E delay of the non-virtualized gateway (1601 msec). Thus, we can conclude that the time needed to instantiate and migrate the chained VMs has a significant impact on the E2E delay of the virtualized gateway, which demonstrates the overhead of virtualization. However, E2E delay in a virtualized environment increases only when a new brand of sensor joins and sends requests to dispatch VMs to VWSAN Provider Domain.

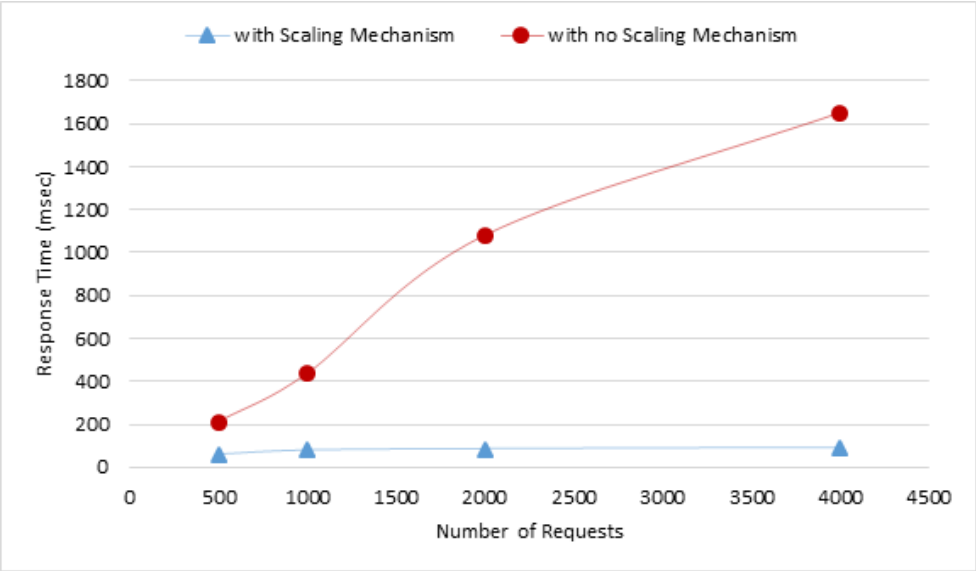


Figure 4.6 Response time for scalability

Test Case 3: Scalability

We used a simplified resource allocation mechanism to test the scalability. It is based on the resource utilization of the VMs (i.e., CPU) and on horizontal scaling. Each T period of time, VM's resources are monitored. If the utilization of the resource exceeds the threshold (i.e., 70%), we perform horizontal scaling. To conduct our case study, we set the number of requests as a variable within a unit time (T) and gradually increase it from 500 to 4000 requests. We considered 10 sec as the unit time. We used Apache JMeter to generate the requests using a uniform distribution of threads.

Figure 4.6 shows the results of our experiments, where we compared it with the same scenario without having a scaling mechanism. In the case of having a scaling mechanism, we notice that as the load (i.e., number of requests) increases, the system experiences a very slight increase in response time. This is because scaling is triggered before the system enters the overload state. For the initial increase in load (i.e., from 500 to 1000), the effect on response time is slightly more than the one afterward. This is because initially as load increases, more resources cannot be allocated until the T period is elapsed. From load 1000 till the maximum load, the response time increases by only 5 msec for every 2-fold increase in load. In contrast, if no scaling is performed, the system suffers from a significant increase in response time, as indicated in the figure. We observe that

from load 1000 till the maximum load, the response time increases by 600 msec for every 2-fold increase in load. Overall, with a scaling mechanism, the load has a very negligible impact on the response time. This demonstrates the scalability of our architecture.

4.4. Conclusions

In this chapter, we introduced an NFV architecture that deploys virtualized instances of a VWSAN gateway in an NFV infrastructure. The virtualized instances are dynamically migrated from a Gateway Provider Domain to several VWSAN Domains. With NFV, it is possible to achieve scalable deployment of gateways in heterogeneous VWSAN environments. In addition, several business actors involved in the proposed NFV architecture creates potentials for unique business models. We also discussed a proof-of-concept of the NFV-based virtualized gateway. We evaluated the prototype by conducting a set of experiments. The performance comparison of virtualized and non-virtualized approaches is analyzed, and the scalability of the architecture is proved.

Chapter 5

5. NFV and SDN - based Distributed Architecture for IoT Gateway

5.1. Introduction

Different types of IoT devices are used in large-scale disaster management applications. For instance, sensors, such as temperature, humidity, microwave, or infrared, may be distributed throughout a forest to monitor the environmental conditions or measure earth movements before and during earthquakes, and robots wide range of capabilities can be used in search and rescue missions. For example, some can penetrate rubble piles and find people beneath them, while others may be equipped with infrared cameras that transmit images back to the application. These IoT devices are usually heterogeneous, each with its own communication protocol and/or data formats.

As mentioned in the previous chapter (i.e., Chapter 4), to address the heterogeneity of IoT devices and their applications, gateways are needed to bridge the traditional communication networks and the IoT devices domain. Provisioning IoT gateways in large-scale disaster scenarios poses many challenges. For instance, traditional gateways lack dynamicity and flexibility. In addition, they are generally centralized and thus not practically feasible in the MANET settings of

large-scale disasters. Moreover, it is difficult and expensive to upgrade or reuse them. NFV [95] and SDN [23] can assist in overcoming these challenges.

The major contribution of this chapter is as follows: a distributed architecture for an IoT gateway based on NFV and SDN is proposed. The proposed architecture considers co-locating the gateway functions with the IoT devices and reusing already deployed gateways. It also considers handling the traffic and chaining between the gateway functions dynamically. A high-level description of the proposed architecture that is composed of two planes is provided, and a detailed description of each plane with its corresponding interfaces and procedures is presented. The proposed architecture is implemented as a proof of concepts in order to evaluate its viability and performance level.

The rest of this chapter is organized as follows, we first describe the proposed architecture, including its modules, interfaces, and procedures. After that, we present the implementation and investigates the performance evaluation. Finally, we conclude in the last subsection.

5.2. Overall Architecture for a Distributed IoT Gateway

In this section, we first present the proposed business model. The architectural principles are then introduced, followed by a high-level description of the proposed distributed IoT gateway architecture. A detailed description of the control and forwarding planes, including the related architectural modules, interfaces, and procedures is discussed next. This section ends with the presentation of illustrative sequence diagrams.

5.2.1. Business Model

In this chapter, we reuse the business model discussed in the previous chapter (i.e., Chapter 4). The specific actors and their relations are schematized in Figure 5.1. The End-User Applications are the applications that require the services of an IoT Provider (Figure 5.1, action 1). The IoT

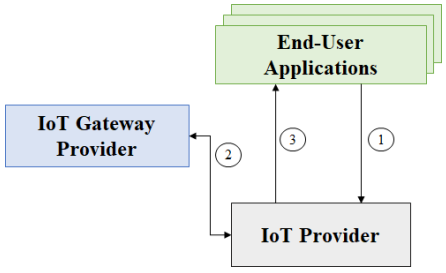


Figure 5.1 The proposed business model

Gateway Provider provides on-the-fly VNFs, representing the gateway functions, to handle the heterogeneity of the applications and the IoT devices (action 2). The IoT Provider provides the IoT devices (i.e., sensors and the robots) required by the application to realize its service. A business agreement between the IoT Gateway Provider and the IoT Provider is assumed that allows the IoT Gateway Provider to manage the infrastructure on which the VNFs are deployed and executed.

It should be noted that in this work it is assumed that there is a communication infrastructure that enables communication between the disaster site and disaster-safe areas, composed of IoT MANETs on the disaster site, and satellite mobile networks and cellular mobile networks to communicate with disaster-safe areas, as in reference [96]. Accordingly, the IoT provider can have a fixed node that can be carried on the relief vehicles during a MANET setup. The fixed node discovers and connects with the working cellular Base Station (BS). In fact, in practical situations, the cellular BS may be down at the central area on the disaster site. With luck, some BSs at a few distances away may be still working, allowing remote communication and information transfer between the IoT Provider in the disaster site area and the IoT Gateway Provider in the disaster-safe areas.

5.2.2. Architectural Principles

1- The first architectural principle is that the application and the gateway are built as a P2P overlay, due to the MANET setting of the disaster scenario.

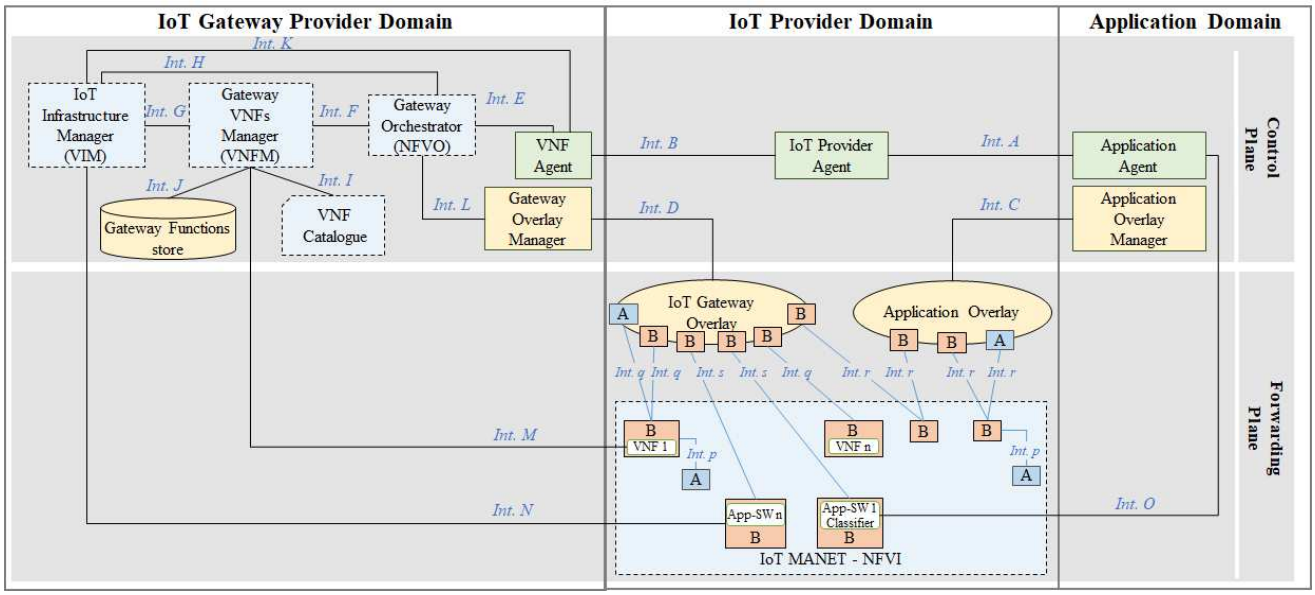


Figure 5.2 The proposed NFV and SDN-based distributed IoT Gateway architecture

2- The second principle is that two types of IoT devices are considered in the infrastructure, constrained (type A) and capable (type B). Constrained devices delegate some operations to the capable devices and capable devices can act on behalf of the constrained devices, as assumed in other works [97].

3- The third principle is that the IoT gateway functions are implemented as VNFs, and SDN is used to dynamically provision the paths between them once they are deployed.

5.2.3. High-level Description of the Architecture

The overall view of the proposed architecture is depicted in Figure 5.2. It includes three domains: the Application Domain, the IoT Provider Domain, and the IoT Gateway Provider Domain. Each functionality of the IoT gateway is implemented as a VNF. It is important to note that the architecture, specifically the portion in the IoT Gateway Provider Domain, is aligned with the ETSI NFV MANO framework [98]. In fact, it extends the MANO framework by adding new architectural modules.

The proposed architecture is layered over two planes: control and forwarding. The control plane handles the signaling procedure between the different domains, including the chaining of the gateway's VNFs. When an application requires an IoT service from the IoT Provider Domain (i.e.,

to receive sensor measurements and then deploy robots accordingly), a signaling procedure is conducted first to negotiate the service and exchange the necessary information. Different domains (i.e., the Application Domain, the IoT Provider Domain, and the IoT Gateway Provider Domain) are engaged in this service negotiation to obtain and deploy the appropriate gateway. The control plane also interacts with the forwarding plane by programming the application-level SDN switches to deliver the requested gateway to the application. Unlike traditional SDN switches, which are IP-level, the SDN Switches in the proposed architecture are application-level. This means the classification of a received packet should be done based on the header values of the application layer, similar to how it is done in [99].

The forwarding plane allows the flow of data through the path according to the control plane logic. It should be noted that all the interfaces of the proposed architecture are designed according to the Representation State Transfer (REST) principle.

5.2.4. Detailed Description of the Control Plane

Here the architectural modules involved in the signaling procedure along with their interfaces are presented, and some of the main procedures are described. It should be noted that some of the modules are reused from the ETSI NFV MANO framework, including the *NFVO* (i.e., *Gateway Orchestrator*), *VNFM* (i.e., *Gateway VNFs Manager*), *VIM* (i.e., *IoT Infrastructure Manager*), and the *VNF Catalogue*. Those modules are depicted with dashed lines in Figure 5.2. The remaining modules are newly-introduced ones.

A) Architectural Modules

1) Modules in the Application Domain

The *Application Agent* is in charge of the signaling procedure. It negotiates the use of IoT infrastructure. The *Application Overlay Manager* is responsible for the *Application Overlay* configuration, activation, and execution.

2) Modules in the IoT Provider Domain

The *IoT Provider Agent* is responsible for requesting the desired gateway from the IoT Gateway Provider Domain.

3) Modules in the IoT Gateway Provider Domain

The *VNF Agent* is responsible for analyzing the requested gateway's features, decomposing the request into a set of gateway VNFs that represent the gateway's functionalities, and requesting the execution of required orchestration plans. The SDN application resides in this module. It defines a set of chains which specify how the gateway VNFs are composed to fulfill the application's need(s).

As stated above, the proposed architecture reuses the *MANO* framework, including *NFVO*, *VNFM*, and *VIM*. The *NFVO* functionality is provided by the *Gateway Orchestrator*, which is in charge of orchestrating the *NFVI* resources and managing the lifecycle of the network services (i.e., the composition of VNFs). The *VNFM* functionality is provided by the *Gateway VNFs Manager*. This manager is responsible for the gateway's VNFs lifecycle management, including instantiation, maintenance, and termination. The *VIM* functionality is provided by the *IoT Infrastructure Manager*, which is responsible for resource allocations for the deployment and execution of the VNFs. The *SDN Controller* is co-located within the VIM. This is in accordance with ETSI SDN Usage in an NFV Architectural Framework [100]. According to this reference, one architectural option for the possible location of *SDN Controller* is to co-locate it with the VIM. This option is adopted in the proposed architecture. The *SDN Controller* establishes the path between the *VNFs* through which the sensor data and the commands to the robots traverse.

The *SDN Controller* is logically centralized. However, we envision it as being physically distributed to back-up controllers as suggested by references [101] [102] in order to incorporate fault tolerance mechanisms such as the ones described in these references. When it comes to scalability, we envision it as running in a virtualized environment in order to enable vertical and horizontal scalability. The reader should note that several existing SDN controllers (e.g., Floodlight [103]) do run in virtualized environments. The *SDN Controller* is connected to all the *Application-level SDN Switches* in the Forwarding Plane and programs them using an extended OpenFlow interface. The details of this interface are presented in the next subsection.

The *Gateway Functions Store* includes the list of *VNFs* (i.e., gateway functions) that the IoT Gateway Provider Domain can provide. It is similar to the Network Function Store proposed by T-NOVA [104] which contains the VNFs provided by several third-party developers, published as

independent entities and accompanied by their metadata. The *VNF Catalogue* represents a repository of all the on-boarded gateway VNF packages which are updated during their lifecycle management operation. The *Gateway Overlay Manager* is responsible for the *IoT Gateway Overlay* configuration, activation, and execution.

B) Interfaces

The general principles used to design the interfaces are presented first. This is followed by the description of the individual interfaces.

1) General Design Principles for the Interfaces

As the SDN switches presented in this chapter are application-level switches we first present the extensions made to the SDN interfaces. The general principles for the design of the other interfaces are presented after that.

SDN interfaces

In our proposed architecture, *Int. N* is the southbound interface and *Int. K* the northbound interface. OpenFlow and ForCES are the two standards currently used at the southbound interface [105]. However, they cannot convey application-level information. In this work, we have extended the most widely used standard (i.e., OpenFlow) as previously suggested in the literature [99][106]. An example of an extension is the addition of application-level fields to the match fields of the flow entries. The match field "OFB_IPV4_SRC" for instance is extended to support "source" "OFB_Application_Level_Address_SRC" (in addition to "OFB_IPV4_SRC"). Yet another example, is the extension of the action "output" to output a message to an application-level address.

Contrary to the southbound API, there is currently no standard for the northbound interface [105]. Most SDN controllers (e.g., OpenDaylight, Floodlight, etc.) do offer their own REST-based northbound APIs. However, like the southbound interface, the existing northbound interfaces do not cater to application-level features. If we take Floodlight, for instance, the resource "static flow pusher" REST API cannot be used as it stands to install flow table entries via REST API. We have therefore made extensions. An example is the extension of the matching field "ipv4_src" to include "application_level_address_src". Yet another example is the extension of "ipv4_dst" to include "application _level_address_dst".

Table 5.1 Example of the API operations exposed by the Gateway Orchestrator to the VNF Agent (i.e., *Int E*.)

REST Resource	Operation	HTTP Action and Resource URI
Orchestration Plan	Execute an orchestration plan	POST: /OrchestrationPlan
	Retrieve a specific orchestration plan	GET: /OrchestrationPlan/{Id}
	Retrieve all orchestration plan	GET:/OrchestrationPlan/all
	Remove an orchestration plan	DELTE: / OrchestrationPlan/{Id}
	Update an orchestration plan	PUT: / OrchestrationPlan/{Id}

Other Interfaces

The other interfaces have been designed from scratch using the well-known RESTful Web services principles [107]. They all expose CRUD (i.e., Create, Read, Update, and Delete) operations. Table 5.1 summarizes the proposed REST interface (i.e., *Int. E*) for the interactions between the *Gateway Orchestrator* module and the *VNF Agent* module. It defines resources on the *Gateway Orchestrator* "Orchestration Plan". This "Orchestration Plan" resource exposes a subset of the uniform interface to the *VNF Agent*. When the latter sends a POST request to the *Gateway Orchestrator* to request the execution of a new orchestration plan, the *Gateway Orchestrator* creates a new "Orchestration Plan" resource and sends the resource URI to the *VNF Agent*. This URI is used to modify or get the status of an existing orchestration plan and to remove the resources of a specific orchestration plan.

2) Individual Interfaces Description

Int. A is used by the Application Domain to request IoT service from the IoT Provider Domain. *Int. B* allows the IoT Provider Domain to request the desired gateway from the IoT Gateway Provider Domain. *Int. C* and *Int. D* are used for *Application Overlay* creation and *IoT Gateway Overlay* creation, respectively. *Int. E* is used by the *VNF Agent* to request the *NFVO* to execute the orchestration plan (deploy required *VNFs*, chain them, and create the gateway overlay).

Int. F, *Int. G*, and *Int. H* are reused from the ETSI NFV MANO framework; they represent Or-Vnfm, Vi-Vnfm, and Or-Vi, respectively. *Int. F* enables the instantiation, maintenance, and termination of the gateway's VNFs. *Int. G* enables the *NFVI* resource allocation for the gateway VNFs. *Int. H* enables the *NFVO* to monitor the *NFVI* resources. *Int. I* is a reference point in an ETSI NFV framework. It allows the *VNFM* to verify if the requested VNF is already deployed. *Int. J* is used to fetch the required VNFs from the *Gateway Function Store*. *Int. K* is the northbound

interface of the *SDN Controller*, named Application Control Interface in ETSI SDN Usage in NFV [100]. *Int. L* is used by the *Gateway Orchestrator* to instruct the *Gateway Overlay Manager* to create the *IoT Gateway Overlay*.

Int. M and *Int. N* are reused from the ETSI NFV MANO framework; in MANO terminology these are Ve-Vnfm-vnf and Nf-Vi, respectively [22]. *Int. M* is for the lifecycle management of the *VNFs*, and *Int. N* represents the southbound API of the *SDN Controller*, named the SDN Resource Control Interface in ETSI SDN Usage in NFV [100]. *Int. O*, located between the *Application Agent* and the flow classifier, is used to redirect the application request to the flow classifier.

C) Procedures

The proposed architecture, as part of the signaling procedure, includes the following two procedures: IoT gateway provisioning and application provisioning. IoT gateway provisioning includes the IoT gateway request and the IoT gateway orchestration. IoT gateway orchestration refers to the IoT gateway deployment, IoT gateway chaining, and IoT gateway overlay creation. For the application provisioning procedure, this chapter focuses on the application overlay creation phase.

Next, the gateway orchestration procedure (Figure 5.3 (a)) with its three phases: deployment, chaining, and overlay creation, is described.

1) IoT Gateway Deployment

The process starts when the IoT Gateway Provider Domain receives a gateway request from the IoT Provider domain (through Int. B). The *VNF Agent* instructs the *Gateway Orchestrator* to execute the orchestration plan through Int. E. The *Gateway VNFs Manager* first checks if the *VNFs* needed for the requested gateway are already present in the IoT Provider Domain. If not, the *Gateway Orchestrator* discovers the capable IoT devices in the IoT Provider Domain (i.e., type B) along with their capabilities and features, such as energy level, response time, location, etc., and maintains a clear view of the network topology. The *Gateway VNFs Manager* then finds the requested *VNFs* in the *Gateway Functions Store* (through Int. J) and instantiates and dispatches them (through Int. M).

2) IoT Gateway Chaining

Once the *VNFs* are deployed, the *SDN Controller* (co-located with the *IoT Infrastructure Manager)* programs the *Application-level SDN Switches* (via Int. N). The *SDN Controller* populates a set of application-level flow entries in the *Application-level Switches* based on the chains defined by the SDN Application in the *VNF Agent*. The SDN application injects these entries in the *SDN Controller* via Int. K. According to these entries, a path is established between the application and the IoT devices through which the data from IoT devices traverse to the application.

3) IoT Gateway Overlay Creation

This process is initiated when the *Gateway Overlay Manager* receives a request from the *Gateway Orchestrator* to create the *IoT Gateway Overlay* (through Int. L). The *Gateway Overlay Manager* first configures the *IoT Gateway Overlay* between the selected type B devices. It then activates the overlay, where the selected IoT devices receive an overlay join request (Int. D).

The proposed architecture in Figure 5.2 includes two overlays built on top of the IoT MANET: the *IoT Gateway Overlay* and the *Application Overlay*. They co-exist simultaneously; each may have its own overlay protocol for message exchange. They share nodes and underlying network links. In order to allow these overlays to interact and cooperate with each other, the proposed approach uses co-located nodes, (nodes that belong to the two overlays) to enable inter-overlay routing and reduce traffic [108]. Every message received by a co-located node can be forwarded to the other overlay the node belongs to. Using super-peers is another approach; however, it leads to costly merging mechanisms [109] and [110].

5.2.5. Detailed Description of the Forwarding Plane

The architectural modules and the interface of this plane are described below. The *NFVI* is reused from the ETSI NFV architectural framework [22] and is shown with dashed lines in Figure 5.2.

A) Architectural Modules

All the modules described in this section are in the IoT Provider Domain. The *NFVI* is able to host the *VNFs* deployed over IoT devices. The *VNFs* are the software instances of the gateway functions.

Type A IoT devices rely on type B devices to join the overlay. They send data to the *IoT Gateway Overlay* through type B devices. Type B devices can execute one or more function of the gateway, and they can represent themselves and/or a type A device in the *IoT Gateway Overlay*. Similarly, type B devices can join the *Application Overlay* on behalf of type A devices. In addition, the same type B device may belong to both overlays.

The A*pplication-level SDN Switches* are programmable by the *SDN Controller*. The *Application-level SDN Switches* in the proposed architecture are placed on a computing hardware in the *NFVI* (as one of the options presented in [100]) in which some type B devices act as SDN switches. It is worth noting that some industrial projects are working on using application-layer intelligence in an SDN environment (i.e., a white paper [111]).

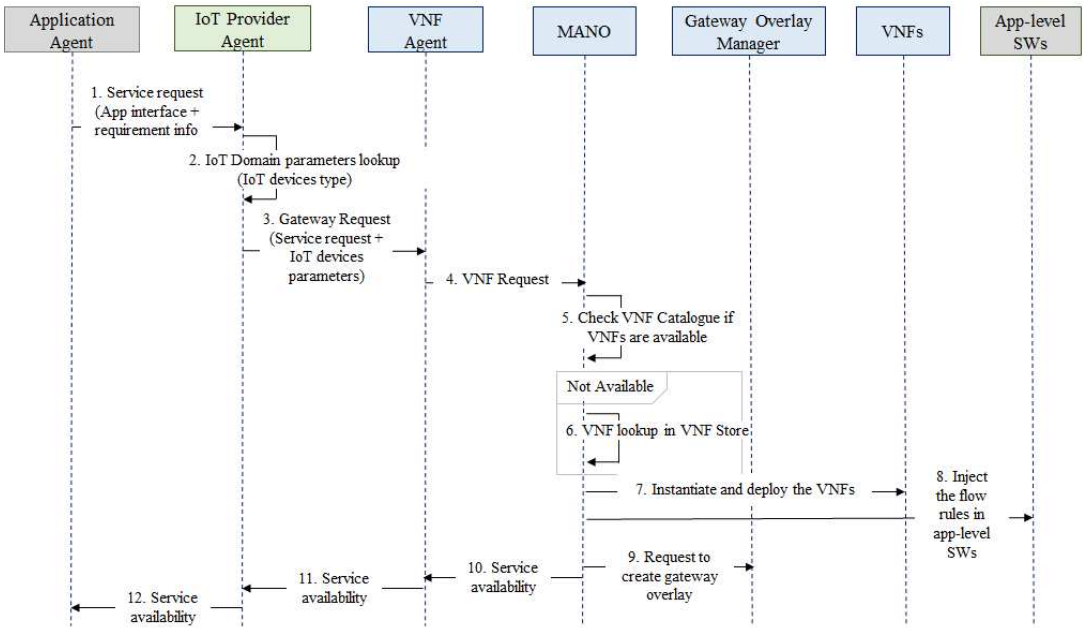
B) Interfaces

Int. p is used to exchange control data between type A and type B IoT devices. *Int. q* is used to send the sensor data to the *IoT Gateway Overlay*. *Int. r* is used to send the data received from the IoT devices executing an application's task to the *Application Overlay*. *Int. s* allows communication between the *Application-level SDN Switches* in order to establish the path between them. It also allows pushing/retrieving of the sensor data between the *Application-level SDN Switches* and the *VNFs*.

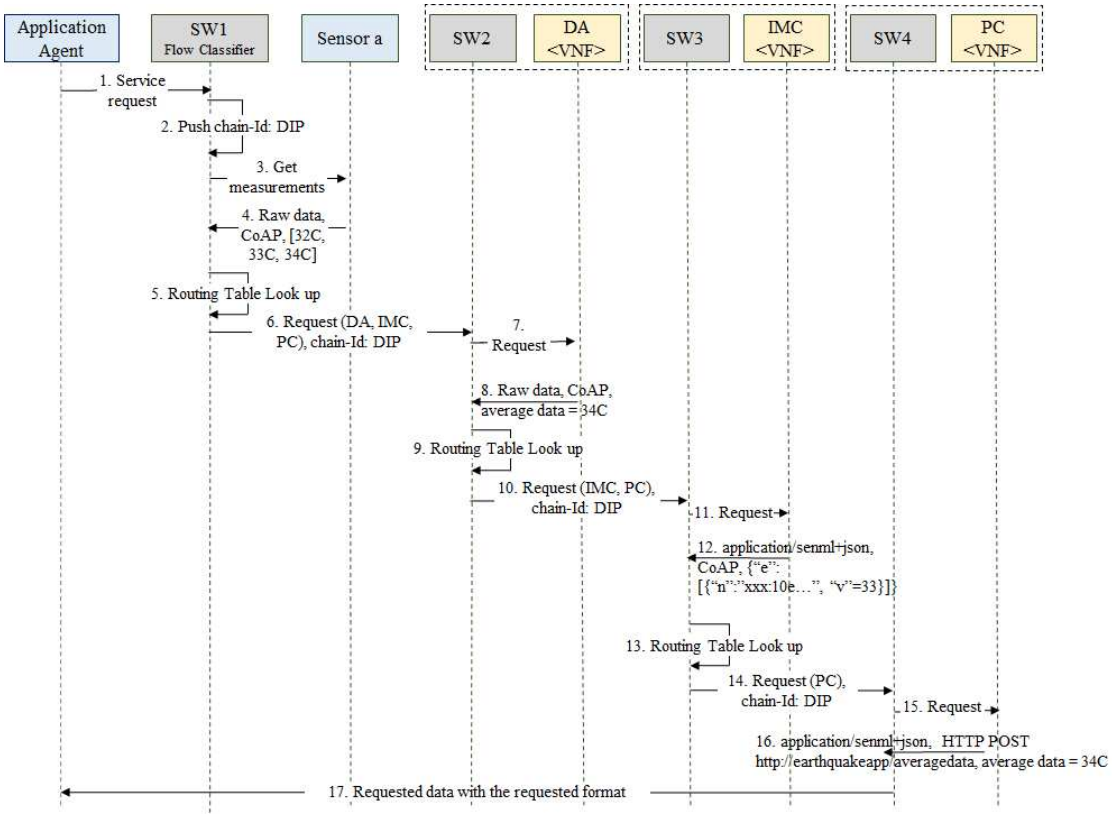
5.2.6. Illustrative Sequence Diagrams

Figure 5.3 illustrates a sequence diagram of the interactions of different architectural modules during the signaling procedure for gateway provisioning (gateway request and gateway orchestration). It also illustrates an end-to-end scenario of the actual flow of data when sending data from the sensors to the application. A fire detection and fighting application is considered, which wants to collect data from temperature sensors to detect prospective fires after an earthquake and can then deploy firefighting robots in case of a disaster.

The gateway provisioning procedure is shown in Figure 5.3 (a). MANO in the figure represents the *Gateway Orchestrator* (i.e., NFVO), the *Gateway VNFs Manager* (i.e., VNFM), and the *IoT Infrastructure Manager* (i.e., VIM). During the deployment phase, the *VNFs* are selected in the IoT Gateway Provider Domain. This selection is done based on the application's interface description and requirements (Figure 5.3 (a) action 1), e.g., SenML over HTTP, average data and



(a)



(b)

Figure 5.3 Sequence diagram for the Gateway Deployment and Chaining

(a) IoT Gateway provisioning procedure
(b) End-to-End scenario of the actual flow of data

Table 5.2 Application-level flow tables

App-level Switch	Match Field	Action
SW1	Application: HTTP, SenML, Average data && IoT: Raw, CoAP	Insert Chain Id A Forward to SW2
SW2	Chain Id = A	Forward to SW3
SW3	Chain Id = A	Forward to SW4
SW4	Chain Id = A	Forward to "sensor a"

IoT device specifications (action 2-3), e.g., Virtenio Sensors with CoAP protocol that send raw data. The IoT devices' specifications are found in a repository held by the IoT Provider Domain.

It is assumed that the requested Gateway's VNFs implement the following functions: (1) Data Aggregator (DA) to send data over a specific threshold; (2) Protocol Converter (PC) to convert the data received from IoT devices to the appropriate model supported by the application and vice versa; and (3) Information Model Converter (IMC) to convert a model from one to another. Similar to the previous chapter (i.e., Chapter 4), we do acknowledge the fact that converting a protocol X (or an information model X) into a protocol Y (or an information model Y) is not always feasible. Consequently, the IoT Gateway Provider provisions the related VNFs only when the conversion is feasible. It is also important to note that when the required *VNFs* are not found in the *Gateway Functions Store*, a service unavailability notification is sent to the *Application Agent*, to either cancel the negotiation or resume signaling after a certain time.

The selected VNFs are instantiated and deployed in the IoT Provider Domain (action 8). The *IoT Infrastructure Manager* next injects the flow entries in the *Application-level SDN Switches* (action 9). Some examples of such entries are listed in Table 5.2. The *IoT Gateway Overlay* is then created in the IoT Provider Domain (action 10). Finally, the *Application Agent* receives a notification about service availability through the *VNF Agent* and the *IoT Provider Agent* (actions 11-13) in which the *Application Agent* is instructed to contact the flow classifier to collect data from the sensors. Table 5.3 demonstrates two prospective chains that could be defined for these VNFs based on the end-user application preference and the IoT devices' properties.

The end-to-end scenario of the forwarding plane is shown in Figure 5.3(b). The *Application Agent* sends its request to the SW1 (action 1) which performs the flow classification according to Table 5.2. It then pushes the chain-Id on the request (action 2). This chain-Id indicates that DA,

Table 5.3 Prospective chains based on application-level requirements

End-user App. Requirements			IoT Devices Properties		VNF Chains	Chain ID
Protocol	**Info Model**	**Data Aggregation**	**Protocol**	**Info Model**		
HTTP	SenML	Average Data	CoAP	Raw Data	DA1, IMC1, PC1	A
HTTP	SensorML	Average Data	HTTP	Raw Data	DA1, IMC2	B

IMC, and PC are needed to deliver the required service to the application. SW1, according to the entries in its routing table, (i.e., Table 5.2) collects measurements from sensors (e.g., sensor x, type A) and sends the request to the second switch (SW2) (action 3-6). The SW2 sends the request to the DA VNF to aggregate the data (actions 7-8). The same applies to SW3 and SW4 (actions 9-16). Finally, the requested data is sent back to the *Application Agent* (action 17) through the *Application Overlay*.

5.3. Performance Evaluation

For the prototype, the recovery phase of the earthquake early warning and recovery application (presented in Chapter 2) is implemented. This phase is as follows; an earthquake recovery application collects the data of sound sensors deployed in the affected areas. These sensors can detect voices or other sounds of possible human presence through the ruins and inform the earthquake recovery application. In order to communicate with these IoT devices, the application needs a gateway for handling the different types of communication interfaces. Sometime later, a fire detection and fighting application that needs to be notified when a fire occurs, adds temperature sensors that can detect fires. It then deploys a fleet of robots that can detect extinguishers and grab one in order to suppress the fire. Accordingly, the gateway needs to be upgraded in a dynamic manner such that it can serve the newly added IoT devices.

Three different types of IoT devices are used. Sunfounder[5] sensors are deployed over the Raspberry Pi[6] (RPi) to detect sounds, Virtenio[7] sensors are deployed to detect fires, and Lego Mindstorms[8] robots with specialized arms are utilized to grab extinguishers and suppress fires. A ball is used to simulate the extinguisher.

[5] sunfounder.com/modules/sensor-module.html/
[6] raspberrypi.org/
[7] virtenio.com/
[8] lego.com/en-us/Mindstorms/

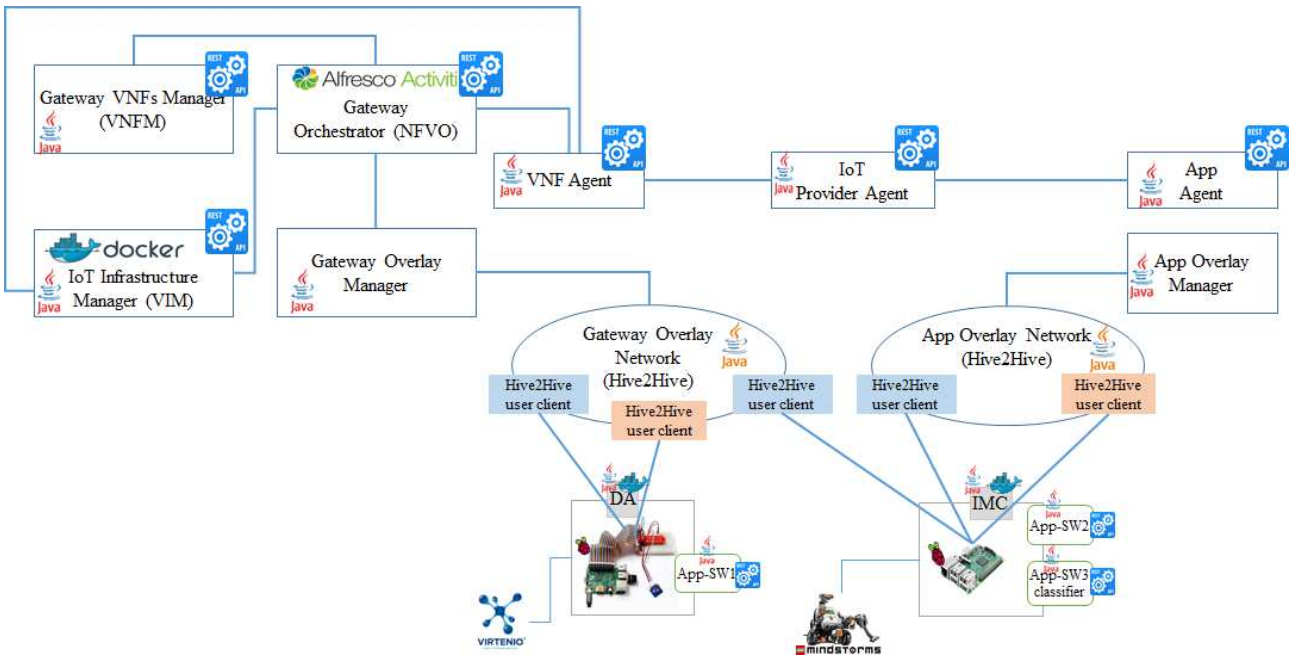


Figure 5.4 Prototype architecture

In these experiments, we assume that the recovery phase of the earthquake early warning and recovery application has one application component: Victim Detector. To that end, Sunfounder sensors on RPis send their raw measurements over HTTP Protocol. This data is first processed by a DA (to send only the sounds of any possible human beings) and then processed by the IMC (mapped to SenML format). The fire detection and fighting application has two application components: Fire Detector and Robot Dispatcher. Virtenio sensors send their raw measurements over CoAP protocol. These measurements are processed by a DA (to send data over a specific threshold) followed by an IMC (mapped to SenML format), and finally by a PC (encoded in HTTP protocol). Accordingly, the same IMC can be reused by both applications. Finally, in order for the application to send commands to the robots, the IMC and the PC convert the HTTP request received into LeJOS JAVA API command that implements the LCP.

In this prototype, it is assumed that an ad-hoc network is already built and that the connection between the involved nodes has been established. It is also assumed that a fixed node in the IoT provider is carried on the relief vehicles during MANET setup and is part of the gateway and the application overlay.

Figure 5.5 IoT Gateway orchestration plan

5.3.1. Proof of Concept Prototype

The validation prototype is implemented according to the architecture depicted in Figure 5.4. The signaling procedure for gateway provisioning (gateway request and gateway orchestration) and the end-to-end scenario of the actual flow of data as described by the illustrative sequence diagrams (i.e., Section 5.2.6) are implemented. In this implementation, container-based gateway orchestration is adopted. Containers are lightweight, stand-alone, and modular. In addition, they allow rapid configuration and deployment. Existing open source MANO solutions are not used in this prototype as they do not support container-based orchestrations.

For the *Gateway Orchestrator* (i.e., NFVO), Alfresco Process Services[9], an enterprise Business Process Management (BPM) solution is used. It allows the creation of process definitions and orchestration plans using the capabilities of the Business Process Model Notation (BPMN). It also exposes REST API to external entities to execute the orchestration plan. Figure 5.5 shows an example of the orchestration plan used for the new deployment of the gateway. In this plan, the required VNFs for the gateway are first deployed, then the VNFs are chained using SDN, and finally, the *IoT Gateway Overlay* is created.

The application-level switches are implemented using Java libraries. These switches expose a REST API implemented as Java Restlet framework to the *SDN Controller* through the fixed node in the IoT Provider domain for handling application-level flow entries. For the SDN Controller, since existing open source SDN controllers do not support our proposed extended features, we implemented it as a simple REST API using the Restlet framework.

The *IoT Gateway Overlay* is created by the *Gateway Overlay Manager*. Hive2Hive[10] API is used, which provides a free, open-source, distributed, and scalable solution for distributed P2P

[9] alfresco.com/
[10] hive2hive.com/

networks. It also provides for headless deployment (e.g., on an RPi) and is configurable and customizable. The overlay is created by the fixed node in the IoT Provider domain as a master client by advertising its address and creating a user profile. The remaining nodes join the overlay network as user clients by registering to the user profile.

The VNFs are implemented using Java libraries and packaged in Docker containers. They are pushed to the DockerHub repository. The container images are pulled from the repository through the fixed node in the IoT Provider domain. The *Application-level SDN Switches* and the VNFs communicate with each other in the overlay using the Hive2Hive framework.

Two VNFs (i.e., DA and IMC) and three Application-level SDN Switches are randomly placed on the two RPis. It should be noted that a VNF placement algorithm can be adapted to deploy the VNFs in the optimal location in the network (e.g., [112]). The remaining architectural modules are modeled as RESTful web services using a Java Restlet framework.

5.3.2. Experimentation Setup

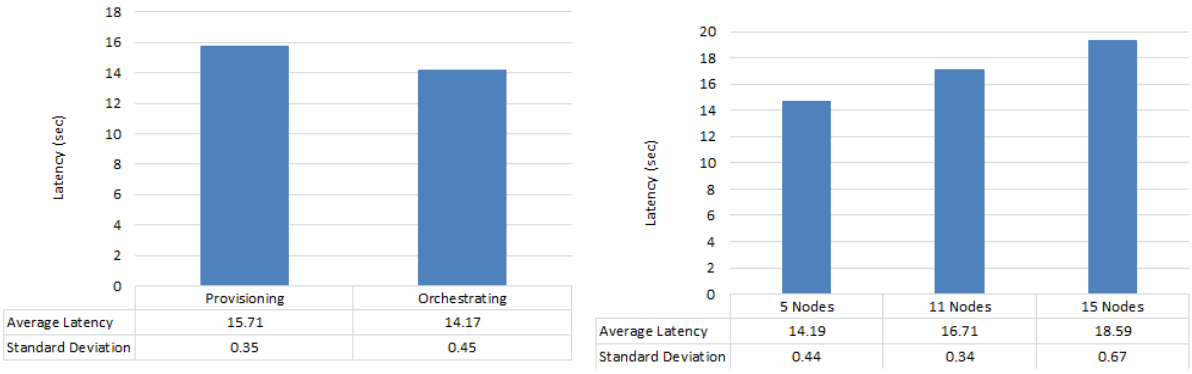
The IoT Gateway Provider domain runs on a 64-bit laptop with Ubuntu 14.04.5 LTS. The Application domain runs on a laptop with Intel® Xeon® CPU clocked at 2.67 GHz and a 6GB RAM with 64-bit Windows 7 Enterprise. The fixed node of the IoT Provider domain runs on a 64-bit laptop with Ubuntu 14.04.5 LTS. The RPis run a Raspbian OS which is based on Debian OS. The first RPi hosts two Hive2Hive user clients (i.e., one on behalf of a Virtenio sensor and one on behalf of the DA VNF). The second RPi hosts three Hive2Hive user clients (i.e., one on behalf of a LEGO Mindstorms robot, one on behalf of an IMC VNF, and one representing the Victim Detector component of the earthquake application.

5.3.3. Measurements and Results

A) Performance Metrics

The performance metrics utilized to evaluate the performance of the proposed architecture are:

- *Gateway provisioning latency* – measured from the time the *Application Agent* sends a request to obtain sensor measurements to the time the gateway is deployed and chained in the IoT domain. Experiments with a different number of VNFs, SDN switches, and overlay nodes are conducted.



(a) (b)

Figure 5.6 Gateway provisioning latency

(a) Provisioning latency vs. orchestration latency
(b) Provisioning latency with different number of VNFs and Application-level SDN Switches

- *Orchestration latency* – Measured from the time the orchestration request to the *Gateway Orchestrator* is initiated to the time the acknowledgment of orchestration is received. The orchestration process, as discussed in the previous section, includes the IoT gateway deployment, IoT gateway chaining, and IoT gateway overlay creation. In addition, the orchestration latency is measured when a request to upgrade an already-deployed gateway is received. To that end, a scenario where new sensors are added by the fire detection and fighting application is assumed. According to this scenario, a gateway with three VNFs is required (i.e., DA, IMC, PC), with one of the VNFs (i.e., IMC) already deployed in the IoT Provider Domain.

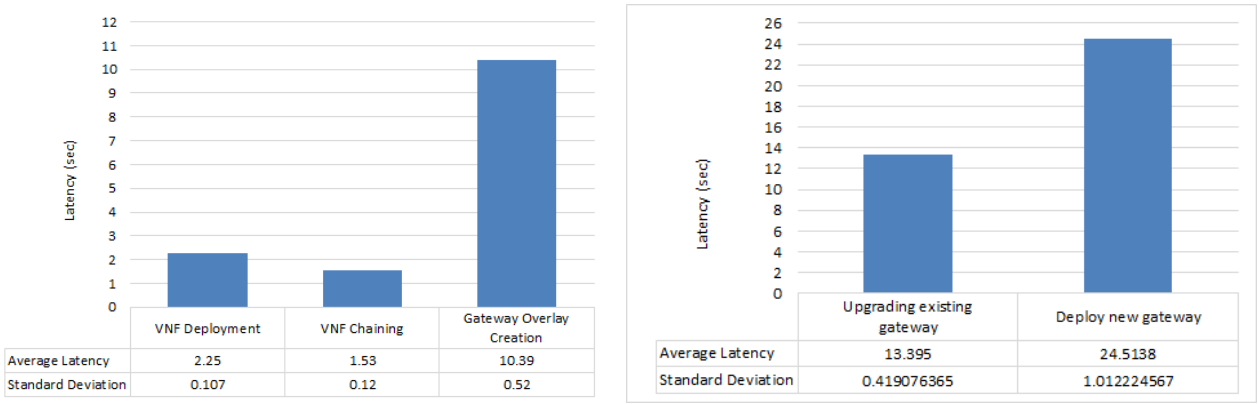
- *End to End (E2E) delay* - Measured from the time the IoT devices send their data to the time the requested data is obtained by the application. Here, the order of the VNFs is varied to show the effect of changing the order of the VNFs on the E2E delay. The E2E delays for both fixed/centralized gateways and the proposed distributed gateway are also calculated.

B) Results and Discussions

This section discusses the performance results obtained, beginning with the provisioning latency.

Test Case 1: Provisioning latency

Figure 5.6 (a) depicts the IoT gateway provisioning latency and the orchestration latency, which is a sub-phase of the provisioning procedure. Both average latency and the standard deviation are provided for 10 consecutive experiments. The average provisioning latency is 15.71



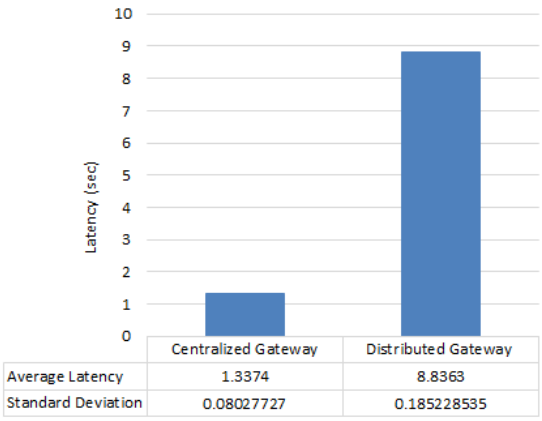
(a) (b)

Figure 5.7 Orchestration latency of the proposed gateway

(a) Orchestration latency of each phase of the orchestration plan
(b) Orchestration latency of upgrading the proposed gateway vs. orchestrating a new gateway

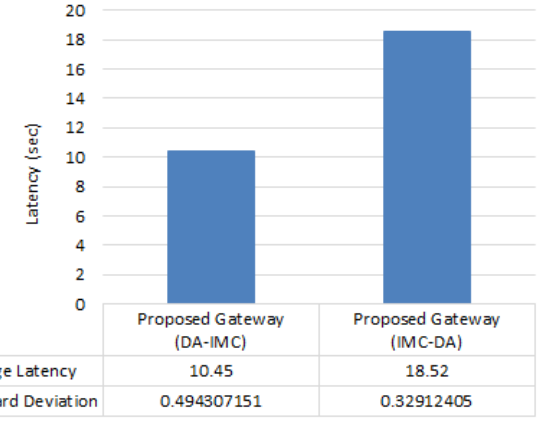
sec. This shows the efficiency of using NFV and SDN-based gateways compared to traditional gateways. The latter imposes lengthy deployment time, requires additional configurations for physical interfaces to implement chains, etc. [113]. Also, it can be observed from the graph that the average orchestration latency (14.17 sec) takes up most of the time of the provisioning procedure.

In order to conduct an accurate insight into how the system behaves depending on the number of VNFs, SDN switches, and overlay nodes, several cases are carried out. A linear topology is used for the Application-level SDN Switches. We gradually increase the number of instances of each VNF while considering a load balancer VNF for each group of VNFs of the same type to equally distribute the load among them. Therefore, the case with 6 VNFs for instance, include 3 instances of each VNF (i.e., Data Aggregator and Information Model Converter), 2 load balancers, and 7 application-level SDN switches. This leads to a gateway overlay with 15 nodes. A Similar number of overlay nodes for disaster management scenarios are considered in the literature, e.g., [114]. It can be observed in Figure 5.6 (b) that by increasing the number of VNFs and the SDN switches the provisioning latency behaves almost at a slight constant rate, linearly. This is because when the number of nodes in the overlay is increased, the system experiences a very slight increase in



	Centralized Gateway	Distributed Gateway
Average Latency	1.3374	8.8363
Standard Deviation	0.08027727	0.185228535

(a)



	Proposed Gateway (DA-IMC)	Proposed Gateway (IMC-DA)
Average Latency	10.45	18.52
Standard Deviation	0.494307151	0.32912405

(b)

Figure 5.8 E2E latency of the proposed gateway

(a) E2E delay for centralized gateway vs proposed distributed gateway
(b) E2E delay of the proposed gateway for different orders of VNFs

the overlay creation latency. This is since all the nodes (i.e., user clients in Hive2Hive terminology) need to register to the user profile created and advertised by the fixed node (i.e., master client) and join the overlay.

Next, the various phases of the orchestration are measured to get a more detailed insight into the orchestration latency.

Test Case 2: Orchestration latency

Figure 5.7 (a) shows the latency of the orchestration, including IoT gateway deployment, IoT gateway chaining, and IoT gateway overlay creation over 10 consecutive measurements. Both average latency and the standard deviation are provided. It can be noted that the overlay creation takes up most of the orchestration time, which indicates that overlay creation imposes an overhead on the overall management and orchestration procedures in terms of latency.

One reason for the time required for overlay creation is that with overlay networks, an additional intermediate level is added between the nodes in the physical network infrastructure. However, despite the overhead, the experimental results show that the overhead associated with the overlay network is not a significant factor compared with the considerable gain of the approach. Overlays play an important role in MANETs where nodes join and leave the group dynamically. The additional layer prevents interference with the existing protocols in the underlying heterogeneous environment of MANET.

Figure 5.7 (b) shows the orchestration latency associated with the procedure of upgrading the gateway vs deploying a new gateway. The proposed architecture allows the upgrading of a pre-deployed gateway for which the orchestration latency is 15.06 sec. In contrast, the orchestration latency associated with the procedure of deploying a completely a new gateway (when upgrading the gateway is not supported) is 24.51 sec. It can be observed that the proposed architecture's feature of allowing the gateway's upgrading decreases the orchestration latency by 38.56%.

Test Case 3: End to End delay

In Figure 5.8 (a), the E2E delay of the proposed distributed gateway with two VNFs (DA-IMC) is compared to a fixed centralized gateway. The fixed gateway aggregates the received data and converts the information model into two tightly coupled functionalities. The average latency for the proposed distributed gateway (i.e., 8.8 sec) is higher than the average latency for a centralized fixed gateway (i.e., 1.3 sec). This is basically because in a centralized gateway the communication latency between different functionalities is eliminated.

However, in the implementation, a linear topology is adopted for the SDN switches; other topologies could be investigated to see if they can reduce this latency. Eliminating the SDN switches and making the VNFs SDN enabled could also reduce this latency, but this might make the VNFs more demanding in terms of processing and storage and may not fit resource-constrained environments. Using other P2P frameworks could also be investigated to see if they can reduce this latency.

In Figure 5.8 (b), the E2E delay of the proposed distributed gateway with two VNFs (DA-IMC) is measured for two valid chaining options of the VNFs. Five consecutive measurements are assumed to be sent from the sound sensor, where the last measurement indicates a possible human being sound. In the first chain (i.e., DA-IMC), the DA receives all the measurements and sends only possible human being sound which is converted to the appropriate model at the IMC. In the second case (i.e., IMC-DA), all the received measurements from the sensors are converted to the model supported by the application (i.e., SenML) and then filtered at the DA to send only the possible human being sound. It should be noted that both cases lead to the same relevant result at the application. The average latency in the first chain (i.e., DA-IMC) is 10.45 sec, while in the second chain (i.e., IMC-DA) it is 18.52 sec.

It is clear that the performance of the gateway in terms of latency is directly affected by the order in which the VNFs are composed and processed. This order has a significant impact on the amount of processing capacity required for each VNF. And, the amount of the required capacity depends on the amount of data handled by that VNF instance. Different chain composition algorithms can be integrated to improve the performance of the gateway e.g., [112].

5.4. Conclusions

This chapter proposed an architecture for on-the-fly distributed gateway provisioning in disaster management using NFV and SDN technologies. These technologies make it possible to address the challenges of traditional gateways, such as dynamicity and flexibility. NFV allows upgrading the pre-existing gateway and deploying the gateway functions anywhere anytime, and SDN enables reusing the same gateway functions in different flows for different applications. The gateway functionalities were provisioned as VNFs and are chained dynamically using the application-level SDN switches. The IoT gateway was built as a P2P overlay taking into consideration the MANET settings of the disaster management scenarios.

A prototype of the proposed architecture was provided, and a set of experiments are conducted to evaluate the architecture. The results showed that building the IoT gateway as a P2P overlay imposes an overhead on the overall management and orchestration procedures, but it produces a considerable gain. The performances of distributed and centralized approaches are also analyzed and the effect of the order of the VNFs on the overall E2E delay is investigated. The results showed the advantages of using on-the-fly provisioning of IoT gateway and the possibility of reusing and updating a pre-existing gateway.

Chapter 6

6. A Coalition Formation Algorithm for Multi-Robot Task Allocation

6.1. Introduction

Disaster management application domains for events such as earthquakes need a very large number of robots (in the order of hundreds or even thousand [3]) in a single coalition in order to cover the whole disaster area and satisfy the requirements of the search and rescue tasks of the application. Hence, the selection of the best coalition in such real-world large-scale scenarios requires solving an optimization problem with the goal of optimizing several conflicting objectives simultaneously.

In this chapter, we propose an algorithm for coalition formation problem. We address the ST-MR-IA (Single-Task Multi-Robots Instantaneous-Assignment) class of Multi-Robot Task Allocation (MRTA) problems following the taxonomy presented in [115]. In the problem at hand, each robot is capable of executing one task at a time and each task needs to be assigned to a robot coalition. Also, the available information about the robots, the tasks, and the environment permits only instantaneous allocation of tasks to robots, without planning for future allocations [115]. Our proposed algorithm is based on Quantum Multi-Objective Particle Swarm Optimization (QMOPSO). QPSO is a discrete version of PSO to solve optimization problems with binary-valued

solution elements [116]. PSO is one of the many options for coalition formation. Simulated Annealing (SA), Genetic Algorithm (GA), and Column-Generation (CG) are other examples. PSO is chosen because of its effectiveness in solving a wide range of applications [117]. It has the ability to find optimal or near-optimal solutions for large-space problems in a short time compared to other heuristics [118].

The goal of the proposed algorithm is to ensure that the optimal coalition of robots is selected with the required capabilities for each task. The proposed algorithm consists of a filtering method, the QMOPSO approach, and a ranking method. Filtering is used to choose the best robots for the execution of the QMOPSO algorithm and to make the robots that have not been selected available for other requests. In addition, location constraints regarding the capability distribution of the robots are taken into consideration. For instance, some tasks require that the combination of a given sensor and actuator should reside on the same robot or on different robots. This is necessary to ensure proper execution of search and rescue task. The proposed algorithm is implemented in order to analyze its performance.

The rest of this chapter is organized as follows. First, it presents the problem formulation, followed by the description of the proposed algorithm. After that, it presents the simulation parameters and settings followed by the validation results. We will conclude this chapter at the end.

6.2. Problem Formulation

To consider this problem, let us define an infrastructure composed of n robots:

$$R = \{R_1, \ldots, R_i, \ldots, R_n\} \quad (1)$$

where n is significantly large and hence the infrastructure can support search and rescue task in large-scale disasters.

Each of these robots has two vectors of capabilities: sensing capabilities (e.g., cameras, sonars) and actuating capabilities (e.g., arms, wheels). It is assumed that each capability is a real non-negative value and indicates the number of sensors/actuators owned by the robots.

For robot R_i, the sensing and actuating capability vectors are:

$$S_{R_i} = \{s_1^i, \ldots, s_r^i\} \quad (2) \qquad A_{R_i} = \{a_1^i, \ldots, a_r^i\} \quad (3)$$

where r is the number of possible sensing and actuating capabilities.

A robot can be in three states: *Idle*, *Allocated*, and *Busy*. The idle state is when the robot does not perform any tasks, the allocated is when the robot is locked with the algorithm running on it, and the busy state is when the robot is performing a task.

The infrastructure can perform *m* tasks assigned to it:

$$T = \{T_1, \ldots, T_j, \ldots, T_m\} \quad (4)$$

Each task T_j is composed of p sub-tasks:

$$Z_{T_j} = \{z_1^j, \ldots, z_k^j, \ldots, z_p^j\} \quad (5)$$

It is assumed that the sub-tasks are executed independently and that each robot is a member of only one sub-task. Each sub-task requires a specific set of sensing and/or a set of actuating capabilities to start.

We represent the capability requirements of each sub-task z_k^j by two vectors, sensing requirements and actuating requirements, as:

For sub-task z_k^j, the capability requirement vectors are:

$$S_{z_k}^j = \left\{s_1^{j_k}, \ldots, s_r^{j_k}\right\} \quad (6)$$

$$A_{z_k}^j = \left\{a_1^{j_k}, \ldots, a_r^{j_k}\right\} \quad (7)$$

Then, the capability requirement vectors for the task T_j is the sum of the capability requirement vectors of the sub-tasks constituting the task T_j:

$$S_{T_j} = \sum_{k=1}^{p} S_{z_k}^j \quad (8)$$

$$A_{T_j} = \sum_{k=1}^{p} A_{z_k}^j \quad (9)$$

Some of the sub-tasks of task T_j are tied by locational constraints regarding the capability distribution of the robots while others may be executed without any locational constraints. According to [50], there are two types of locational constraints; a combination of sensors and actuators should reside on the same robots or on different robots. The locational constraints can be represented as Constraints Satisfaction Problem (CSP) [50].

CSP consists of three components:

1. The set of variables, that is the required sensor and actuators for the task

$$X = \{x_1, \ldots, x_j, \ldots, x_k\} \quad (10)$$

where $X = \{s_1, s_2, \ldots, s_n, a_1, a_2, \ldots, a_n\}$

2. The set of values for each variable, that is the available robots possessing the required capabilities for each variable

For variable x_i, the set of values is:

$$V_{x_i} = \{R_j, \ldots, R_n\} \quad (11)$$

3. The set of constraints between different collections of variables

$$C = \{C_1, C_2, .., C_n\} \quad (12)$$

where each C_i is one of the following types: $x_i \neq x_j, x_i = x_j$. The goal in CSP is to assign a value for each variable such that the constraints are satisfied.

A coalition CLT for any task has two vectors of capabilities: sensing S_c and actuating A_c while each is the sum of the capabilities owned by the robots in that coalition:

$$S_c = \sum_{R_i \in C} S_{R_i} \quad (13)$$

$$A_c = \sum_{R_i \in C} A_{R_i} \quad (14)$$

Coalition CLT^m can perform task T_j only if:

1. The vector of its capabilities satisfies the following:

$$S_c \geq S_{T_j} \quad \text{And/or} \quad A_c \geq A_{T_j} \quad (15)$$

2. And its members meet the locational constraints.

It is assumed that a coalition can work on a single task at a time and that each robot is a member of one coalition at a time.

$$C1_i \cap C2_i = \emptyset \quad (16)$$

The objective is to find a coalition that minimizes the deployment cost of the robots, minimizes the time needed to perform a task by the robots, and to minimize the number of robots in a coalition.

6.3. Coalition Formation Algorithm for Multi-Robot System

We propose an algorithm for coalition formation for Multi-Robot system. The Pseudocode for this algorithm is given in Algorithm 6.1. The set of inputs for the algorithm are n (the maximum number of robots allowed in a group), Time (the maximum time period to complete a given task), Cost (the cost the customer agrees on), Filtering_Rule, Task_Requirements (the required sensors and actuators for a given task), Locational Constraints (the capability distributions for sub-tasks constituting a given task), and Criteria_Importance (defining the weights to rank the Pareto-optimal solutions based on more than one criterion - i.e., objectives in our case).

The algorithm starts with filtering the robots based on the Filtering_Rule. In this function, if the battery level of the robots is lower than the Filtering_Rule, they are excluded from the next steps. It then applies the QMOPSO-based algorithm. Multi-objective problems generate a set of non-dominated

or Pareto-optimal solutions. The solutions are ranked after excluding the solutions that exceed the time, the cost, the number threshold, and the infeasible solutions. Promethee II ranking [119] is applied, which is a multi-criteria ranking method with lots of success due to its mathematical properties and its user-friendliness. In this method, the Pareto-optimal solutions are compared pairwise. The difference between the evaluations of two Pareto-optimal solutions over each criterion is considered. The criteria in our case are the objectives (i.e., time, cost, and number of robots). The Pareto-optimal solutions are ranked using the Criteria-Importance/weight of the objectives. The highest rank solution denotes the best robot coalition.

The QMOPSO algorithm first initializes the particles. The Pseudocode is given in Algorithm 6.2. A particle is defined based on the quantum bit. Two vectors are initialized:

- Quantum particle vector $V(t)^i$, which is the velocity for particle *i* and is initialized to random values between [0,1]:

$$V(t)^i = [v(t)^i_1,\ v(t)^i_2,\ \ldots, v(t)^i_n] \quad (17)$$

- Discrete particle vector $p(t)^i$, which is initialized by initializing a random number for each $v(t)^i_j$ and then, according to the condition in (19) and (20), the discrete particle vector is initialized:

$$p(t)^i = [p(t)^i_1,\ p(t)^i_2,\ \ldots, p(t)^i_n] \quad (18)$$

where *n* is the size of the problem, i.e., the total number of robots.

$$\text{If} \quad rand^i_j > v(t)^i_j \rightarrow p(t)^i_j = 1 \quad (19)$$

$$\text{Otherwise} \quad p(t)^i_j = 0 \quad (20)$$

First, the initial population is evaluated by calculating the values of the three objective functions for each particle. The particles that represent non-dominated solutions are stored in a repository (*REP)*. Each particle keeps track of its best local position, which is the best solution obtained by this particle so far ($P_{i_{localBest}}$). At each iteration, the algorithm selects $P_{globalBest}$ that denotes the best position achieved so far by any particle in the population. It is selected by ranking the solutions in *REP* and choosing the one with the highest rank. Also, the velocity equation is updated according to equation (21) and the particle vector is updated in the same way in equations 17 to 20.

$$V(t+1) = w \times V(t) + c_1 \times V_{localbest}(t) + c_2 \times V_{globalbest}(t) \quad (21)$$

$$V_{localbest}(t) = \alpha \times p_{localbest}(t) + \beta \times \left(1 - p_{localbest}(t)\right) \quad (22)$$

$$V_{globalbest}(t) = \alpha \times p_{globalbest}(t) + \beta \times \left(1 - p_{globalbest}(t)\right) \quad (23)$$

where $+\beta = 1$, $\beta < 1$, $0 < \alpha$. α and β is control parameters, w represents the degree of belief

Algorithm 6.1: Coalition Formation Algorithm for Multi-Robot Systems	
1	**Inputs**: n, time, cost, Filtering_Rule, Task_Requirements, Criteria_Importance, allRobots
2	**Set** Selected_Robots ← [ ], Selected_Clts ← [ ], Robots ← [ ]
3	***Function*:** Filtered_Robots = Filter_Robots (allRobots, Filtering_Rule)
4	Selected_Robots =Filtered_Robots
5	***Function*:** apply QMOPSO to find the best coalition
6	**for** each (Robot in best coalition)
7	set Robot.State = busy
8	deploy Robot
9	**end**
10	**if** more than one Pareto-Optimal solution **then**
11	**if** time, cost, number of robots for each particle exceed thresholds (t, c, n)
12	remove particle
13	**else if** particle is infeasible
14	remove particle
15	**else**
16	Rank the Pareto-Optimal solutions
17	Select particle with highest ranking
18	selected_coalition = Particle with highest ranking
19	**end**
20	**else if** one Pareto-Optimal solution
21	selected_coalition = the Pareto-Optimal solution
22	**end**

on oneself, c_1 is the local maximum, and c_2 is the global maximum.

$P_{i_{localbest}}$ is updated by applying Pareto dominance. If the current position is dominated by the one in the memory, the one in the memory is kept; otherwise, the one in the memory is replaced by the current position. To update the *REP,* if the *REP* is empty, the current non-dominated particle is added to the *REP*; otherwise, the two particles are compared as follows: If both particles are feasible, Pareto-dominancy is applied; if one is feasible and the other infeasible, the feasible dominates; if both are infeasible, the one with the highest degree of constraint satisfactions is selected. We define a particle's feasibility degree as the degree of constraint satisfactions.

A task T_{req} is considered to have *U* capability requirements and *M* locational constraints. The capability requirements and locational constraints are considered as T_{req} and *C* respectively. Then a particle is feasible if it satisfies T_{req} and *C,* and it is infeasible otherwise. We determine a particle's feasibility degree as the weighted sum of feasibility degree with respect to T_{req} and *C*. If a particle satisfies *u* capability requirements and satisfies *m* locational constraints, then the particle's feasibility degree with respect to T_{req} and C are expressed as:

$$sat_{T_{req}} = u/U \tag{24}$$

$$sat_{C} = m/M \tag{25}$$

A particle's feasibility degree can now be calculated as:

Algorithm 6.2: QMOPSO-based Heuristic Algorithm	
1	**Initialization**: number of iteration, $j \leftarrow 0, V(t), P(t)$
2	t=0
3	value = Evaluate Population ($P(t)$)
4	store the position of particles that represents non-dominated vector in repository *REP*
5	initialize memory for each particle
6	$p_{i_{localBest}}[i] = P_i(t)$
7	$j = j + 1$
8	**while** $j <$ number of iteration
9	set $P_{globalBest}$ by selecting from the *REP*
10	**for** each particle $P(t)$
11	update velocity and position of particles
12	value = Evaluate Population
13	update the$P_{localBest}$
14	**if** the current $P(t)$is non-dominated by $p_{i_{localBest}}[i]$
15	$p_{i_{localBest}}[i] = P_i(t)$
16	**end**
17	**end**
18	select the non-dominated particles
19	update the *REP* by comparing current non-dominated particles with the ones in *REP*
20	**end**

$$P_{feas} = sat_{T_{req}} * W_T + sat_C * W_C \quad (26)$$

where W_T and W_C are the weights chosen such that:

$$W_T + W_C = 1, \quad 0 \leq W_T, W_C \leq 1 \quad (27)$$

Note that if a particle is feasible, then the feasibility degree is 1.

6.4. Algorithm Evaluation

In order to evaluate the algorithm, we have performed our experiments with a different problem and population sizes. In each experiment, speed, cost, position, battery level of each robot, and position of the target - which is the fire location – are randomly generated. All the robots are in the idle state at the beginning of each experiment. We have compared our algorithm with two well-known heuristic-based algorithms: NSGA-II and SPEA-II [51]. All algorithms have been implemented in Matlab.

Table 6.1 Algorithm evaluation parameters

Parameter	Value
General	
Population size	100, 200
Problem size (number of robots)	10-10000
Maximum number of iterations	100
$\alpha,\ \beta$	0.3, 0.7
w, c_1, c_2	0.25, 0.25, 0.5
Threshold for filtering	40%
Number of objectives	3
Number of sub-tasks	3
Criteria_Importance	time
NSGA-II and SPEA-II	
Tournament size	2
Pool size for tournament selection	Population number / 2
Mutation probability	10%
Crossover probability	90%
Distribution index for crossover	20
Distribution index for mutation	20

Table 6.1 shows the evaluation parameters along with their values.

6.4.1. Performance Metrics

- *Error Ratio*: The percentage of non-dominated solutions that are not part of a reference Pareto-set: When the true Pareto set is known, it is used as the reference Pareto-set. When the true Pareto-set is not known, the reference Pareto-set is obtained by combining Pareto-sets of all algorithms and applying non-dominancy.
- *Set Coverage (SC (A, B)):* Given two sets, is the percentage of non-dominated solutions in set B covered (i.e., dominated) by those in set A: If SC (A, B) > SC (B, A), then A is relatively better than B. A is absolutely better than B when SC (A, B) = 1 and SC (B, A) = 0.
- *Spacing*: standard deviation of distances of non-dominated solutions from their closest neighbors.
- *Processing Time (PT) (Sec):* the time needed for the algorithm to select the most efficient coalition.
- *Filtering Time (sec)*: The time needed for filtering the robots in QMOPSO.
- *Repository Update Time (sec):* The time needed by QMOPSO to update the repository at each iteration. It includes the delay incurred by the constraint handling method.

6.4.2. Results and Discussion

Test case 1 - convergence and diversity: Table 6.2 shows the error ratio, the set coverage, and the spacing metrics of QMOPSO, NSGA-II, and SPEA-II for a small-scale problem (i.e.., 10 robots).

Table 6.2 Error Ratio, Set Coverage and Spacing (10 Robots, Population size=200)

Algorithm	Error Rate	Set Coverage			Spacing
		QMOPSO	*NSGA-II*	*SPEA-II*	
QMOPSO	0.6	-	0.3	0	31.63
NSGA-II	0.8	0.66	-	-	51.45
SPEA-II	0.33	1	-	-	67.14

Table 6.3 Error Rate & Set Coverage (Population Size=100)

No. of Robots	Error Ratio			Set Coverage			
	QMOPSO	*NSGA-II*	*SPEA-II*	*(QMOPSO, NSGA-II)*	*(NSGA-II, QMOPSO)*	*(QMOPSO, SPEA-II)*	*(SPEA-II, QMOPSO)*
1000	0.23	0.32	0.30	0.72	0	0.8	0.1
5000	0.20	0.31	0.41	0.9	0.1	1	0
10000	0.11	0.26	0.2	1	0	0.93	0

We have also generated the true Pareto-optimal solutions by the enumerated search method. For the three algorithms, we have used a population size of 200. The error ratio of QMOPSO is higher than that of SPEA-II but lower than that of NSGA-II. The set coverage metric shows that 30% of solutions in NSGA-II are covered by QMOPSO and 66% of QMOPSO are covered by NSGA-II. Hence, NSGA-II is relatively better than QMOPSO. Since NSGA-II and SPEA-II do not cover each other, their relative dominance cannot be concluded. Overall, we observe that NSGA-II performs better than QMOPSO in terms of convergence. However, when it comes to diversity, QMOPSO outperforms NSGA-II and SPEA-II. This is concluded from the lowest spacing value in case of QMOPSO, which indicates a good distribution of solutions. Table 6.3 shows the error ratio for large-size problems (e.g., 1000, 5000, and 10000 robots). We observe that for any problem size, QMOPSO outperforms NSGA-II and SPEA-II. In fact, it achieves the lowest error ratio for the largest problem size (10000 robots). It shows a better convergence of QMOPSO for large-scale problems. SPEA-II has the highest error ratio for 5000 robots. Table 6.3 also shows the set coverage metric. As observed, when the problem size is 1000, QMOPSO is relatively better than both NSGA-II and SPEA-II. For a problem size of 5000, QMOPSO is absolutely better than SPEA-II as all solutions of SPEA-II are dominated by those of QMOPSO and none of QMOPSO solutions is dominated by those in SPEA-II. QMOPSO for a problem size of 5000 is relatively better than NSGA-II. However, for a problem size of 10000, QMOPSO is relatively better and absolutely better than SPEA-II and NSGA-II respectively. Overall,

Table 6.4 Spacing (Population Size=100)

No. of Robots	Spacing		
	QMOPSO	*NSGA-II*	*SPEA-II*
1000	18.23	23.11	40.36
5000	16.11	35.61	37.22
10000	8.24	19.23	28.19

SC results show that QMOPSO produces a better solution than NSGA-II and SPEA-II. Table 6.4 shows the spacing metric for three algorithms. We observe that QMOPSO attains the lowest value of spacing for any problem size, thereby achieving the highest diversity and even distribution of solutions. The diversity of NSGA-II lies between QMOPSO and SPEA-II. Figure 6.1 (a)-(c) shows the non-dominated-fronts obtained at some iterations for a problem size of 5000. We have found that with an iteration increase, the solutions in QMOPSO evolve more quickly than those in NSGA-II and SPEA-II. It shows the ability of QMOPSO to explore the search space more efficiently than others.

Test case 2 - processing time of the algorithms with a various number of robots: We compare the PT of our algorithm with NSGA-II and SPEA-II. The three algorithms are implemented and applied in the same environment, with the same number of robots, task requirements, and robot capabilities. The size of the population is 100. Figure 6.2 shows the processing time of the three algorithms with a various number of robots. For the QMOPSO algorithm, we consider the PT with and without the filtering method. We notice that the PT decreases when the filtering method is used. This is because the filtering method reduces the number of robots on which the algorithm runs. On the other hand, without filtering, the PT of the algorithm increases with an increase in the number of robots. The rationale behind this is the fact that the higher number of robots results in a higher dimension of the particle. As an important observation, the PT without these methods is still smaller than that of NSGA-II and SPEA-II; this is due to the simple mathematical operations of QMOPSO compared to other algorithms. In QMOPSO, the velocity equation is the sole equation updated at each iteration.

Test case 3 - repository update time: We have considered two types of constraints: Task requirements and location constraints. For the task requirements, we have considered 6 requirements (s_1, s_2, s_3, a_1, a_2, a_3) with a random number of units for each. For the locational constraints, we have represented the problem using CSP as described in Section 4.2 and we have considered three locational constraints ($s_1 = a_1$, $s_2 = a_2$, $s_3 = a_3$). A simplified method is used to calculate the satisfaction degree of a particle/coalition for the task requirements and locational constraints. We

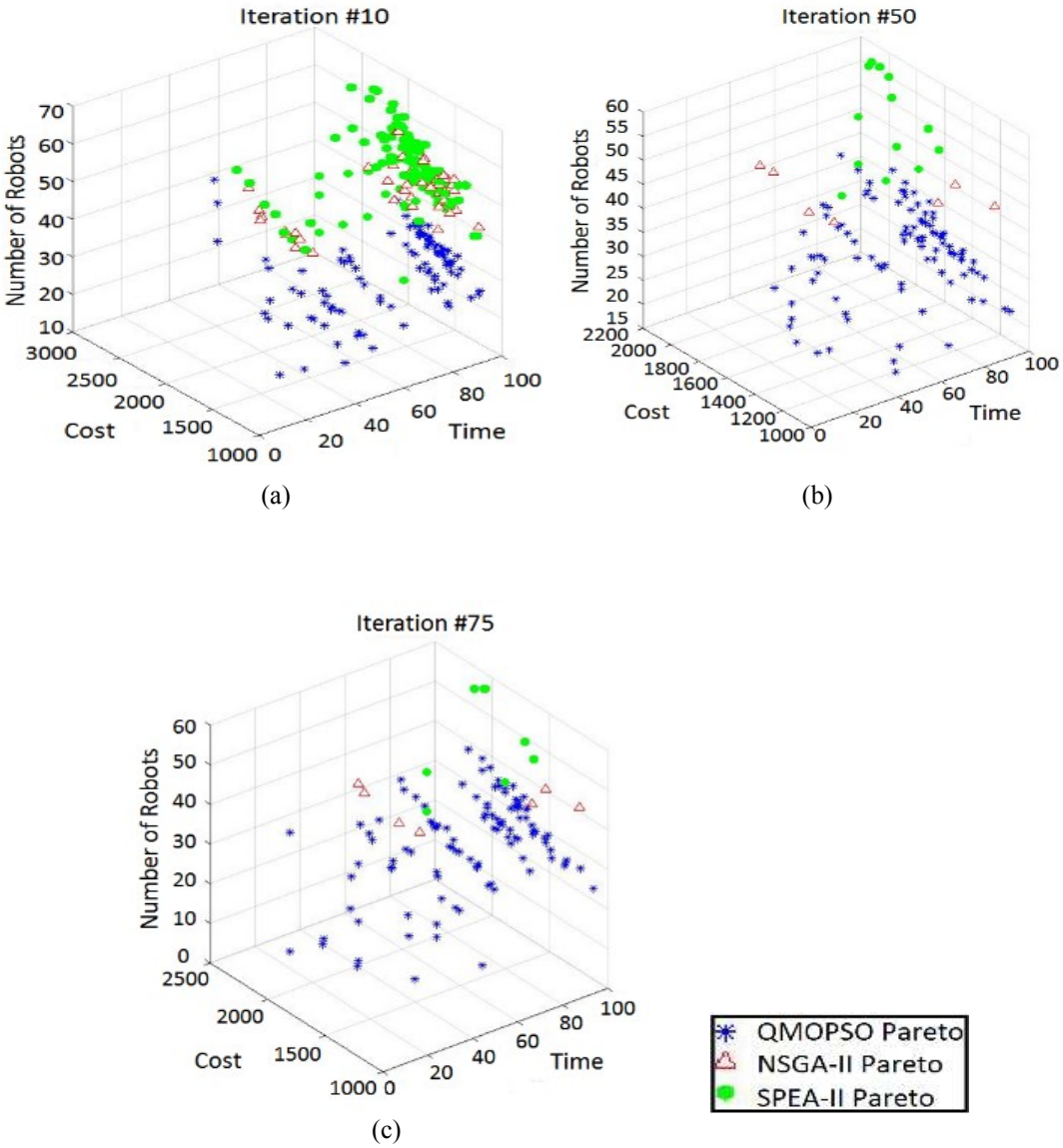


Figure 6.1 Non-dominated fronts obtained at different iteration for problem size 5000 and population size 200

(a) After 10 iterations
(b) After 50 iterations
(c) After 75 iterations

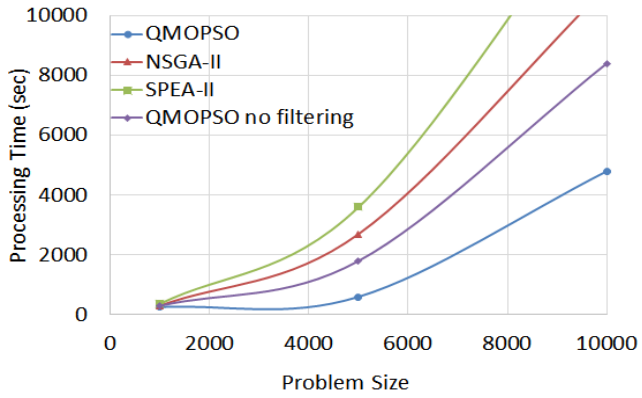


Figure 6.2 Processing time with different problem sizes (Population size=100)

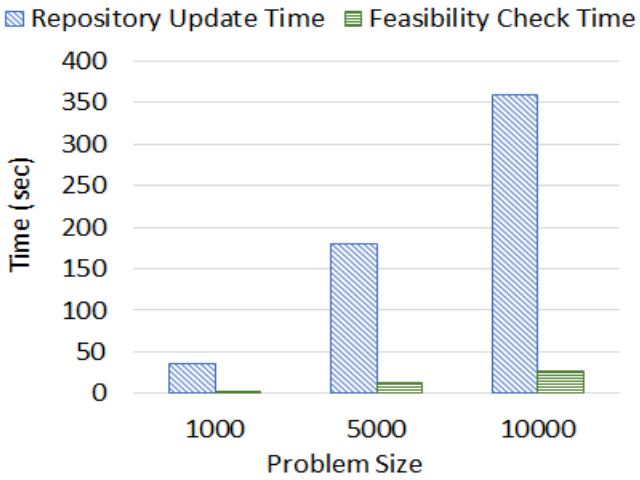


Figure 6.3 The effect of feasibility check on average repository updating time

calculate the effect of our proposed method to solve the two constraints (task requirements and locational constraints) on the average repository updating time. The results in Figure 6.3 demonstrate the time needed to perform the feasibility check versus the overall repository update time, considering different numbers in a population. As we notice, the time needed for our proposed feasibility checking method is negligible compared to the total time for updating the repository.

Test case 4 - filtering time: We have also calculated the time needed to perform the filtering function compared to the overall processing time of QMOPSO. Figure 6.4 shows that the filtering time is negligible compared to the overall processing time of the algorithm. The time needed for filtering does not introduce additional overhead on the algorithm processing time. Since this method

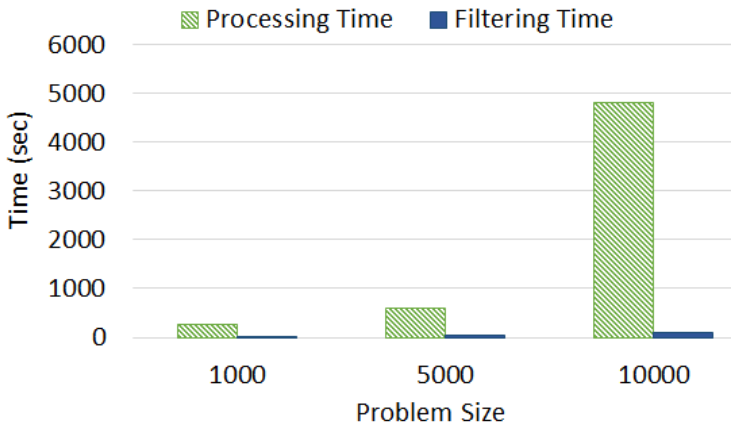


Figure 6.4 The time needed for the filtering function

excludes some robots using battery levels, it ensures that the remaining robots have sufficient battery to accomplish the task. Since it does not affect the processing time of the algorithm, the overall efficiency is achieved.

6.5. Conclusion

In this chapter, we have proposed a coalition formation algorithm for multi-robot systems. To show the effectiveness of our algorithm, we have conducted extensive simulation experiments and compared our algorithm with other existing algorithms. The results demonstrate that the proposed algorithm cannot only improve the solution but can also significantly reduce the processing time. They also show that the filtering and the repository updating mechanisms do not add overhead on the processing time. It is also observed that QMOPSO achieves higher diversity, the lowest error rate, and produces a better solution compared to NSGA-II and SPEA-II for large problem sizes.

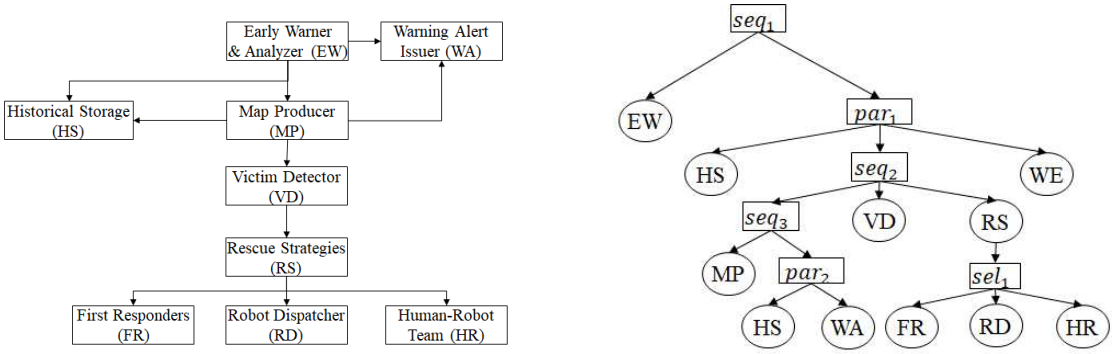
Chapter 7

7. Application Component Placement in NFV-based Hybrid Cloud/Fog Systems with Mobile Fog Nodes

7.1. Introduction

Many service providers use cloud computing to deploy their applications as a way to reduce cost whilst exploiting the elasticity feature provided by the cloud. However, the wide area network used to connect the cloud to the end-users might cause high latency, which may not be tolerable for some applications. On the other hand, fog computing, a concept introduced by CISCO in 2012, provides an intermediate layer between end-users and the cloud which allows the deployment of some of the application components in the fog at the edge while keeping some others in the cloud, thereby reducing latency [13].

Applications can be implemented in cloud/fog systems as a set of interacting components that can be executed in sequence, in parallel, or by using more complex constructs such as selections and loops. They can, therefore, be modeled as structured graphs with sub-structures consisting of these constructs. The selection sub-structure introduces non-determinism in the execution.



(a) (b)

Figure 7.1 Earthquake early warning and recovery application components

(a) Component-based application
(b) Structured VNF-FG representation

Meaning, there are uncertainties associated with the components' executions. These uncertainties imply that, for such non-deterministic input, from an initial state, there may be none, exactly one, or many possible transitions. This chapter assumes that applications' components are implemented as VNFs. The structured graphs representing the applications are therefore VNF Forwarding Graphs (VNF-FG) (i.e., sets of VNFs chained in specific orders).

The focus of this chapter is on application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. It tackles the challenges of heterogeneity of cloud and fog, fog nodes mobility, and non-deterministic VNF-FG graphs. The heterogeneity is addressed by considering makespan (an important Quality of Service (QoS) criterion) in addition to cost (a budget that the application provider should pay for consuming resources) when it comes to optimization. Indeed, cost minimization encourages cloud usage while makespan minimization encourages fog usage. A compromise is required for the appropriate placement decision. The mobility of fog nodes is modeled using the Random Waypoint (RWP) model [16]. Based on the stationary distribution of fog nodes' location, we calculate the expected makespan and cost for the sub-structures: sequence, parallel, selection, and loop. Next, we aggregate the sub-structures' calculations to obtain the application's makespan and execution cost. The non-determinism nature of the VNF-FGs is tackled by assigning probabilities to selection sub-structures and mean numbers of iterations to loop sub-structures. These probabilities can be obtained through prediction models. The problem is formulated as an Integer Linear Programming (ILP) problem, and regarding the complexity, a Tabu Search-based Component Placement (TSCP) algorithm is proposed to find a

sub-optimal solution in feasible time. The proposed model and algorithm are implemented in order to analyze their performance.

The rest of this chapter is organized as follows. First, it presents the system model, followed by the description of the mobility model and the optimization problem. Then, it discusses the designed Tabu Search-based component placement heuristic. After that, it presents the simulation parameters and settings followed by the validation results. We will conclude this chapter at the end.

7.2. System Model

In this section, we explain the modeling of the components implemented as VNFs, the structured VNF-FGs, the cloud and the fog systems, and the IoT/end-users that may interact with components.

VNFs – Each component of the application is implemented as a VNF. Let T be the set of VNF types in the system. We denote the type t of a VNF with f^t, which can be shared by more than one application. The resource requirements for processing a VNF f^t per unit of resource (CPU, memory, storage) is represented by ϑ_{f^t} and the processing capacity of f^t is represented by c_{f^t}. The set of available instances for f^t is represented by I_{f^t}. Each VNF type $t \in T$ has a predefined license cost ∂_{f^t}. We denote the maximum allowed VNF processing utilization with μ_{f^t}.

Structured VNF-FGs - Let Req be the set of structured VNF-FG requests received by the system. We represent a single request with $R \in Req$. The set of required VNF types for request R is indicated by vnf_R ($vnf_R \subset T$). The structured VNF-FG for request R is represented as a tree [14] in which leaf nodes represent VNFs belonging to vnf_R, while a middle node with index i, namely S_i, represents one of the sub-structures (i.e., sequence, parallel, selection, loop) i.e., $S_i \in \{seq, par, sel, loop\}$. Each middle node S_i has at least two children where a child can be either a VNF or a sub-structure. Figure 7.1 (a) shows the earthquake early warning and recovery application's components presented in Chapter 2. According to the constructed tree (i.e., Figure 7.1 (b)), we can define the relation between VNFs in the chain; the immediate predecessor of a VNF f^t can be determined by parsing the tree [14]. Let $IP(f^t)$ denote the immediate predecessor of f^t and $(IP(f^t), f^t)$ a VNF edge if and only if the packets from VNF $IP(f^t)$ must be forwarded

to the VNF f^t. We assume that the amount of traffic sent from $IP(f^t)$ to f^t for request R is $A_{f^t}^R$. More details of the structured VNF-FG can be found in [14].

Cloud/Fog - We consider that the cloud and the fog domains are modeled as graphs: $G^Z = (N^Z, E^Z)$, where $Z = C|F$ is used to indicate cloud or fog. Here, N^Z is a set of physical cloud/fog nodes while E^Z is a set of cloud/fog edges representing the communication links among nodes. We use c_{n^Z} and γ_{n^Z} to represent the capacity and the cost, respectively, per unit of resource (e.g., CPU, memory, storage) usage of node n^Z. We represent the threshold for resource usage of a cloud/fog node with μ_{n^Z}. The delay per traffic unit processing of VNF type t hosted on a cloud/fog node $n^Z \in N^Z$ is represented by $D_{n^Z}^{f^t}$.

Table 7.1 Summary of key notations and decision variables

Input Parameters	
VNFs	
T	Set of VNF types
f^t	VNF of type $t \in T$
ϑ_{f^t}	Resource requirements for processing f^t (in processing units)
c_{f^t}	Processing capacity of f^t (in traffic units)
I_{f^t}	Set of VNF instances associated to f^t
∂_{f^t}	License cost for f^t
Structured VNF-FGs	
Req	Set of structured VNF-FG requests assigned to the system
R	Single request for VNF-FG, $R \in Req$
vnf_R	Set of required VNF types for request R, $vnf_R \subset T$
S_i	One of the sub-structures, $S_i \in \{seq, par, sel, loop\}$
it	Expected number of iterations of a $loop$ sub-structure
h_{f^t}	Probability of selecting child f^t of a sel sub-structure
$IP(f^t)$	Immediate predecessor of f^t
$(IP(f^t), f^t)$	VNF edge between f^t and $IP(f^t)$
$A_{f^t}^R$	Amount of traffic from $IP(f^t)$ to f^t for request R
Cloud/Fog Network	
N^Z	Set of cloud/fog nodes
E^J	Set of all possible communication in the network, $E^J = E^Z \cup E^{CF}$
E^Z	Set of cloud/fog edges, $E^Z \in E^J$
E^{CF}	Set of edges between cloud and fog nodes $E^{CF} \in E^J$
c_{n^Z}	Cloud/fog node capacity (in processing resource units)
γ_{n^Z}	Cloud/fog node cost per processing unit usage
$D_{n^Z}^{f^t}$	The processing delay of f^t on node n^Z
$D_{e^J}(A, \boldsymbol{X}, \boldsymbol{Y})$	The transmission delay of sending traffic A through e^J
$\rho_{e^J}(A, \boldsymbol{X}, \boldsymbol{Y})$	The transmission cost of sending traffic A through e^J
$BW_{e^J}(\boldsymbol{X}, \boldsymbol{Y})$	The bandwidth capacity of e^J
$Lat_{e^J}(\boldsymbol{X}, \boldsymbol{Y})$	The network latency of e^J

Input Parameters (Cont.)	
IoT Devices/End-Users	
U_R	Set of IoT/end-users for request R
E^{u,n^Z}	Set of links between IoT/end-users and $n^Z \in N^Z$
$\omega^R_{u\times f^t}$	1, if there is communication between u and f^t for request R
$A^R_{u\times f^t}$	Amount of traffic between u and f^t for request R
$D_{e^{u,n^Z}}(A,\boldsymbol{X},\boldsymbol{Y})$	The transmission delay of sending traffic A between u and n^Z
$\rho_{e^{u,,n^Z}}(A,\boldsymbol{X},\boldsymbol{Y})$	The transmission cost of sending traffic A between u and n^Z
$BW_{e^{u,n^Z}}(\boldsymbol{X},\boldsymbol{Y})$	The bandwidth capacity between u and n^Z
$Lat_{e^{u,n^Z}}(\boldsymbol{X},\boldsymbol{Y})$	The network latency u and n^Z
Location Analysis	
v^{n^Z}	Movement velocity of node n^Z
$p_{st}^{n_z}$	probability that node n^Z is stationary
$p_p^{n_z}$	Probability that a node n^Z is in pause
$E[L]$	Expected distance between two waypoints
$E(PS)$	Expected value of pause time
$f_x^{n^Z}(\boldsymbol{X})$	Stationary PDF of location $\boldsymbol{X}=(X,Y)$ of node n^Z
$f_{init}^{n^Z}(\boldsymbol{X})$	Initial spatial distribution of location $\boldsymbol{X}=(X,Y)$ of node n^Z
$f_m^{n^Z}(\boldsymbol{X})$	Stationary PDF of location $\boldsymbol{X}=(X,Y)$ of mobile nodes moving in $[0,1]^2$ according to RWP model with $p_{st}^{n_z}=p_p^{n_z}=0$
Decision Variables	
x_{i,f^t,n^C}	Binary variable, indicating if instance *i* of VNF type t is instantiated on cloud/fog node
x^R_{i,f^t,n^C}	Binary variable, indicating if instance *i* of VNF type t instantiated on cloud/fog node is assigned to request *R*

Cloud nodes are not mobile, while fog nodes can be either mobile or stationary. To model such behavior, we define $p_{st}^{n^Z}$ as the probability that node n^Z is not mobile. Obviously, when $Z = C$ then $p_{st}^{n^Z} = 1$. We also assume that cloud and fog nodes are located in a two-dimensional rectangular region $Q \in [0,1]^2$. Note that two-dimensional localization has also been used in ad-hoc networks [120]. Thus, $\boldsymbol{X} = (x,y) \in [0,1]^2$ denotes the location of a cloud/fog node.

Next, we model the communication between cloud and fog nodes. We assume that E^{CF} is the set of edges that indicate the communication between cloud and fog nodes. Let $E^J = E^Z \cup E^{CF}$ be the set of all possible communications in the network, then an edge is represented by $e^J = (n^{Z_l}, n^{Z_m}) \in E^J$, which represents communication between any two cloud/fog nodes, or one cloud node and a fog node. When the location of n^{Z_l} is $\boldsymbol{X}$ and the location of n^{Z_m} is $\boldsymbol{Y}$, then, for e^J, we define $D_{e^J}(A,\boldsymbol{X},\boldsymbol{Y})$ and $\rho_{e^J}(A,\boldsymbol{X},\boldsymbol{Y})$, which represent the delay and the cost, respectively, of transmitting traffic amount of A through $e^J \in E^J$. We also define $BW_{e^J}(\boldsymbol{X},\boldsymbol{Y})$ which represents the bandwidth capacity of e^J, and $Lat_{e^J}(\boldsymbol{X},\boldsymbol{Y})$, which represents the network latency. In this

regard, the same nodes will communicate with different delays, costs, and bandwidths when they are located in various points in the rectangular region Q. We represent the threshold for usage of the bandwidth capacity by μ_{e^J}.

IoT devices/End-users – The application components may communicate with IoT/end-users. We denote the set of IoT/end-users for request $R \in Req$ by U_R. We assume E^{u,n^z} are the links that indicate the communication between IoT/end-users and cloud/fog nodes. We define two matrices, $\omega^R_{n\times m}$ and $A^R_{n\times m}$, which represent respectively the communication and the amount of traffic exchanged between IoT/end-users and the VNFs of request R. n represents the number of IoT/end-users communicating with VNFs in request R while m represents the number of VNFs in R. $\omega^R_{u\times f^t} \in \{0,1\}$ is 1 if there is communication between IoT/end-user u and the VNF f^t of request R, while $A^R_{u\times f^t}$ provides the amount of traffic exchanged between IoT/end-user u and the VNF f^t of request R.

The locations of IoT/end-users are assumed to be fixed and defined in the region Q. $D_{e^{u,n^z}}(A, \boldsymbol{X}, \boldsymbol{Y})$ and $\rho_{e^{u,,n^z}}(A, \boldsymbol{X}, \boldsymbol{Y})$ are respectively the delay and the cost of sending traffic of amount A between an IoT/end-user and a cloud/fog node when they are located in locations $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, in region Q. $BW_{e^{u,n^z}}(\boldsymbol{X}, \boldsymbol{Y})$ and $Lat_{e^{u,n^z}}(\boldsymbol{X}, \boldsymbol{Y})$ are respectively the bandwidth capacity and the network latency between an IoT/end-user and a cloud/fog node. The symbol $\mu_{e^{u,n^z}}$ indicates the maximum allowed link utilization for the communication between an IoT/end-user and a cloud/fog node.

7.3. Cloud/Fog Node Location Analysis and Optimization Formulation

Here we first calculate the stationary Probability Density Function (PDF) of cloud/fog nodes locations, and then we explain the objective function and the constraints of our optimization model.

7.3.1. Cloud/Fog Node Location Analysis

To calculate the stationary probability distribution of the cloud/fog node location we focus on a mobile fog node. We calculate the PDF of its location and explain how the PDF can also be used for a stationary fog node or for a cloud node.

We model the mobility of a mobile fog node n^z with a RWP Model that has been used in the ad-hoc networking research community [16]. We assume node n^z moves independently of other nodes in a region $Q \in [0,1]^2$. The node is placed in an arbitrary location namely, "waypoint" l_0 in the region $\boldsymbol{X} \in Q$ according to an initial spatial distribution represented by $f_{init}^{n^z}(\boldsymbol{X})$. The fog node selects its first random destination point; "waypoint" l_1 to move to. It goes there and pauses for a random duration. It then picks another waypoint l_2 to visit, moves towards it, and pauses at it for another random duration. The process continues in similar steps. We assume the movement velocity is $v^{n^z} > 0$. We assume that the pause duration after each movement period is chosen from an arbitrary PDF, namely, $f_{PS}^{n^z}(ps)$ in the interval $[ps_{min}, ps_{max}]$, with $ps_{min} \geq 0$ and well-defined expected value $E[PS]$.

Let $\boldsymbol{X}$ be the random variable representing the waypoint of a fog node. The stationary probability distribution of the fog node location is then calculated as (1) [16]:

$$f_x^{n^z}(\boldsymbol{X}) = p_{st}^{n_z} f_{init}^{n^z}(\boldsymbol{X}) + \left(1 - p_{st}^{n^z}\right) p_p^{n^z} + \left(1 - p_{st}^{n^z}\right)\left(1 - p_p^{n^z}\right) f_m^{n^z}(\boldsymbol{X}) \qquad (1)$$

where $\boldsymbol{X} \in Q$ is the value of the random variable x and denotes a waypoint in region Q. $p_p^{n^z}$ is the probability that a node is in pause, calculated as (2):

$$p_p^{n^z} = \frac{E(PS)}{E(PS) + \frac{1}{v^{n^z}} E(L)} \qquad (2)$$

where $E[L]$ is the expected distance between two waypoints placed uniformly at random in Q, which is calculated as:

$$E[L] = 0.521405 \qquad (3)$$

For $\boldsymbol{X} = (x, y)$, the term $f_m^{n^z}$(x,y) as used in (1) is the mobility component, is defined as :

$$f_m^{n^z}(x,y) = \begin{cases} f_m^*(x,y) & 0 < x \leq 0.5, 0 < y \leq x \\ f_m^*(y,x) & 0 < x \leq 0.5, x \leq y \leq 0.5 \\ f_m^*(1-y,x) & 0 < x \leq 0.5, 0.5 \leq y \leq 1-x \\ f_m^*(x,1-y) & 0 < x \leq 0.5, 1-x < y < 1 \\ f_m^*(1-x,y) & 0.5 \leq x < 1, 0 < y \leq 1-x \\ f_m^*(y,1-x) & 0.5 \leq x < 1, 1-x \leq y \leq 0.5 \\ f_m^*(1-y,1-x) & 0.5 \leq x < 1, 0.5 \leq y \leq x \\ f_m^*(1-x,1-y) & 0.5 \leq x < 1, x \leq y < 1 \\ 0 & otherwise \end{cases} \qquad (4)$$

where f_m^* is defined on $Q^* = \{(x,y) \in [0,1]^2 | \ (0 < x \leq 0.5) \wedge (0 < y \leq x)\}$, with

$$f_m^*(x,y) = 6y + \frac{3}{4}(1-2x+2x^2)\left(\frac{y}{y-1} + \frac{y^2}{(x-1)x}\right) + \frac{3y}{2}\left[(2x-1)(y+1)\ln\left(\frac{1-x}{x}\right) + (1-2x+2x^2+y)\ln\left(\frac{1-y}{y}\right)\right] \quad (5)$$

The PDF calculated in Eq. (1) can be used for stationary fog nodes and for cloud nodes as well. Indeed, for these nodes we have $p_{st}^{n^Z} = 1$, which, by using (1), gives the PDF of the node location as $f_x^{n^Z}(\boldsymbol{X}) = f_{init}^{n^Z}(\boldsymbol{X})$, as expected.

7.3.2. Optimization Formulation

We formulate our problem as an optimization problem with the objective of minimizing the weighted aggregated function of makespan and cost. We define the following decision variables:

$$x_{i,f^t,n^Z} = \begin{cases} 1, & if\ instance\ i\ of\ f^t\ is\ placed\ on\ n^Z \\ 0, & otherwise \end{cases} \quad (6)$$

$$x^R_{i,f^t,n^Z} = \begin{cases} 1, & if\ instance\ i\ of\ f^t\ on\ n^Z\ is\ assigned\ to\ R \\ 0, & otherwise \end{cases} \quad (7)$$

In the rest of this section, we explain the expected makespan and cost calculations, followed by the objective function and the constraints. Table 7.1 lists the key notations and decision variables.

A) Makespan and Communication Cost Computation

The makespan is an application's execution time, defined as the time it takes for the first component to start execution until the execution of the last component is completed [121][14]. Note that the communication times with the IoT/end-users are also included in makespan calculations. In turn, the application execution cost is defined as the monetary cost for communication between application components and also between IoT/end-users and components.

The calculation of the expected application makespan and communication cost is performed based on parsing the associated tree structure of the VNF-FG, as explained in Section 7.2. The time and cost of the leaf nodes representing the VNFs are calculated first. These values are then aggregated to calculate the time and the cost for the middle nodes. The middle nodes represent sub-structures. The total makespan and the cost of the root of the tree are then calculated by aggregating the calculated time/cost values of the nodes from the bottom to the top according to the tree structure.

1) VNFs-Level Calculation

We first explain the VNF time calculations and then the VNF cost calculations. The processing time of the traffic received by each VNF from its immediate predecessors and IoT/end-users belonging to request R is calculated as below:

$$M_{proc}(R, f^t) = \sum_{n^Z \in N^Z} \sum_{i \in I_{f^t}} x^R_{i,f^t,n^Z} . A^R_{f^t} . D^{f^t}_{n^Z} \tag{8}$$

The communication time required to transmit traffic $A^R_{f^t}$ to a VNF f^t belonging to a VNF-FG request R from $IP(f^t)$ over edge $e^J_{lm} = (n^{Z_l}, n^{Z_m})$ and to transmit traffic $A^R_{u \times f^t}$ between f^t and IoT/end-users over edge e^{u,n^Z} is calculated as:

$$\begin{aligned} M_{com}(R, f^t) = max(& \sum_{n^Z \in N^Z} \sum_{i,j \in I_{f^t}} x^R_{i,f^t,n^{Z_l}} . x^R_{j,IP(f^t),n^{Z_m}} . E(D_{e^J_{lm}}(A^R_{f^t})), \\ & \sum_{n^Z \in N^Z} \sum_{i \in I_{f^t}} \sum_{u \in U_R} x^R_{i,f^t,n^Z} . \omega^R_{u \times f^t} . E(D_{e^{u,n^Z}}(A^R_{u \times f^t}))) \end{aligned} \tag{9}$$

In Eq. (9), $E(D_{e^J_{lm}}(A^R_{f^t}))$is the expected delay of transmitting traffic $A^R_{f^t}$ on edge e^J_{lm}, which is calculated as Eq. (10).

$$E\left(D_{e^J_{lm}}(A)\right) = \int_{[0,1]^2} \int_{[0,1]^2} f_x^{n^{Z_l}}(\boldsymbol{X}) . f_y^{n^{Z_m}}(\boldsymbol{Y}) . D_{e^J_{lm}}(A, \boldsymbol{X}, \boldsymbol{Y}) . dX\, dY \tag{10}$$

In Eq. (10), $\boldsymbol{X}$ and $\boldsymbol{Y} \in [0,1]^2$ are n^{Z_l} and n^{Z_m} random location variables as defined in Section 7.3.1. $D_{e^J_{lm}}\left(A^R_{f^t}, \boldsymbol{X}, \boldsymbol{Y}\right)$ is the data transfer time for sending traffic $A^R_{f^t}$ on edge e^J_{lm}. It is calculated as the relation of the size of the transmitted traffic between two nodes to the edge bandwidth. The network latency is considered as well, which is a function of the distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$. This calculation is given below:

$$D_{e^J_{lm}}(A, \boldsymbol{X}, \boldsymbol{Y}) = \frac{A}{BW_{e^J_{lm}}(\boldsymbol{X}, \boldsymbol{Y})} + Lat_{e^J_{lm}}(\boldsymbol{X}, \boldsymbol{Y}) \tag{11}$$

Table 7.2 The cost and the makespan estimation for $S_i \in \{seq, par, sel, loop\}$

Sub-structures	Communication Cost $C_{com}(S_i)$	Processing Time $M_{proc}(S_i)$	Communication Time $M_{com}(S_i)$
$S_i\ is\ seq$	$\sum_{f^t \in S_i} C_{com}(f^t)$	$\sum_{f^t \in S_i} M_{proc}(f^t)$	$\sum_{f^t \in S_i} M_{com}(f^t)$
$S_i\ is\ par$	$\sum_{f^t \in S_i} C_{com}(f^t)$	$\max_{f^t \in S_i} M_{proc}(f^t)$	$\max_{f^t \in S_i} M_{com}(f^t)$
$S_i\ is\ sel$	$\sum_{f^t \in S_i} h_{f^t}.\, C_{com}(f^t)$	$\sum_{f^t \in S_i} h_{f^t}.\, M_{proc}(f^t)$	$\sum_{f^t \in S_i} h_{f^t}.\, M_{com}(f^t)$
$S_i\ is\ loop$	$it.\ \sum_{f^t \in S_i} C_{com}(f^t)$	$it.\ \sum_{f^t \in S_i} M_{proc}(f^t)$	$it.\ \sum_{f^t \in S_i} M_{com}(f^t)$

In Eq. (9), $E(D_{e^{u,n^z}}(A^R_{u\times f^t}))$ is the expected delay for the transmission of $A^R_{u\times f^t}$ amount of traffic between an IoT/end-user and a cloud/fog node. Eq. (12) gives this calculation. Here, $\boldsymbol{Z} \in [0,1]^2$ is the IoT/end-user location and $D_{e^{u,n^z}}(A, \boldsymbol{Z}, \boldsymbol{X})$ is the data transfer time for sending traffic of amount A between IoT/end-users and cloud/fog nodes. In Eq. (12), $D_{e^{u_j,n_i^z}}(A, \boldsymbol{Z}, \boldsymbol{X})$ is calculated in the same way as Eq. (11).

$$E\left(D_{e^{U,n^z}}(A)\right) = \int_{[0,1]^2} f_x^{n_i^z}(\boldsymbol{X}).\, D_{e^{u_j,n_i^z}}(A, \boldsymbol{Z}, \boldsymbol{X})\, dX \tag{12}$$

The approach described above for time can also be used to calculate the communication costs. The communication cost incurred by utilizing the links to send traffic to a VNF f^t from its immediate predecessor and also utilizing the links to send traffic between IoT/end-users and f^t are calculated as:

$$C_{com}(R, f^t) = \sum_{n^z \in N^z} \sum_{i,j \in I_{f^t}} x^R_{i,f^t,n^{z_l}}.\, x^R_{j,IP(f^t),n^{z_m}}\, .\, E(\rho_{e^J_{lm}}(A^R_{f^t})) \ +$$

$$\sum_{n^z \in N^z} \sum_{i \in I_{f^t}} \sum_{u \in U_R} x^R_{i,f^t,n^z}.\, \omega^R_{u\times f^t}.\, E(\rho_{e^{u,n^z}}(A^R_{u\times f^t})) \tag{13}$$

To calculate $E(\rho_{e^J_{lm}}(A^R_{ft}))$, the $D_{e^J_{lm}}(A, \boldsymbol{X}, \boldsymbol{Y})$ in Eq. (10) should be replaced by $\rho_{e^J_{lm}}(A, \boldsymbol{X}, \boldsymbol{Y})$. Similarly, to calculate $E(\rho_{e^{u,n^Z}}(A^R_{u\times f^t}))$, the $D_{e^{u_j,n^Z_i}}(A, \boldsymbol{Z}, \boldsymbol{X})$ in Eq. (12) must be replaced by $\rho_{e^{u_j,n^Z_i}}(A, \boldsymbol{Z}, \boldsymbol{X})$.

2) VNF-FG Level Calculation

The calculations of the processing/communication time and the communication cost for the sub-structures sequence, parallel, selection, and loop are shown in Table 7.2. In a sequence sub-structure, the time and the cost of all of its children are accumulated. A $loop$ can be considered as a sequence structure that is repeated for a certain number of iterations. We define it as the expected number of iterations of a loop structure, it is calculated as: $it = \frac{q}{1-q}$ where q is the probability of the loop's occurrence. For a parallel sub-structure, all of its children are executed in parallel, hence the time is determined based on the maximum time value of its children. However, the cost is the sum of the costs for all children. The calculation for a selection sub-structure, the probabilities of the selection's children are involved in the calculation. Let h_{f^t} represent the probability of selecting a child f^t of a selection sub-structure. $h_{f^t} = 1$ for the children of sequence, parallel, and loop sub-structures.

Finally, to calculate the total makespan and the cost of a VNF-FG, the makespan and the cost of the root of the tree are computed by aggregating the time and the cost of the VNFs and of the basic sub-structures in a bottom-to-top manner according to the tree structure. The total makespan and the total cost of a VNF-FG request R are calculated as given in Eq. (14) and Eq. (15), respectively:

$$M(R) = M_{proc}(R, root) + M_{com}(R, root) \quad (14)$$

$$C(R) = C_{com}(R, root) \quad (15)$$

B) Optimization Formulation

In this section, we explain the objective function and the constraints of the optimization problem. Our objective is to enable the embedding of VNF-FGs in cloud and fog NFVIs such that the makespan and the cost are minimized, as shown in Eq. (16).

$$obj = Min\left(\alpha \sum_{\forall R \in Req} M(R) + (1-\alpha)\left[\sum_{\forall R \in Req} C(R) + C_{dep}\right]\right) \quad (16)$$

C_{dep} represents both license cost of VNFs and the hosting cost,

$$C_{dep} = C_{Lic} + C_{hst} \quad (17)$$

The license cost is the cost of the total software license costs for the VNFs instantiation,

$$C_{Lic} = \sum_{n^Z \in N^Z} \sum_{t \in T} \sum_{i \in I_{f^t}} x_{i,f^t,n^Z} . \partial_{f^t} \quad (18)$$

and the hosting cost is the cost of the assigned resources to VNFs belonging to VNF-FG requests. It is calculated as:

$$C_{hst} = \sum_{n^Z \in N^Z} \sum_{i \in I_{f^t}} x_{i,f^t,n^Z} . \gamma_{n^Z} . \vartheta_{f^t} \quad (19)$$

In Eq. (16), α is the weight parameter that defines priorities between makespan and cost, $1 \geq \alpha \geq 0$. E.g., $\alpha = 1$ motivates placement in the fog, while $\alpha = 0$ motivates placement in the cloud. Generally, the fog provides lower latency due to its proximity to IoT/end-users, however, the resources in the fog are more expensive.

Now, we explain the constraints involved in the problem. Eq. (20) ensures that the total resources required by instances of all VNF types do not exceed the capacity of a cloud/fog node:

$$\sum_{t \in T} \sum_{i \in I_{f^t}} \vartheta_{f^t} . x_{i,f^t,n^Z} \leq \mu_{n^Z} . c_{n^Z} \quad \forall n^Z \in N^Z \quad (20)$$

Eq. (21) ensures that the communication links where the source and the destination are both in the cloud or both in the fog, or where the source is in one and the destination is in the other are not overloaded from the aspect of link utilization. A similar discussion exists for the communication links between the IoT/end-users and cloud/fog nodes according to the constraint in Eq. (22).

$$\sum_{\forall R \in Req} A^R_{f^t} . x^R_{i,f^t,n^{Z_l}} . x^R_{j,IP(f^t),n^{Z_m}} \leq \mu_{e^J} . BW_{e^J} \quad \forall e^J \in E^J \quad (21)$$

Algorithm 7.1: Tabu Search Algorithm	
1	**initialization:** Create initial placement randomly S_0,
2	$S_{curr} \leftarrow S_0, S_{best} \leftarrow S_0, j \leftarrow 0, Tabu\_list \leftarrow \emptyset$
3	**while** $j \leq i_{stop}$
4	$neighborhood\_list \leftarrow$ create candidate neighborhood list
5	**for** each $neighbor \in [neighborhood\_list]$
6	evaluate the neighbor $E(neighbor)$
7	**end**
8	$best\_neighbor \leftarrow \underset{Neighbor}{\text{argmin}}\ E(neighbor)$
9	$best\_move \leftarrow$ select the move that led to $best\_neighbor$
10	$j \leftarrow j + 1$
11	**if** $best\_move$ is not in $Tabu\_list$
12	$Tabu\_list \leftarrow best\_move$ for i_{tabu} iterations
13	**else if** $fitness(best\_neighbor) < E(S_{best})$
14	remove $best\_move$ from $Tabu\_list$
15	**end**
16	**if** $fitness(best\_neighbor) < E(S_{best})$
17	$S_{best} \leftarrow best\_neighbor$
18	$j \leftarrow 0$
19	**end**
20	**end**
21	**return** S_{best}

$$\sum_{\forall R \in Req} A^R_{u \times f^t} \cdot x^R_{i,f^t,n^Z} \cdot \omega^R_{u \times f^t} \leq \mu_{e^{u,n^Z}} \cdot BW_{e^{u,n^Z}} \qquad \forall u \in U_R, n^Z \in N^Z \tag{22}$$

Eq. (23) ensures that the capacity of an instance of a VNF f^t is not exceeded by the total traffic requested by its immediate predecessor(s) and the IoT/end-users communicating with it.

$$\sum_{\forall R \in Req} (A^R_{f^t} \cdot x^R_{i,f^t,n^Z} + A^R_{u \times f^t} \cdot x^R_{i,f^t,n^Z} \cdot \omega^R_{u \times f^t}) \leq \mu_{f^t} \cdot c_{f^t} \qquad \forall t \in T, \forall i \in I_{f^t}, \forall u \in U_R, \forall n^Z \in N^Z \tag{23}$$

Eq. (24) ensures that the assigned VNF instances are already deployed in the network and Eq. (25) ensures that at least one instance of each required VNF type is deployed.

$$x^R_{i,f^t,n^Z} \leq x_{i,f^t,n^Z} \qquad \forall R \in Req, f^t \in vnf_R, i \in I_{f^t}, n^Z \in N^Z \tag{24}$$

$$\sum_{\forall n^Z \in N^Z} \sum_{\forall i \in I_{f^t}} x_{i,f^t,n^Z} \geq 1 \qquad \forall t \in T \tag{25}$$

It should be noted that Eq. (9) and (21) and the processing and communication time equations in Table 7.2 for parallel sub-structure are non-linear. However, they can be linearized by replacing

them with linear equations. For instance, $x^R_{i,f^t,n^{z_l}} . x^R_{j,IP(f^t),n^{z_m}}$ in Eq. (21) can be replaced by $Q^R_{n^{z_l},n^{z_m},f^t,IP(f^t),i,j}$, as shown below:

$$Q^R_{n^{z_l},n^{z_m},f^t,IP(f^t),i,j} = x^R_{i,f^t,n^{z_l}} . x^R_{j,IP(f^t),n^{z_m}} \tag{26}$$

$$Q^R_{n^{z_l},n^{z_m},f^t,IP(f^t),i,j} \leq x^R_{i,f^t,n^{z_l}}$$
$$\forall n^z \in N^z, R \in Req, f^t \in vnf_R, i \in I_{f^t}, j \in I_{IP(f^t)} \tag{27}$$

$$Q^R_{n^{z_l},n^{z_m},f^t,I_{IP(f^t)},i,j} \leq x^R_{j,IP(f^t),n^{z_m}}$$
$$\forall n^z \in N^z, R \in Req, f^t \in vnf_R, i \in I_{f^t}, j \in I_{IP(f^t)} \tag{28}$$

$$Q^R_{n^{z_l},n^{z_m},f^t,I_{IP(f^t)},i,j} \geq x^R_{i,f^t,n^{z_l}} + x^R_{j,IP(f^t),n^{z_m}} - 1$$
$$\forall n^z \in N^z, R \in Req, f^t \in vnf_R, i \in I_{f^t}, j \in I_{IP(f^t)} \tag{29}$$

In addition, in order to linearize the max function in processing/communication time equations in Table 7.2 for parallel sub-structure, and in Eq. (9), we replace $\max(x_1, x_2)$ with z such that:

$$\begin{aligned} z &= \max(x_1, x_2) \\ z &\geq x_1 \\ z &\geq x_2 \end{aligned} \tag{30}$$

7.4. Tabu Search-based Component Placement

In this section, we propose a Tabu Search-based Component Placement (TSCP) algorithm for the optimization problem explained in Section 7.3. The search space size is exponential in terms of the number of VNF types, VNFs instances, number of cloud/fog nodes, and number of requests. Thus, as will be seen in Section 7.5, the runtime for finding the optimal solution with CPLEX is quite long, even for small-scale scenarios. Therefore, a heuristic approach is required to perform the placements in real system scales with acceptable run times. Tabu Search meta-heuristic has been shown to be promising in terms of finding a near-optimal solution in combinatorial optimization problems (e.g., [33][34]) and VNF placement problems [124][125], and so we exploit it in our component placement algorithm.

Tabu is an iterative search process that starts exploring the search space from an initial solution and iteratively performs moves to transit from the current solution to a better one in its neighborhood until the stopping criterion is satisfied. Tabu Search uses a memory structure called Tabu-list to avoid looping during the search process, thereby preventing cycling to previously

visited solutions [126]. In the rest of this section, we explain the major elements of the Tabu Search algorithm as outlined in Algorithm 7.1.

1- Tabu starts searching with an initial placement. VNF types that communicate with IoT devices are randomly assigned to a fog node with enough capacity to process the VNF. The rest of the VNFs are assigned randomly to a cloud node with sufficient capacity Eq. (20). Note that the constraints satisfaction in the search process will be considered in the evaluation phase as will be discussed later in this section.
2- Tabu explores the neighborhood of the current placement to improve the quality of the just-identified best placement. A neighborhood is generated by applying a single move from the current placement. We define four moves as below:
   - **VNF Reassignment –** A VNF is selected randomly and moved to a node with enough capacity and minimum amount of aggregated processing time, hosting cost, and communication time/cost with its immediate predecessors and IoT/end-users (for all the requests using this VNF). Note that the aggregation is performed as in the weighting used in Eq. (16).
   - **Bulk VNF reassignment -** A node is selected randomly, and the VNFs on that node are assigned to another node with enough capacity to host the VNFs and minimum amount of aggregated processing time, hosting cost, and communication time/cost with its immediate predecessors and IoT/end-users for all VNFs.
   - **Request reassignment-** A request is selected randomly and one of its required VNFs is assigned to another instance with enough capacity to tolerate the traffic and minimum amount of aggregated processing time, hosting cost, and communication time/cost with its immediate predecessors and IoT/end-users.
   - **Bulk request reassignment -** A VNF is selected, and all its requests are assigned to another VNF instance with enough capacity and minimum amount of aggregated processing time, hosting cost, and communication time/cost with its immediate predecessors and IoT/end-users.
3- To avoid visiting the same solution several times, Tabu uses a list called Tabu list to store moves marked as Tabu. The move that generates the best neighborhood i.e., $best\_move$ is saved in $Tabu\_list$ for a specific length of time, or number of iterations i.e., i_{tabu}. Further,

a Tabu move can be released from $Tabu\_list$ if it meets the aspiration criterion, defined as the case when a better solution than the current best solution has been found.

4- In each iteration of the Tabu search process, the neighbors are evaluated in order to recognize the best solution and move towards that. We use the aggregation of the objective function as defined by Eq. (16) and the penalty function imposed due to constraints' violation to evaluate each placement. Eq. (31) indicates the evaluation function:

$$E(S_{curr}) = \begin{cases} obj(S_{curr}), & \text{If } S_{curr}\ is\ feasible \\ obj(S_{curr}) + p(S_{curr}), & otherwise \end{cases} \quad (31)$$

where $p(S_{curr})$ is the penalty function for the current placement. We have used the suggested penalty calculation in [127]. The left and right sides of the constraints (20), (21), (22), and (23) are represented with g_m for $m = 1 \dots 4$, and b_m respectively. In this regard, a constraint can be represented by $g_m(S_{curr}) < b_m$. The penalty is calculated as below:

$$p(S_{curr}) = \sum_{m=1}^{M} \varsigma_m \max(0, g_m(S_{curr}) - b_m) \quad (32)$$

ς_m is the normalization coefficient to make $p(S_{curr})$ and $obj(S_{curr})$ in the same scale.

5- The algorithm will stop when the best solution (i.e., S_{best}) does not improve for a certain number of consecutive iterations (i_{stop}).

7.5. Performance evaluation

Here we evaluate the performance of our proposed placement algorithm, the TSCP, comparing it with the optimal solution gained by CPLEX (Optimal), to the TSCP (Random Explore) where the optimization variables are changed by random moves instead of makespan/cost driven moves as discussed in Section 7.4, and finally, to a first-fit greedy placement (Greedy). Greedy iterates over the set of VNF-FGs associated with applications. For each VNF in a VNF-FG, Greedy first checks if that VNF type is already deployed in the network and if it has adequate capacity. If such a deployed VNF is found, Greedy assigns it to the request. Otherwise, Greedy instantiates a new VNF of that type on the first fitted node (from the aspect of VNF processing and communication with the immediate predecessors). In the rest of this section, we explain the experimental setup and then we present the evaluation results.

Table 7.3 Summary of simulation parameters

Parameter	Value
VNFs	
Number of VNF types	[3-27]
VNF resource requirements (vCPU)	[1- 4]
VNF processing capacity per GB	[1- 2]
VNF license cost ($)	100
Structured VNF-FGs	
Number of VNF-FG requests	[1-50]
Number of VNFs in a VNF-FG request	[3-10]
Traffic Amount (KB)	[0.1-180]
Cloud/Fog Network	
Number of nodes: cloud, fog	[4, 8], [6, 12]
Nodes capacity (vCPU): cloud, fog	8, [2-4]
Nodes cost ($/vCPU): Cloud, fog	[2.33- 4.65], [4.65- 5.82]
Nodes delay (msec/MB): cloud, fog	0.25, 25
Bandwidth cost ($/GB): cloud, fog, cloud-fog edges	0.155, [0.25-2], [10-20]
Bandwidth (Gbps): cloud, fog, cloud-fog edges	10, [0.1-1], [1, 10]
Latency (msec): cloud, fog, cloud-fog edges	[50-100], [10-50], [100-255]
IoT Devices/End-Users	
Number of IoT/end-users	[5-30]
Bandwidth cost ($/GB): IoT-cloud, IoT-fog	20, [0.05-0.25]
Bandwidth: IoT-cloud and IoT-fog	10Gbps, [250Kbps-54Mbps]
Latency (msec): IoT-cloud and IoT-fog	250, [7-20]
Location Analysis	
Fog nodes velocity	0.015
Probability that fog node is stationary	0.2
Pause Time (msec)	[10-300]

7.5.1. Experimental Setup

We synthesized 50 structured VNF-FGs according to the method proposed in [128]. This method generates structured VNF-FGs (see Figure 7.1 (b)) that respect the parameters, including the number of VNFs, graph height, the maximum number of children, and the selection to parallel ratio. The number of VNFs in a structured VNF-FG is chosen randomly between 3 and 10. The height of a structured VNF-FG is chosen randomly from {2, 4, 6, 8}. The selection to parallel ratio in the case of having a split in the graph is randomly chosen from {0, 0.2, 0.4, 0.6, 0.8, 1}. The maximum number of children is 5. Note that equal probabilities are assigned to the children of the selection sub-structures. For the sake of simplicity, we have assumed that there is no loop in the structured VNF-FG.

We have assumed a license cost of $100 for the VNF instantiation. Each VNF uses an OpenStack VM from tiny to large size, with 1 to 4 vCPUs. The size of the data transmitted in the chains is selected randomly in the range of 100 bytes to 80 KB [129].

Two cloud/fog infrastructures of 10 and 20 nodes are considered. The first infrastructure consists of 4 cloud nodes and 6 fog nodes, whilst the second infrastructure consists of 8 cloud nodes and 12 fog nodes. Note that similar scales have been used in [68] and [62] for cloud/fog infrastructures. A fog node is mobile with a probability of 0.8. In the case of mobility, the velocity is 0.015 and the pause time is chosen uniformly in the range of 10 to 300 msec. We assume the cloud nodes have 8 vCPUs and the fog nodes have a random number between 2 and 4 vCPUs. The cloud nodes communicate with each other with bandwidth 10Gbps, the fog nodes communicate with each other with a random bandwidth in the range of 100Mbps to 1Gbps, and the cloud and fog nodes communicate with each other with a random bandwidth in the range of 1Gbps to 10Gbps [129].

The cost of cloud/fog node usage is selected randomly in the range of $(2.33 to 4.65)/vCPU and $(4.65 to 5.82)/vCPU respectively [130]. The cloud communication bandwidth cost is $0.155 per GB transmission and the communications costs in the fog vary between $0.25 and $2 per GB transmission. Finally, for the cloud and fog communications, the cost is random in the range of $10 to $20 per GB transmission [125].

The processing delay on cloud and fog nodes is set to 0.25 msec and 25 msec, respectively, per Megabyte traffic processing [129]. The communication latency in the cloud, in the fog, and between the cloud and the fog are ranges within (50 to 100) msec, (10 to 50) msec, and (100 to 255) msec, respectively.

Note that the bandwidth, the cost, and the latency in communications vary randomly in the mentioned ranges depending on the fog node location involved in the communication.

The number of IoT/end-users ranges from 5 to 30 per application. The communication bandwidth between IoT devices and the cloud is 10Gbps, whilst it is in the range of 250Kbps to 54Mbps for communication with fog nodes [129]. Accordingly, the communication cost for the communication cost for the cloud and the fog are set to $20/GB and uniform in the range of $(0.05 to 0.25)/GB, respectively [131]. The communication latencies with the cloud and the fog are set to 250msec and (7 to 20) msec, respectively. Finally, we found the values of 60 and 20 appropriate for the experiments for i_{tabu} and i_{stop}, respectively. For all the experiments, we assume that the

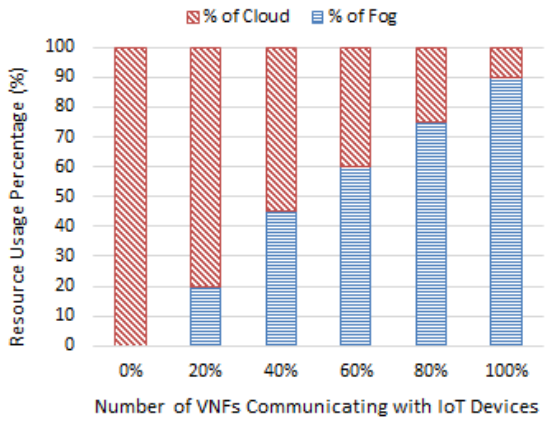


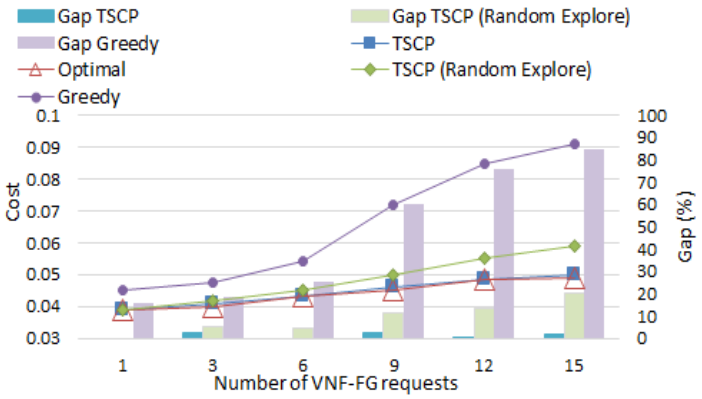
Figure 7.3 Resources usage percentage when varying the number of VNFs communicating with IoT/end-users

whole capacity of the VNFs and communication links can be used. Table 7.3 lists the parameters in the simulation.

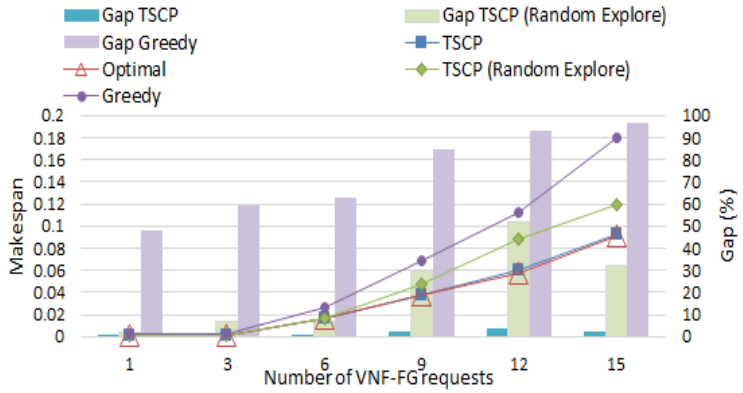
7.5.2. Evaluation Results

In the rest of this section, the average of the normalized cost, makespan, and aggregated of them for all the requests are given for 10 runs. Figure 7.2 shows the percentages of the cloud/fog resources usage in TSCP for various values of α. As it can be observed, when α increases, more components are placed in the fog to reduce the requests' makespan. In particular, in the infrastructure with 10 nodes, i.e., Figure 7.2 (a), when $\alpha = 1$, some resources are still used in the cloud due to the limited number of fog nodes or due to the fog nodes' capacity limitations (from the aspect of VNF processing and communication). On the other hand, with an infrastructure with 20 nodes and thus more available fog nodes (i.e., Figure 7.2 (b)), all the components are deployed in the fog. As α decreases, cloud resources are used more. In the extreme case, when $\alpha = 0$, all components are deployed in the cloud to minimize the cost.

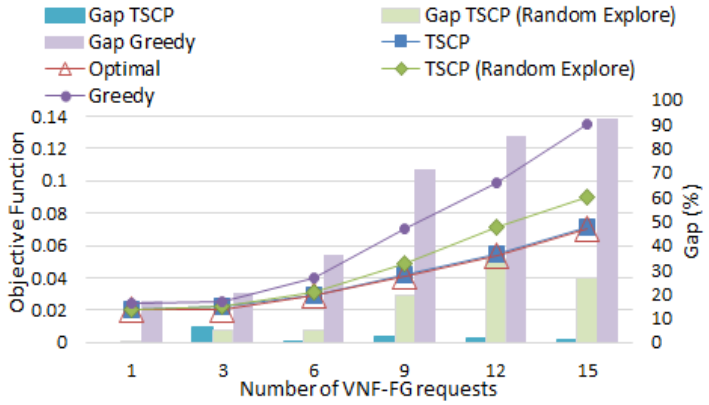
Figure 7.3 shows the resources usage percentages in TSCP for various amounts of communication with IoT/end-users. We have changed the number of the VNFs that communicate with IoT/end-users in each VNF-FG request for the case of an infrastructure with 20 nodes and $\alpha = 0.5$. As visible in Figure 7.3, when the communications with IoT/end-users increase, more fog resources are used to reduce the communication time with IoT/end-users, and accordingly, to reduce the aggregated makespan and cost.



(a)

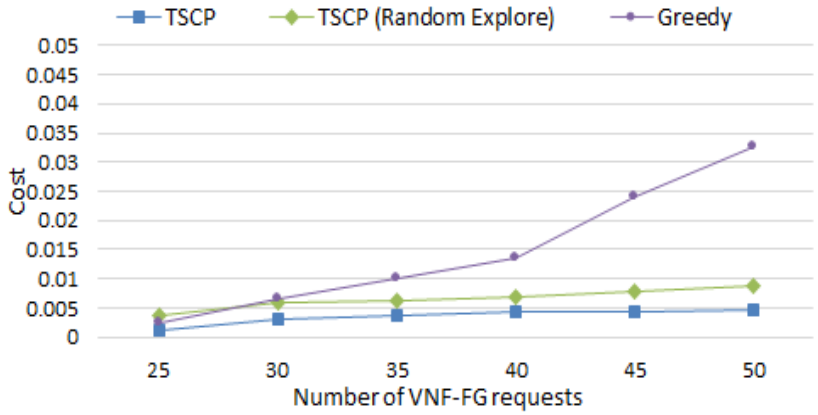


(b)

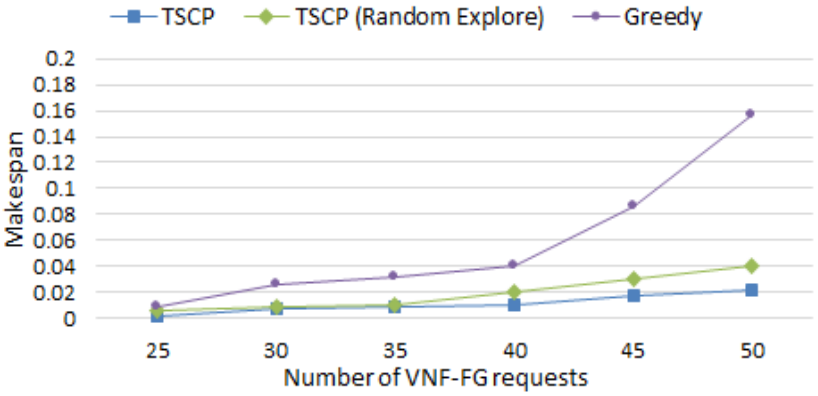


(c)

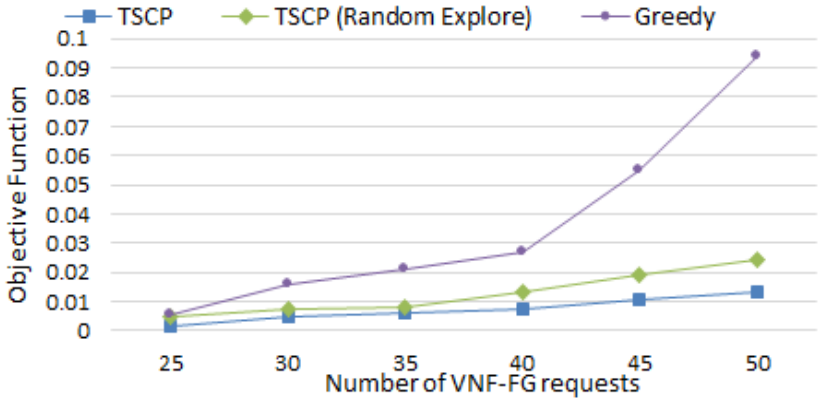
Figure 7.4 Total cost (a), makespan, (b) and aggregated weighted function of cost and makespan (c) for Optimal, TSCP, Greedy, and TSCP (Random Explore), together with the gap from optimality for TSCP, TSCP (Random Explore), and Greedy for 10 nodes and up to 15 VNF-FG requests with $\alpha = 0.5$



(a)



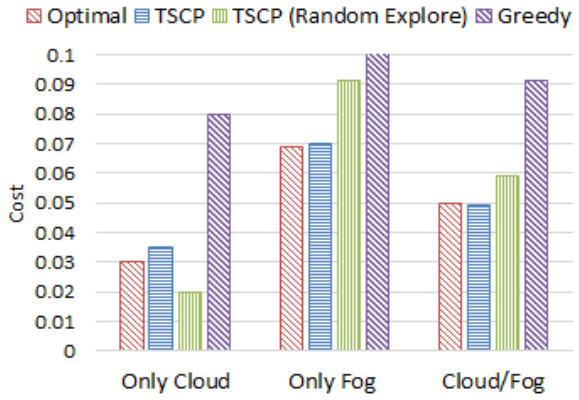
(b)



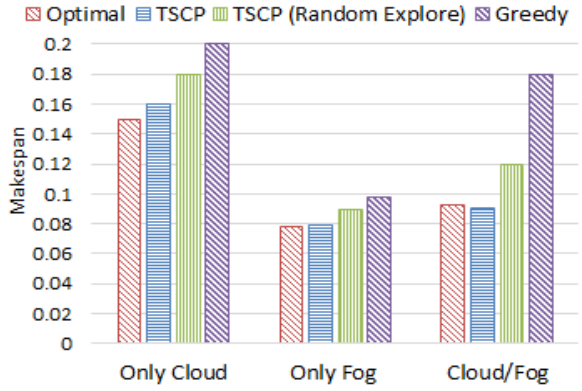
(c)

Figure 7.5 Total cost (a), makespan, (b) and aggregated weighted function of cost and makespan (c) for Optimal, TSCP, Greedy, and TSCP (Random Explore) for 20 nodes and 50 VNF-FG requests with $\alpha = 0.5$

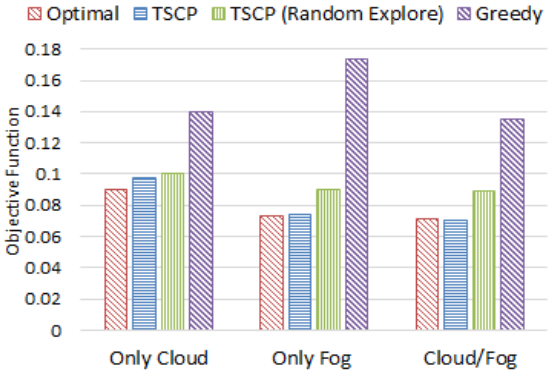
h



(a)



(b)



(c)

Figure 7.6 Total cost (a), makespan, (b) and aggregated weighted function of cost and makespan (c) for optimal, TSCP, Greedy, and TSCP (Random Explore) for 10 nodes and 15 VNF-FG requests with $\alpha = 0.5$, considering three scenarios: only cloud, only fog, cloud/fog system

$\alpha = 0.5$ for two different scales. Figure 7.4 indicates the results for an infrastructure with 10 nodes and up to 15 requests. In this figure (i.e., Figure 7.4) the average gap between the TSCP, Greedy, and the TSCP (Random Explore) algorithms with respect to the Optimal results is also demonstrated. As can be seen, the TSCP has cost, makespan, and objective functions that are very close to those of the Optimal result. Greedy shows the worst performance, as it selects the first available cloud/fog node without taking into account the time/cost of VNF execution. The TSCP outperforms TSCP (Random Explore), which demonstrates the effectiveness of the VNF execution time, hosting cost, and communication time/cost consideration in the TSCP exploration phase, as performed by the moves introduced in Section 7.4. Please note that the actual values for the makespan in Figure 7.5 (b) for points 1 and 3 are 48msec and 123msec, respectively. However, because of the normalization, in this figure, the values are close to zero.

Figure 7.5 illustrates similar results for the larger scale, i.e., infrastructure with 20 nodes and up to 50 VNF-FG requests. Note that we could not get the optimal results at this scale due to its very long run time. The better performance of TCSP compared to that of the TSCP (Random Explore) and Greedy is much more remarkable here than in the smaller-scale experiment with a smaller solution space size. While the TSCP outperforms the other methods in aggregated makespan and cost by up to 47.23% (see Figure 7.4), this value is up to 85% for larger-scale experiments. As can be observed, Greedy has the worst performance, since it does not consider the time/cost of VNF execution and communication when selecting the cloud/fog nodes. Similar to Figure 7.4, the actual makespan values for the points 25 and 30 in Figure 7.5 (b) are 1985.2msec and 2143.5msec, respectively. These values in Figure 7.5 (b) are close to zero because of the normalization. It should be noted that the actual values for the cost and makespan considering 1 to 50 requests are in the range of 300 to 2400 units of currency considering $ as a unit, and from 48msec to 3230msec, respectively.

Figure 7.6 illustrates the results for the infrastructure with 10 nodes and 15 VNF-FG requests. Different types of infrastructures are considered: when the infrastructure is provided as a cloud, when it is provided as a fog, and the hybrid case consisting of both cloud and fog. As can be observed in Figure 7.6 (a), in every method, the cost is minimized by using only the cloud, as the resources in the cloud are cheaper than those in the fog. On the other hand, as can be seen in Figure

Table 7.4 Average execution time

Experiment Parameter		Execution Time (sec)	
Number of Nodes	***Number of VNF-FGs***	***Optimal***	***Tabu Search Algorithm***
10	5	4800	0.21
20	5	5400	0.58
10	10	21600	1.32
20	10	25560	2.45
10	15	54000	5.12
20	15	> 86400	5.6
20	50	∞	57

7.6 (b), makespan is minimized by using only the fog. This is because the fog provides lower communication time than the cloud due to its proximity to IoT/end-users; a situation which leads to makespan reduction. We can also see that the best results for the aggregated weighted function of makespan and cost (i.e., Figure 7.6 (c)) are obtained when the components are placed on a hybrid cloud/fog system for all of the algorithms.

Table 7.4 shows the computational complexity of the TSCP in comparison with the Optimal solution. The TSCP clearly has a much shorter execution time than the Optimal solution. For the infrastructure with 20 nodes and 5 requests, the execution time of the Optimal solution exceeds 1 hour. The execution time increases as the scale of the infrastructure or the number of requests increases. For example, when the number of requests increases to 15, it took one full day to find the optimal placement, while the TSCP could find a near-optimal placement in less than 6 seconds.

7.6. Conclusion

This chapter studies the application component placement problem in NFV-based hybrid cloud/fog systems with mobile fog nodes. The applications' components are implemented as VNFs. A structured VNF-FGs containing sub-structures such as sequence, parallel, selection, and loop is established to model the execution sequence of the components. The mobility of fog nodes is modeled via the random waypoint mobility model. Based on the stationary analysis of the random waypoint model, the expected execution time and cost of the components and sub-structures are calculated. The calculations of the sub-structures are aggregated to calculate the expected application makespan and cost. The placement problem is modeled as an ILP optimization that minimizes the aggregated makespan and cost for all requests. A Tabu-based

algorithm is proposed to solve the problem in large scales of cloud/fog infrastructure and for a high number of requests. The simulation results show that the proposed algorithm operates at near-optimal for small scales and improves the makespan, the cost, and the aggregated of them for larger scales. Our studies also show that the greater the communication between the application components and the IoT/end-users, the more fog resources are used to reduce the makespan.

Chapter 8

8. Conclusion and Future Work

Provisioning large-scale IoT-based disaster management systems face several challenges such as the dynamic formation of an optimal coalition of IoT devices, the heterogeneity of IoT devices, and the QoS of these applications. This thesis proposed softwarization approaches to address these challenges. We approached these challenges in two complementary ways; architectural and algorithmic. For the architectural contributions, in chapter 3 we proposed a cloud-based architecture that allows selecting the optimal group of robots for search and rescue tasks of disaster management applications. It discussed the architectural modules and the interfaces that cover the IaaS aspects. This contribution allows publishing and discovering robots belonging to different infrastructures and proposed a well-defined language that allows describing robots' capabilities based on existing standards. The proposed architecture enables flexible, elastic, and cost-efficient use of robots benefiting cloud advantages such as virtualization and scalability.

To enable the interoperability across IoT devices and applications, we proposed IoT gateway architecture based on NFV and SDN. Both centralized (i.e., Chapter 4) and distributed (i.e., Chapter 5) approaches were considered. For the centralized approach, the elastic scalability of the architecture is considered and for the distributed approach, co-locating the gateway functionalities with the IoT devices is considered. Reusing already deployed gateways and handling the traffic and chaining between the gateway functions dynamically are also considered in the distributed

approach. For both approaches, a high-level description of the proposed architecture that is composed of two planes is provided, and a detailed description of each plane with its corresponding interfaces and procedures is presented. The proposed architectures enable on-the-fly provisioning of IoT gateways. Updating existing gateways are also among the benefits of the proposed architectures.

To ensure that the optimal coalition of robots is selected dynamically with the required capabilities for resource efficiency, we proposed a coalition formation algorithm for multi-robot task allocation in Chapter 6. The proposed algorithm takes into consideration the location constraints regarding the capability distribution of the robots. It consists of a filtering method, QMOPSO approach, and a ranking method. The proposed algorithm improves the solution and has a significantly reduces the processing time.

To meet the QoS requirements of disaster management application, we proposed an application component placement algorithm over hybrid cloud/fog NFVIs in Chapter 7. Both stationary and mobile fog nodes were considered. The proposed algorithm considered minimizing the aggregated weighted functions of applications makespan and cost. It also considered non-deterministic VNF-FG graphs by assigning probabilities to selection sub-structures and mean numbers of iterations to loop sub-structures. The mobility of fog nodes was modeled using the RWP model. Based on the stationary distribution of fog nodes' location, the expected makespan and cost for the sub-structures: sequence, parallel, selection, and loop were calculated. The calculations were aggregated in order to obtain the application's makespan and execution cost. The problem was formulated as an ILP problem and a Tabu Search-based Component Placement (TSCP) algorithm was proposed to find a sub-optimal solution in feasible time. The simulation results showed that the proposed algorithm operated at near-optimal for small scales and improves the makespan, the cost, and the aggregated of them for larger scales.

8.1. Future Work

This thesis presented significant contributions towards the softwarization of large-scale IoT-based disaster management systems. Yet, there exist several research directions for the future.

8.1.1. Node-level Virtualization

In the future, we would like to incorporate node-level virtualization in the proposed cloud-based architecture for IoT application provisioning in Chapter 3. Node-level virtualization can be achieved by sequential (one-by-one) or simultaneous execution (by context switching/multi-threading) of application tasks on a sensor node. According to [132] node-level virtualization of sensors can be realized by either i) a capable operating system like Contiki, ii) using a middleware like Agilla or iii) by using a virtual machine like Squawk that directly runs over the sensors hardware. It would be interesting to investigate the node-level virtualization approaches for robots and incorporate them to the proposed architecture.

8.1.2. Resource Allocation Algorithms

Chapter 4 and Chapter 5 propose IoT gateway architecture. The proposed architecture relies on a simple dynamic resource allocation algorithm to meet the growing demand of applications. It is based on the resource utilization of the VMs (i.e., CPU) and on horizontal scaling. However, there is a need to design appropriate resource allocation algorithms in the specific context of VNFs. Such algorithms should enable vertical scaling – i.e., increasing the resources of a VNF instance (e.g., CPU, memory) and/or horizontal scaling – i.e., increasing the number of VNF instances that serve an application. A potential starting point can be considered the resource allocation algorithms that exist today for VMs (e.g., [11] and [12]).

8.1.3. Application Component Placement

The approach proposed in Chapter 7 for application component placement focused on static (offline) placement strategies. However, such mode of placement may not be adequate for dynamic systems where applications arrive at the system dynamically and fog nodes move. In real-world scenarios, there could be a variety of motivations for modifying the placement of the application components. For instance, a fog node communicating with an IoT devices might move far from the device. In such cases, the continuity of the offered services needs to be ensured despite this movements. An important study can be to consider the online placement of application components that include migration techniques such as [133] and [134] to handle the mobility of fog nodes.

8.1.4. Architecture for Hybrid Cloud/Fog System

As fog computing matures, there is a need for hybrid cloud/fog architectures to provision application with components spanning cloud and fog. Existing PaaSs do not enable this [135]. In the recent survey on fog computing [13], we defined several requirements and research directions for cloud and fog integrations. These requirements can be used as a starting point to design such architectures. For instance, the proposed architectures should enable the migration of application components during runtime from one hosting node to another (e.g., e.g. from cloud to fog and vice versa or from fog to another fog).

Bibliography

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutor.*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[2] "World Trade Center 911 Disaster « Center for Robot-Assisted Search and Rescue (CRASAR) at Texas A&M University." .

[3] H. Kitano *et al.*, "RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research," in *1999 IEEE International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings*, 1999, vol. 6, pp. 739–743 vol.6.

[4] D. Pop, G. Iuhasz, C. Craciun, and S. Panica, "Support Services for Applications Execution in Multi-clouds Environments," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 2016, pp. 343–348.

[5] B. D. Martino, "Applications Portability and Services Interoperability among Multiple Clouds," *IEEE Cloud Comput.*, vol. 1, no. 1, pp. 74–77, May 2014.

[6] C. Mouradian, S. Yangui, and R. H. Glitho, "Robots as-a-service in cloud computing: Search and rescue in large-scale disasters case study," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018, pp. 1–7.

[7] C. Mouradian, J. Sahoo, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A coalition formation algorithm for Multi-Robot Task Allocation in large-scale natural disasters," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 1909–1914.

[8] C. Mouradian, T. Saha, J. Sahoo, R. Glitho, M. Morrow, and P. Polakos, "NFV based gateways for virtualized wireless sensor networks: A case study," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1883–1888.

[9] C. Mouradian *et al.*, "Network functions virtualization architecture for gateways for virtualized wireless sensor and actuator networks," *IEEE Netw.*, vol. 30, no. 3, pp. 72–80, May 2016.

[10] C. Mouradian, N. T. Jahromi, and R. H. Glitho, "NFV and SDN - based Distributed IoT Gateway for Large-Scale Disaster Management," *IEEE Internet Things J.*, pp. 1–1, 2018.

[11] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight Resource Scaling for Cloud Applications," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 644–651.

[12] "AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting," *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/ec2/. [Accessed: 19-Aug-2015].

[13] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-art and Research Challenges," *IEEE Commun. Surv. Tutor.*, vol. PP, no. 99, pp. 1–1, 2017.

[14] C. Mouradian, S. Kianpisheh, and R. H. Glitho, "Application Component Placement in NFV-based Hybrid Cloud/Fog Systems," *ArXiv180604578 Cs*, May 2018.

[15] C. Mouradian, S. Kianpisheh, M. Abu-Lebdeh, F. Ebrahimnezhad, N. Tahghigh Jahromi, and R. H. Glitho, "Application Component Placement in NFV-based Hybrid Cloud/Fog Systems with Mobile Fog Nodes," *IEEE J. Sel. Areas Commun. - Submitt.*

[16] C. Bettstetter, G. Resta, and P. Santi, "The node distribution of the random waypoint mobility model for wireless ad hoc networks," *IEEE Trans. Mob. Comput.*, vol. 2, no. 3, pp. 257–269, Jul. 2003.

[17] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[18] Z. Alazawi, S. Altowaijri, R. Mehmood, and M. B. Abdljabar, "Intelligent disaster management system based on cloud-enabled vehicular networks," in *2011 11th International Conference on ITS Telecommunications*, 2011, pp. 361–368.

[19] M. Erdelj and E. Natalizio, "UAV-assisted disaster management: Applications and open issues," in *2016 International Conference on Computing, Networking and Communications (ICNC)*, 2016, pp. 1–5.

[20] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Commun. Surv. Tutor.*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[21] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network Slicing amp;#x0026; Softwarization: A Survey on Principles, Enabling Technologies amp;#x0026; Solutions," *IEEE Commun. Surv. Tutor.*, pp. 1–1, 2018.

[22] “Network Functions Virtualisation (NFV); Architectural Framework. Availble: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.” .
[23] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, “A Survey on Software-Defined Networking,” *IEEE Commun. Surv. Tutor.*, vol. 17, no. 1, pp. 27–51, Firstquarter 2015.
[24] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud Computing: State-of-the-art and Research Challenges,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.
[25] C. Mouradian, F. Z. Errounda, F. Belqasmi, and R. Glitho, “An Infrastructure for Robotic Applications as Cloud Computing Services,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 377–382.
[26] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, pp. 13–16.
[27] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog Computing: A Platform for Internet of Things and Analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, Eds. Springer International Publishing, 2014, pp. 169–186.
[28] M. Yannuzzi, R. Milito, R. Serral-Gracia, D. Montero, and M. Nemirovsky, “Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing,” in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2014, pp. 325–329.
[29] “Earthquake Early Warning.” [Online]. Available: https://earthquake.usgs.gov/research/earlywarning/nextsteps.php. [Accessed: 09-Jan-2017].
[30] “Zizmos.” [Online]. Available: https://www.zizmos.com/. [Accessed: 18-Oct-2018].
[31] “Grillo.” [Online]. Available: https://grillo.io/. [Accessed: 18-Oct-2018].
[32] “draft-jennings-senml-07 - Media Types for Sensor Markup Language (SENML).” [Online]. Available: https://tools.ietf.org/html/draft-jennings-senml-07. [Accessed: 03-May-2015].
[33] C. Mouradian, S. Yangui, and R. H. Glitho, “Robots as-a-Service in Cloud Computing: Search and Rescue in Large-scale Disasters Case Study,” *ArXiv171004919 Cs*, Oct. 2017.
[34] S. K. Datta, C. Bonnet, and N. Nikaein, “An IoT gateway centric architecture to provide novel M2M services,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 514–519.
[35] C. Schlenoff and E. Messina, “A Robot Ontology for Urban Search and Rescue,” in *Proceedings of the 2005 ACM Workshop on Research in Knowledge Representation for Autonomous Systems*, New York, NY, USA, 2005, pp. 27–34.
[36] R. Chatterjee and F. Matsuni, “Robot description ontology and disaster scene description ontology: analysis of necessity and scope in rescue infrastructure context - Advanced Robotics - Volume 19, No 8, pp.839-859.” [Online]. Available: http://www.tandfonline.com/doi/abs/10.1163/1568553055011528. [Accessed: 03-May-2015].
[37] Y. Chen, Z. Du, and M. García-Acosta, “Robot as a Service in Cloud Computing,” in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)*, 2010, pp. 151–158.
[38] Z. Du, W. Yang, Y. Chen, X. Sun, X. Wang, and C. Xu, “Design of a Robot Cloud Center,” in *2011 10th International Symposium on Autonomous Decentralized Systems (ISADS)*, 2011, pp. 269–275.
[39] L. Turnbull and B. Samanta, “Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure,” in *2013 Proceedings of IEEE Southeastcon*, 2013, pp. 1–4.
[40] B. Liu, Y. Chen, E. Blasch, K. Pham, D. Shen, and G. Chen, “A Holistic Cloud-Enabled Robotics System for Real-Time Video Tracking Application,” in *Future Information Technology*, J. J. (Jong H. Park, I. Stojmenovic, M. Choi, and F. Xhafa, Eds. Springer Berlin Heidelberg, 2014, pp. 455–468.
[41] P. Merle, C. Gourdin, and N. Mitton, “Mobile Cloud Robotics as a Service with OCCIware,” in *2017 IEEE International Congress on Internet of Things (ICIOT)*, 2017, pp. 50–57.
[42] G. Mohanarajah, D. Hunziker, R. D’Andrea, and M. Waibel, “Rapyuta: A Cloud Robotics Platform,” *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 481–493, Apr. 2015.
[43] S. K. Datta and C. Bonnet, “Smart M2M Gateway Based Architecture for M2M Device and Endpoint Management,” in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, 2014, pp. 61–68.
[44] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, “IOT Gateway: BridgingWireless Sensor Networks into Internet of Things,” in *2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2010, pp. 347–352.

[45] S. Guoqiang, C. Yanming, Z. Chao, and Z. Yanxu, “Design and Implementation of a Smart IoT Gateway,” in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, 2013, pp. 720–723.
[46] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, “A SDN-based architecture for horizontal Internet of Things services,” in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–7.
[47] M. Ojo, D. Adami, and S. Giordano, “A SDN-IoT Architecture with NFV Implementation,” in *2016 IEEE Globecom Workshops (GC Wkshps)*, 2016, pp. 1–6.
[48] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “Edge computing enabling the Internet of Things,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 603–608.
[49] H.-Y. Liu and J.-F. Chen, “Multi-Robot Cooperation Coalition Formation Based on Genetic Algorithm,” in *2006 International Conference on Machine Learning and Cybernetics*, 2006, pp. 85–88.
[50] L. Vig and J. A. Adams, “Multi-Robot Coalition Formation,” *IEEE Trans. Robot.*, vol. 22, no. 4, pp. 637–649, 2006.
[51] M. Agarwal, N. Kumar, and L. Vig, “Non-Additive Multi-Objective Robot Coalition Formation,” *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3736–3747, Jun. 2014.
[52] T. C. Service, S. D. Sen, and J. A. Adams, “A simultaneous descending auction for task allocation,” in *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2014, pp. 379–384.
[53] J. G. Manathara, P. B. Sujit, and R. W. Beard, “Multiple UAV Coalitions for a Search and Prosecute Mission,” *J. Intell. Robot. Syst.*, vol. 62, no. 1, pp. 125–158, Apr. 2011.
[54] B. Qian and H. H. Cheng, “A mobile agent-based coalition formation system for multi-robot systems,” in *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, 2016, pp. 1–6.
[55] P. M and G. R. Suresh, “Coalition formation and Task Allocation of multiple autonomous robots,” in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, 2015, pp. 1–5.
[56] A. Rauniyar and P. K. Muhuri, “Multi-robot coalition formation problem: Task allocation with adaptive immigrants based genetic algorithms,” in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 000137–000142.
[57] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, “Quality of Experience (QoE)-aware placement of applications in Fog computing environments,” *J. Parallel Distrib. Comput.*, Mar. 2018.
[58] R. Deng, R. Lu, C. Lai, and T. H. Luan, “Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing,” in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 3909–3914.
[59] L. Yin, J. Luo, and H. Luo, “Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacture,” *IEEE Trans. Ind. Inform.*, pp. 1–1, 2018.
[60] X.-Q. Pham and E.-N. Huh, “Towards task scheduling in a cloud-fog computing system,” in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016, pp. 1–4.
[61] “Swati Agarwal, Shashank Yadav, Arun Kumar Yadav,"An Efficient Architecture and Algorithm for Resource Provisioning in Fog Computing", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.8, No.1, pp.48-61, 2016.”
[62] M. Taneja and A. Davy, “Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm,” in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1222–1228.
[63] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, “Towards QoS-Aware Fog Service Placement,” in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 89–96.
[64] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen, “Help your mobile applications with fog computing,” in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, 2015, pp. 1–6.
[65] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, “Mobility-Aware Application Scheduling in Fog Computing,” *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar. 2017.
[66] F. Nawab, D. Agrawal, and A. E. Abbadi, “Nomadic Datacenters at the Network Edge: Data Management Challenges for the Cloud with Mobile Infrastructure.” OpenProceedings.org, 2018.
[67] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, “Fog Following Me: Latency and Quality Balanced Task Allocation in Vehicular Fog Computing,” in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2018, pp. 1–9.
[68] M. G. R. Alam, Y. K. Tun, and C. S. Hong, “Multi-agent and reinforcement learning based code offloading in mobile fog,” in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 285–290.

[69] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 418–423.
[70] W. Fang, M. Zeng, X. Liu, W. Lu, and Z. Zhu, "Joint Spectrum and IT Resource Allocation for Efficient VNF Service Chaining in Inter-Datacenter Elastic Optical Networks," *IEEE Commun. Lett.*, vol. 20, no. 8, pp. 1539–1542, Aug. 2016.
[71] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015, pp. 255–260.
[72] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 98–106.
[73] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-Assisted NFV Service Chain Deployment Based on Affiliation-Aware vNF Placement," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.
[74] M. Mechtri, C. Ghribi, and D. Zeghlache, "A Scalable Algorithm for the Placement of Service Function Chains," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.
[75] "OpenFog Consortium, 'OpenFog Reference Architecture for Fog Computing', Tech. Rep., February 2017." .
[76] "R. Fielding, 'Architectural Styles and the Design of Network-based Software Architectures', Ph.D. dissertation, University of California, Irvine, 2000." .
[77] M. Day, J. Rosenberg, and H. Sugano, "A Model for Presence and Instant Messaging." RFC 2778, Feb-2778.
[78] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," *IEEE Commun. Mag.*, vol. 48, no. 12, pp. 92–100, Dec. 2010.
[79] "Telecommunications Infrastructure In Disasters: Preparing Cities for Crisis Communications, Anthony M. Townsend, Mitchell L. Moss, April 2005." .
[80] "City of New York, Department of Information Technology and Telecommunications. December 2002. 'Request for Proposals for Franchises Authorizing Construction and Provision of Lateral Ducts and Related Facilities to House Telecommunications Fiber Links Transmitting Local High-Capacity Telecommunications Services Between Mainline Systems and Building Entrances.' [http://www.nyc.gov/htmłdoitt/downloads/pdf/lateral_ducts_rfp_122002.pdf]." .
[81] CISCO, "Tactical Communications Using Cisco IPICS 2.0 Across High Latency Networks - Cisco," 2016. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/physical-security/ip-interoperability-collaboration-system/prod_white_paper0900aecd805f7726.html. [Accessed: 23-Jun-2016].
[82] "Emergency Mobile Radio Network based on Software-Defined Radio, Takeuchi Takashi, Honda Atsushi, Watanabe Hideki, Eto Yasutaka, Fujita Yoshitaka, Yaga Manabu." .
[83] G. Baldini, T. Sturman, A. Dalode, A. Kropp, and C. Sacchi, "An emergency communication system based on software-defined radio," *EURASIP J. Wirel. Commun. Netw.*, vol. 2014, no. 1, pp. 1–16, Oct. 2014.
[84] R. Murphy, "Drones Save Lives in Disasters, When They're Allowed to Fly," Sep-2015. [Online]. Available: http://www.space.com/30555-beginning-with-katrina-drones-save-lives-in-disasters.html. [Accessed: 08-Jun-2016].
[85] S. Miyama, M. Imai, and Y. Anzai, "Rescue robot under disaster situation: position acquisition with Omni-directional Sensor," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, 2003, vol. 4, pp. 3132–3137 vol.3.
[86] H. Sugiyama, T. Tsujioka, and M. Murata, "Integrated operations of multi-robot rescue system with ad hoc networking," in *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009*, 2009, pp. 535–539.
[87] R. Guerraoui and A. Schiper, "Fault-tolerance by replication in distributed systems," in *Reliable Software Technologies — Ada-Europe '96*, 1996, pp. 38–57.
[88] C. Amza, A. L. Cox, and W. Zwaenepoel, "Data replication strategies for fault tolerance and availability on commodity clusters," in *Proceedings International Conference on Dependable Systems and Networks, 2000. DSN 2000*, 2000, pp. 459–467.
[89] "LEGO Mindstorms." [Online]. Available: http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com.
[90] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: early architecture and research perspectives," *IEEE Netw.*, vol. 29, no. 3, pp. 104–112, May 2015.
[91] S. K. Datta, C. Bonnet, and N. Nikaein, "An IoT gateway centric architecture to provide novel M2M services," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 514–519.

[92] S. K. Datta, C. Bonnet, and N. Nikaein, “CCT: Connect and Control Things: A novel mobile application to manage M2M devices and endpoints,” in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6.
[93] M. Scholler, M. Stiemerling, A. Ripke, and R. Bless, “Resilient deployment of virtual network functions,” in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2013, pp. 208–214.
[94] X. Ge *et al.*, “OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, New York, NY, USA, 2014, pp. 353–354.
[95] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Commun. Surv. Tutor.*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.
[96] Y. Bai, W. Du, Z. Ma, C. Shen, Y. Zhou, and B. Chen, “Emergency communication system by heterogeneous wireless networking,” in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, 2010, pp. 488–492.
[97] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, “Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives,” *Appear IEEE Netw. Mag.*, Jan. 2015.
[98] “GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration - gs_NFV-MAN001v010101p.pdf.” .
[99] N. T. Jahromi *et al.*, “NFV and SDN-based cost-efficient and agile value-added video services provisioning in content delivery networks,” in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 671–677.
[100] “ETSI GS NFV-EVE 005 V1.1.1 - gs_NFV-EVE005v010101p.pdf.” .
[101] L. Sidki, Y. Ben-Shimol, and A. Sadovski, “Fault tolerant mechanisms for SDN controllers,” in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 173–178.
[102] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, “A Fault-Tolerant and Consistent SDN Controller,” in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.
[103] “Floodlight OpenFlow Controller -,” *Project Floodlight*. [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed: 09-Jun-2018].
[104] G. Xilouris *et al.*, “T-NOVA: A marketplace for virtualized network functions,” in *2014 European Conference on Networks and Communications (EuCNC)*, 2014, pp. 1–5.
[105] F. Bannour, S. Souihi, and A. Mellouk, “Distributed SDN Control: Survey, Taxonomy, and Challenges,” *IEEE Commun. Surv. Tutor.*, vol. 20, no. 1, pp. 333–354, Firstquarter 2018.
[106] L. Dolberg, J. François, S. R. Chowdhury, R. Ahmed, R. Boutaba, and T. Engel, “A generic framework to support application-level flow management in software-defined networks,” in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 121–125.
[107] F. Belqasmi, R. Glitho, and C. Fu, “RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey,” *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 66–73, 2011.
[108] M. Kwon and S. Fahmy, “Synergy: an overlay internetworking architecture,” in *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, 2005, pp. 401–406.
[109] A. Datta and K. Aberer, “The Challenges of Merging Two Similar Structured Overlays: A Tale of Two Networks,” in *Self-Organizing Systems*, Springer, Berlin, Heidelberg, 2006, pp. 7–22.
[110] T. M. Shafaat, A. Ghodsi, and S. Haridi, “Handling Network Partitions and Mergers in Structured Overlay Networks,” in *Seventh IEEE International Conference on Peer-to-Peer Computing (P2P 2007)*, 2007, pp. 132–139.
[111] “White Paper: DPI & Traffic Analysis in a Virtualizing World - Heavy-Reading_Qosmos_DPI-SDN-NFV_White-Paper_Jan2014.pdf.” .
[112] J. G. Herrera and J. F. Botero, “Resource Allocation in NFV: A Comprehensive Survey,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.
[113] “NFV, GS. ‘001: Network Functions Virtualisation (NFV); Use Cases, V 1.2. 1.’ ETSI.” May-2017.
[114] M. Hormati, F. Khendek, R. Glitho, and F. Belqasmi, “Differentiated QoS for overlay-based disaster response systems,” in *2014 IEEE International Conference on Communications Workshops (ICC)*, 2014, pp. 225–230.
[115] B. P. Gerkey and M. J. Matarić, “A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems,” *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.

[116] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *, 1997 IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation*, 1997, vol. 5, pp. 4104–4108 vol.5.
[117] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.
[118] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 56, no. 4, pp. 1309–1318, May 2009.
[119] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer, 2005.
[120] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation Offloading for Service Workflow in Mobile Cloud Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.
[121] L. Qu, C. Assi, and K. Shaban, "Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.
[122] D. Habet, "Tabu Search to Solve Real-Life Combinatorial Optimization Problems: A Case of Study," in *Foundations of Computational Intelligence Volume 3: Global Optimization*, A. Abraham, A.-E. Hassanien, P. Siarry, and A. Engelbrecht, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 129–151.
[123] R. Battiti and G. Tecchiolli, "The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization," *Ann. Oper. Res.*, vol. 63, no. 2, pp. 151–188, Apr. 1996.
[124] W. Wang, P. Hong, D. Lee, J. Pei, and L. Bo, "Virtual network forwarding graph embedding based on Tabu Search," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, 2017, pp. 1–6.
[125] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. W. Tchouati, "On the Placement of VNF Managers in Large-Scale and Distributed NFV Systems," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 875–889, Dec. 2017.
[126] "Tabu Search—Part I | ORSA Journal on Computing." [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/ijoc.1.3.190. [Accessed: 16-Aug-2018].
[127] A. F. Kuri-Morales and J. Gutiérrez-García, "Penalty Function Methods for Constrained Optimization with Genetic Algorithms: A Statistical Analysis," in *MICAI 2002: Advances in Artificial Intelligence*, 2002, pp. 108–117.
[128] S. Kianpisheh and N. Moghadam Charkari, "A grid workflow Quality-of-Service estimation based on resource availability prediction," *J. Supercomput.*, vol. 67, no. 2, pp. 496–527, Feb. 2014.
[129] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On Reducing IoT Service Delay via Fog Offloading," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 998–1010, Apr. 2018.
[130] A. Brogi, S. Forti, and A. Ibrahim, "Deploying Fog Applications: How Much Does It Cost, By the Way?:," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, Funchal, Madeira, Portugal, 2018, pp. 68–77.
[131] Haider and Faisal, "On the Planning and Design Problem of Fog Networks," Text, Carleton University, 2018.
[132] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Commun. Surv. Tutor.*, vol. 18, no. 1, pp. 553–576, Firstquarter 2016.
[133] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic Service Migration in Mobile Edge-Clouds," *ArXiv150605261 Cs Math*, Jun. 2015.
[134] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," *IEEE Trans. Serv. Comput.*, pp. 1–1, 2018.
[135] S. Yangui *et al.*, "A platform as-a-service for hybrid cloud/fog environments," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2016, pp. 1–7.