SIP servlets-based service provisioning in MANETs

Slimane Bah

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

January 2010

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Slimane Bah**

Entitled: **SIP servlets-based service provisioning in MANETs**

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

DOCTOR OF PHILOSOPHY (Electrical and Computer 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. Fariborz Haghighat	Chair
Dr. Michel Kadoch	External Examiner
Dr. Juergen Rilling	External to Program
Dr. Mourad Debbabi	Examiner
Dr. Ferhat Khendek	Examiner
Dr. Rachida Dssouli	Supervisor
Dr. Roch Glitho	Supervisor

Approved by Dr. William E. Lynch
Chair, Department of Electrical and Computer Engineering

Dr. Robin Drew
Dean, Faculty of Engineering and Computer Science

ABSTRACT

SIP servlets-based service provisioning in MANETs

Slimane Bah, Ph.D.
Concordia University, 2010

Mobile Ad-hoc NETworks (MANETs) are a part of the fourth generation networks vision. They are new wireless networks having transient mobile nodes with no need for a pre-installed infrastructure. They are of utmost interest for the future networks owing to their flexibility, effortlessness of deployment and related low cost. They come in two flavours: standalone MANETs and integrated with the conventional 3G network.

Providing value-added services is the core concept of several paradigms and has been extensively studied in legacy network. However, providing such services in MANETs is a challenging process. Indeed, MANETs are known for their heterogeneous devices, limited resources, dynamic topology and frequent disconnections/connections. New SIP based solutions for signalling and media handling in these networks are emerging. Furthermore, SIP is the primary protocol for 3G networks. Therefore, SIP servlets become a promising paradigm for service provisioning in MANETs.

This thesis addresses the service provisioning aspects in both standalone MANETs and integrated 3G/MANETs. The SIP servlets framework is considered as the starting point while Multihop Cellular Networks (MCNs), the widely studied networks, are used as an example of integrated 3G/MANETs.

Background information is provided, architectures requirements are derived and related work is reviewed. A novel business model is proposed for service provision in standalone

MANETs. The business model defines the business roles and the relationship and interfaces between them. We also propose a service invocation and execution architecture implementing the business model. The solution is based on overlay network and a distribution scheme of the SIP servlets engine. The overlay network enables self-organization and self-recovery to take into account MANETs characteristics. As for the integrated 3G/MANETs we propose high level architectural alternatives for service provisioning in MCNs. We identify the most interesting alternatives from the network operator point of view and proposed a detailed and concrete architecture for the promising alternative. Overall architecture, functional entities and procedures are presented. During this work, we built prototypes as proof-of-concept and made preliminary performance measurements, used SPIN as protocol validation tool and adopted OPNET for simulation. The results show that we can provide services in MANETs as we do in conventional networks with reasonable performance.

ACKNOWLEDGEMENTS

I would like first and for most to express my sincere gratitude to my supervisors Dr. Rachida Dssouli and Dr. Roch Glitho for their support, patience, guidance and motivation. I learned a lot from working with them. I warmly thank Dr. Dssouli for her continual encouragements, valuable advices and kindness. I sincerely thank Dr. Glitho for his assistance, help, pertinent suggestions and his availability. This work was possible thanks to the experience and expertise of my great supervisors.

I would like to extend my thanks to the members of my examining committee for reviewing my work and for their valuable comments during all my thesis phases.

Special thanks to Concordia University and Ericsson Canada for their financial support. I thank Ericsson for giving me the opportunity to work in its Research and Development department.

I am also grateful to all my colleagues at the Telecommunication Service Engineering laboratory for their help, discussions and experiences we have shared. It is a pleasure to work with you.

At last but not least, I would like to thank my family: I am greatly thankful to my parents who unconditionally support me. My regards are addressed to my parents-in-law for their encouragements. I would like also to express my sincere gratitude to my wife Lamia for all the sacrifices she has made.

DEDICATION

To my parents

To my wife Lamia

To my son Aymane

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

1G:	First Generation Wireless System
2G:	Second Generation Wireless System
3G:	Third Generation Wireless System
4G:	Forth Generation Wireless System
3GPP	Third Generation Partnership Project
3GPP2	Third Generation Partnership Project version 2
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
API	Application Programming Interface
ARS	Ad hoc Relay Station
AS	Application Server
B2BUA	Back to Back User Agent
BAN	Body Area Network
BS	Base Station
CAMEL	Customized Applications for Mobile network Enhanced Logic
CGI	Common Gateway Interface
CIB	Context Information Base
CORBA	Common Object Request Broker Architecture
CP	Capabilities Provider
CSCF	Call Session Control Function
DARPA	Defense Advanced Research Projects Agency (United States)
E-AS	Extended Application Server
EEP	Execution Environment Provider
EESPP	Execution Environment Sub Part Provider
E-HSS	Extended Home Subscriber Server
EMS	Enhanced Messaging Service
E-SSE	Extended SIP Servlets Engine
EU	End User
FTP	File Transfer Protocol
GGSN	Gateway GPRS support Node

GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HSS	Home Subscriber Server
HTTP	HyperText Transfer Protocol
iCAR	Integrated Cellular and Ad Hoc Relaying Systems
I-CSCF	Interrogating-Call Session Control Function
IETF	Internet Engineering Task Force
IETF	Internet Engineering Task Force
IMP	IP Multimedia Subsystem
IMS	IP Multimedia Subsystem
IM-SSF	IP Multimedia Service Switching Function
IN	Intelligent Networks
IT	Information technology
ITU-T	International Telecommunication Union – Telecommunication sector
JAIN	Java APIs for Integrated Networks
JSLEE	JAIN Service Logic Execution Environment
LIME	Linda In a Mobile Environment
LSD	Lightweight Service Discovery
MAC	Media Access Control (address)
MANET	Mobile Ad-hoc NETworks
MCN	Multihop Cellular Network
MGCF	Media Gateway Control Function
MGW	Media Gateway
MH	Mobile Host
MRF	Media Resource Function
NGN	Next Generation Network
OMG	Object Management Group
OPNET	OPtimized Network Engineering Tools
OSA	Open Service Access
OSA-SCS	Open Service Access-Service Capability Server
OSLR	Optimized Link State Routing

PAN	Personal Area Network
P-CSCF	Proxy-Call Session Control Function
PDA	Personal Digitial Assistant
PDP	Pervasive Discovery Protocol
PRNET	Packet Radio Network
PROMELA	PROtocol/PROcess MEta LAnguage
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RNC	Radio Network Controller
SCF	Service Capability Feature
SCNs	Single-hop Cellular Networks
SCS	Service Capability Server
S-CSCF	Serving-Call Session Control Function
SE	Servlets Engine
SGSN	Serving GPRS Support Node
SGW	Service Gateway
SIB	Service Independent building Blocks
SIP	Session Initiation Protocol
SIP	Session Initiation Protocol
SLEE	Service Logic Execution Environment
SLP	Service Location Protocol
SMS	Short Messaging Service
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Provider
SPIN	Simple Promela INterpreter
SSE	SIP Servlets Engine
SSEP	SIP Servlets Engine Provider
SSP	SIP Servlets provider.
TCP	Transmission Control Protocol

TINA	Telecommunication Information Network Architecture
TINA-C	Telecommunication Information Network Architecture Consortium
TPAL	Transport Adaptation Layer
UCAN	Unified Cellular and Ad-Hoc network Architecture
UDDI	Universal Description Discovery and Integration
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
URI	Universal Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network
WSDL	Web Services Description Language
WSDL	Web Services Description language
WWAN	Wireless Wide Area Network
XML	eXtensible Markup Language
ZRP	Zone Routing Protocol

CHAPTER 1: Introduction

This chapter starts with the motivations for this subject, and then states the problem, the thesis objectives and its major contributions, along with the related publications. It ends with the thesis organization.

1.1 Motivations

Over the last decade the telecommunications domain has gone through historical changes. In just a few years the evolution of networks, technologies and even the telecommunications vision has experienced rapid changes. For instance, mobility has become a necessity in telecommunications. In the mobile context, several network generations have been studied and experimented upon. The common goal is to provide better services to consumers. However, the actual services and how they have been provided have changed dramatically [1], [2] due to advances in standards and technologies.

The first generation (1G) of mobile telecommunications was released in the early 80s. It was based on analog cellular systems and was intended primarily to provide voice calls on the move. The quality of this service was poor, and still the demand was growing. Then, the second generation (2G) was released in the 90s. The world had moved towards digital standards. The most widely-used standard was Global System for Mobile

Communications (GSM). With the digital system, a few services were added to the voice call: small-data transmission, Short Messaging Service (SMS) and Enhanced Messaging Service (EMS). Despite this evolution, the 2G was still a circuit-switched network, and so it inherited the drawbacks of those networks. The 2.5G was seen as an enhancement to the 2G because it moved forward to packet-switched networks for data services. Hence, new services and opportunities became available: Internet browsing, e-mail, file and data transfer at higher rates. As the technology advances and consumers demand grows, new needs appear. In the 2000's, the third generation networks (3G) became a reality. 3G provides a service-oriented network that ensures high service quality, high transfer rates and that opens the door to a wide range of services such as multimedia services, fast mobile Internet browsing, and TV direct to a mobile device. Mobile devices are highly integrated into today's lifestyle. Bringing together the Internet Protocol (IP) and telecommunications under the 3G umbrella has allowed new business opportunities and different types of networks and architectures to flourish.

In the near future, the main challenge will not come from technology but from integration. Therefore, the fourth generation networks (4G), also called beyond 3G networks, are envisioned as the coexistence and cooperation of legacy and new networks [3].

Mobile Ad-hoc NETworks (MANETs) are an example of such new networks. They have already made significant contributions to military and disaster relief operations. Efforts to expand their use to civilian life have been gaining more and more momentum. Recently, a variety of research has recently been published on MANETs.

MANETs are a collection of mobile nodes with no pre-established configuration or infrastructure. Owing to their on-the-fly aspect, MANETs provide interesting opportunities and allow new business models [4], [5]. MANETs are opportunistic, self-organized and self-managed networks. They form and grow in a much more natural way than infrastructure-based networks. Their deployment involves no extra costs, since no new entities are required except for end-user devices. Furthermore, heterogeneous devices, and thus any user, can take part in a MANET. The dynamic topology of such networks makes them flexible and fault-resistant.

The military and disaster relief domains have already taken advantage of these networks for some time. MANETs' applications have been designed essentially for battlefield and emergency situations. However, the wide use of wireless handheld devices, due to the decreased cost of wireless technology, has allowed MANETs to penetrate the commercial, educational and personal domains. In fact, several studies to extend the benefits of MENETs have been successfully conducted in recent years. The generalization of such networks brings new challenges [6]. Their highly dynamic topology, situations of unpredictable connections and disconnections, integration with heterogeneous devices, especially those with limited resources (e.g. bandwidth, battery power) constitute the main challenges. To date, research has focused on the lower-layer issues [7], [8], [9]. Thus far, no concrete solution for the application layer has been proposed and MANETs have stayed at an experimental deployment level.

Service architectures allow new services to be created and brought to users quickly and reliably. These encompass two aspects: the business model and the service lifecycle. The former defines the business entities and the interactions between them. The service lifecycle is a four-phase process: service creation, service deployment, service usage, and service withdrawal. The most important phase is the service usage, which includes a main process: service provisioning. Service provisioning allows an entity to advertise the services it is willing to share with other entities. It also enables these entities to access and invoke the desired services via the service invocation process. Furthermore, service provisioning enables service execution, which is a process that runs services and manages the involved entities during the execution time.

Furthermore, MANETs allow new business opportunities by enabling new services and applications. Potential services for MANETs are: conferencing such as interest-based conferencing service we considered in this thesis; gaming such as urban games where the game area is the street. The players interact with the real world; and entertainment such as watching stream video clip or TV shows. However, the success of MANETs depends on a realistic and concrete business model and service provisioning solutions. This process is very challenging given their particular characteristics. Furthermore, MANETs may be integrated with legacy networks (e.g. 3G) which introduces different issues compared to the standard service provisioning process in an infrastructure-based network.

1.2 Problem statement and objectives

As previously mentioned, services are the heart of any network. Several paradigms have been proposed to provide services to end-users (e.g. mobile agent, web services, Parlay). However, providing value-added services to 4G network users requires an appropriate solution. There is a lack of research into the application and service layers of such networks. Furthermore, MANETs are known for their highly dynamic topology, limited resources, peer-to-peer communication and fully decentralized management. Providing services for these networks thus involves several challenges.

The main objective of this thesis is to propose an architecture for service invocation and execution in MANETs. This work focuses on both standalone MANETs and Multihop Cellular Networks (MCNs) as an example of the integration of MANETs and 3G networks. MCNs are in the center of interest for many research groups, especially telecommunications actors, since they open up new business opportunities. Several research issues can be derived from the above mentioned global goal.

The first issue is to derive a set of requirements for the service provisioning architecture in MANETs. Requirements should be proposed for the global architecture, the subsequent protocols and functional entities for both standalone MANETs and integrated 3G/MANETs. Respecting the requirements will ensure that the proposed solutions are suitable for the MANET environment.

The second challenge targets the business model for standalone MANETs. Defining a business model is an important step towards standardization. Indeed, it defines the players involved in a service provisioning and their interactions. Therefore, it is of utmost importance to carefully define a business model that takes into account MANETs characteristics. Basically we need to know what a business model looks like in a distributed environment with no infrastructure and resource-limited handheld devices. How many business model entities can be envisioned, and how they will interact must be included in this business model.

The third issue is related to service execution in standalone MANETs. In other words, how can the previously defined business model be generalized for service execution?

How different instances coexist and cooperate to run a service and what messages or protocols are required will need to be elaborated. And, how will the multiple business entities be organized. Then, how the service is invoked and executed with the resulting architecture also needs to be specified.

The following challenges target the integrated 3G/MANET systems. Indeed, stand-alone MANETs have limited applications since they are isolated from external access. By integrating them to existing 3G networks (e.g. MCNs), MANETs become interesting networks for applications and services. A large community of users can then take advantage of them. Therefore, the fourth main issue is to first to identify the different and possible ways for integrating MANETs and MCNs at the application level. A variety of alternatives to provide 3G services to MANETs users or provide MANETs services to 3G subscribers are feasible. Thus, it is essential to know what these alternatives are, their benefits and the research challenges they involve. Second, the integration, in terms of interactions and cooperation, should be described with MCN characteristics in mind. Specifically, how users can invoke and execute a service in these networks. Consequently, it is fundamental to identify what are the functional entities, procedures and protocols needed to achieve this goal.

The objectives of the thesis are summarized as follows:

- Derive a set of requirements at different levels for service provisioning in standalone MANETs.
- Propose a novel business model that enables service provisioning in standalone MANETs.
- Define a general architecture for service provisioning in standalone MANETs.

- Propose a concrete architecture for provisioning services in integrated 3G/MANET systems, more precisely, in Multihop Cellular Networks (MCNs).

1.3 Summary of contributions

This section pinpoints the main contributions of the thesis and presents references to the related publications.

- **Critical review of the state of the art and derived requirements**: we derived general requirements for service provisioning architecture in standalone MANETs and also for integrated 3G/MANETs. Furthermore, we define refined requirements for related protocols and entities and for a concrete solution in integrated 3G/MANETs systems. Based on these requirements we then present a critical review of the existing solutions and conclude that none of them are suitable for integrated 3G/MANET systems.
- **Business model for service provisioning in standalone MANETs** ([10], [11]): we have proposed a new business model that takes into account MANETs characteristics. In fact, the new business model does not depend on a permanent central entity -- lightweight functions are offered by each role and the business model is flexible enough to allow dynamic discovery of the provided functional entities. Furthermore, the proposed business model not only targets organizations but individuals as well. Thus, any participant can take part in the business model. Proof–of-concept prototypes were implemented to demonstrate the solution feasibility.
- **An overlay network for service invocation and execution in standalone MANETs** ([12]): We have proposed an overlay network that fits the previously introduced

business model. The overlay network was designed to allow service invocation and execution in a highly dynamic environment such as in MANETs. This network is composed of different types of nodes that coexist and cooperate to provide a service execution environment. The solution includes a protocol for self-organization and self-recovery, making the overlay network fault-tolerant. The protocol validation was performed.

- **Service provisioning architecture for integrated 3G/MANETs (**[13], [14], [15]**)**: we proposed a novel architecture for integrated 3G/MANETs. Different high-level architectural alternatives [13] for service provisioning in integrated 3G/MANETs were studied and described. From this, a concrete architecture corresponding to the most interesting solution from the network operator point of view was detailed [14]. The architecture is based on a new functional entity called Service GateWay (SGW) and no major upgrades are required for the existing 3G and MANET service provisioning entities. For the performance evaluation of the proposed architecture, we opted for simulation using OPNET. The collected results show that the solution is obviously introducing delays compared to the existing 3G architecture [15]. However, these delays are acceptable.

1.4 Thesis organization

The rest of this thesis is organized as follows: Chapter 2 presents crucial background information on Mobile Ad-hoc Networks and Multihop Cellular Networks. It introduces key concepts of SIP Servlets, since our architecture is based on this paradigm. Existing service provisioning solutions are then described. In chapter 3, we first derive a set of requirements for the service provisioning architecture for standalone MANETs, and then for integrated 3G/MANETs. Requirements regarding the service invocation and execution architecture are derived as well. Then, we critically review the state of the art by subjecting the existing solutions to our requirements. Chapter 4 proposes a novel business model, designed for standalone MANETs. This chapter shows that our proposition meets the previously mentioned requirements. The different business roles and their interactions are elaborated. Chapter 5 is devoted to the service invocation and execution architecture in standalone MANETs. We propose and define an overlay network with the related overlay protocol. This architecture enables self-organization and self-recovery. Chapter 6 discusses the architecture for service provisioning in integrated 3G/MANET systems. Multihop cellular networks (MCNs) are considered as an example of an integrated 3G/MANET system. This chapter gives an exhaustive overview of the architectural alternatives for integration at the service level. The most interesting alternative from the network operator point of view is then detailed: assumptions, functional entities and procedures are discussed. Chapter 7 describes the different proof of concepts we have implemented for the standalone MANET solution. Chapter 8 elaborates on the OPNET simulation setups and results for the integrated 3G/MANET architecture. The chapter 9 concludes the work, gives the summary of contributions and outlines items for future work.

CHAPTER 2: Background

This chapter presents the background information required for the optimal comprehension of this thesis. We start by introducing mobile ad-hoc networks, their description, characteristics and evolution, and multihop cellular networks. Next, we provide an overview of the service architecture concepts including the service lifecycle and business model notions. Then, we describe the service provisioning architecture of choice for 3G networks. Finally, we present the SIP Servelts service provisioning paradigm.

2.1 Mobile Ad Hoc Networks

Mobile ad hoc networks (MANETs) can be defined as a collection of autonomous and self-configuring nodes or terminals that communicate with each other by forming a multihop radio network and maintaining connectivity in a decentralized manner [16]. The term “ad hoc” means that the network is established arbitrarily for a limited period of time and for a specific objective [17].

The major goal of MANETs is to set up communications where there is no pre-established infrastructure (e.g. a battlefield), or where the infrastructure has failed (e.g. in disaster relief), or when a pre-established infrastructure is not adequate for the current needs (e.g. interconnection of low-energy environmental sensors) [17].

Basically, in a MANET each node plays the role of a client, a server and a router. The network is based on the wireless 802.11 standard for large scale networks and Bluetooth specifications for short range communications. Mobile ad hoc devices and nodes can range from laptops to small handheld gadgets: Palmtop, Personal Digital Assistants (PDAs), mobile smart phones, pagers, sensors and the like.

2.1.1 The evolution of MANETs

Work on mobile ad hoc networks began in the early 1970s. The project was initiated by the U.S. Defense Advanced Research Projects Agency (DARPA). In fact, the first step towards MANETs was the ALOHA project (1970), which showed the feasibility of packet broadcasting over a single-hop network. Then, in 1979 the DARPA started experimenting with multihop, multi-access Packet Radio NETwork (i.e. the PRNET project) [18]. Inspired by the success of PRNET and the wide use of inexpensive 802.11 radio cards for personal computers, many projects led to the development of ad hoc routing algorithms during the 1990s [16], [18]. Furthermore, the Internet Engineering Task Force (IETF) created the MANET group, which works mainly on the routing aspects of MANETs [19].

MANETs have now gained even more momentum -- taking advantage of the maturity of research in the lower layers, the advances in wireless technology and standards and the low cost and diversity of small devices. New opportunities and applications have become very promising. Indeed, the solutions already include: community networks, home networks, vehicle networks, sensor networks, emergency networks and hotspots. MANETs have opened up multiple commercial applications such as: entertainment, education, shopping and collaborative work.

2.1.2 Classification of MANETs

MANETs are considered a subset of wireless networks with the particularity of being infrastructure-less. Furthermore, sensor networks are viewed as independent subsets of the MANET family. However, sensor networks are significantly different from MANETs at the physical, MAC, network and application layers [17]. Thus, the issues of concern in sensors' networks are not the same as those in MANETs. Therefore, they are not considered in this work.

MANETs are generally classified according to the communication coverage area. In fact, they include four network types: Body Area Networks (BAN), Personal Area Networks (PAN), Wireless Local Area Networks (WLAN) and Wireless Wide Area Networks (WWAN) [4], [16]. Figure 2.1 illustrates this classification.

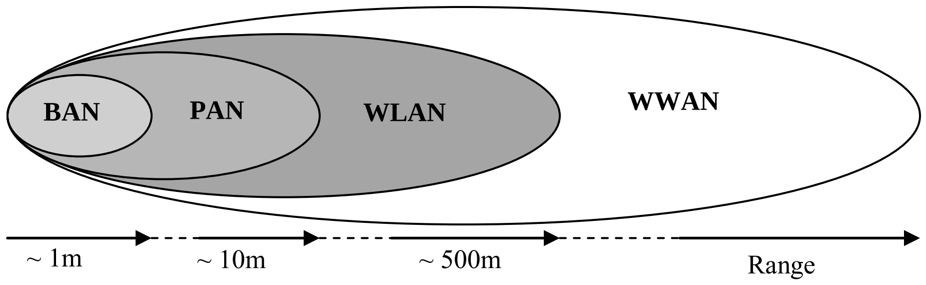


Figure 2.1: MANETS categories based on their communication coverage

The ad hoc WWANs have connections that cover a large geographic area. Generally, a sensors' network forms a WWAN. Soldiers in a battlefield usually have access to these networks. Another example is large-scale games that use sensors. In [20] the authors present challenges and directions related to the mobile ad hoc wide area networks. The infrastructure-less WLAN targets medium-size areas such as a campus or an enterprise or an airport. PANs allow users to establish connections with other entities in the surrounding area using personal devices (e.g. laptops, PDAs, cellular phones). A BAN is linked to wearable devices (e.g. microphones, earphones, watches) and provides connectivity through these gadgets. A BAN can be either interconnected with other BANs to communicate with other people or connected to a PAN for Internet access.

Furthermore, MANETs are classified into three models: standalone MANETs [21], connected MANETs [22] and hybrid MANETs [23], [24], also called the integrated model. In standalone models, the network is completely isolated from any external connection or infrastructure network. The ad-hoc network is formed by the devices within the communication range. Standalone MANETs are very useful since they are easy and cost nothing to set up. Basically, they are temporary networks and are useful where no infrastructure is available, such as in a battlefield or in a disaster relief area. Standalone ad-hoc networks address the need for deploying a network immediately. However, their application is limited since no external access is provided and thus they cannot make a large number of different users benefit from their services. Figure 2.2 shows a typical standalone mobile ad-hoc network.

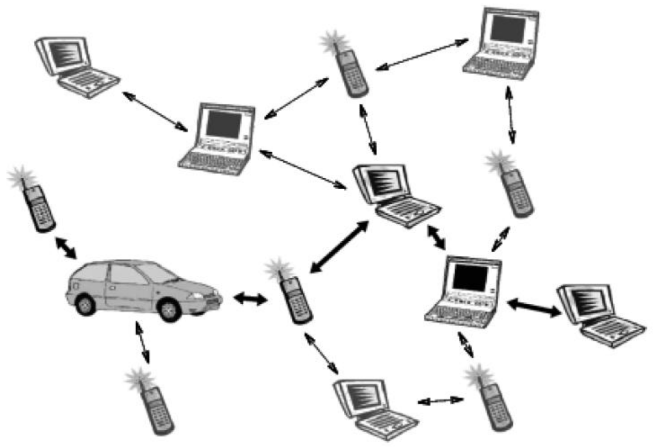


Figure 2.2: A typical standalone mobile ad-hoc network

The second model, connected MANETs, are standalone ad-hoc networks with an access point (AP) to a larger network, in most cases, the Internet. Figure 2.3 gives an overview of a connected MANET model. This model can be used as an extension to network coverage. Hotspots are a good example of connected ad-hoc network models. Indeed, hotspots tend to be deployed in an ad-hoc manner wherever wired Internet access is available.

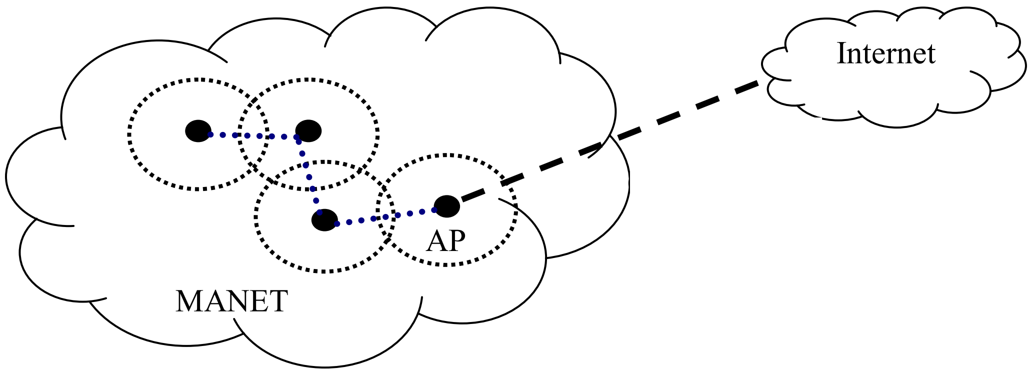


Figure 2.3: Overview of the connected mobile ad-hoc network model

The third model is the result of the integration and coexistence of standalone MANETs with an infrastructure network. Initially the goal of such integration was to improve the connectivity [25]. Furthermore, this integration is usually achieved with a wireless cellular network. The resulting network consists of a sparse network of base stations and ad-hoc nodes. Figure 2.4 illustrates a general view of the integrated model, which presents a trade-off between classical cellular networks and pure ad-hoc networks. Traffic can be routed either through ad-hoc nodes or through base stations (BS). Integrated 3G/MANET networks [26] are the most promising and well-known solution for the hybrid model of MANETs. The most common example of integrated 3G/MANETs is Multihop Cellular Networks (MCNs), which are elaborated on in the next sub-section.

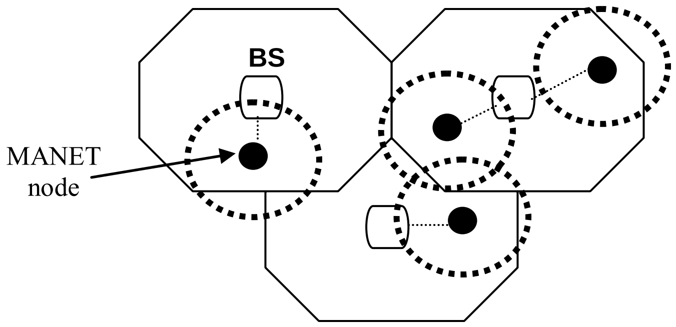


Figure 2.4: General view of the integrated cellular network/MANET model

This thesis focuses on standalone MANETs and MCNs.

2.1.3 Multihop Cellular Networks (MCNs)

Several wireless technologies are available today to respond to an increasing demand for a variety of new brand services. Furthermore, small gadgets have become an integral part of our everyday life. Therefore, allowing users to seamlessly access various services from different devices and via heterogeneous networks is an excellent response to this situation. The integration of these technologies is a fundamental step to provide mobile users with services “anytime, anywhere, with any device” and with the guaranteed Quality of Service (QoS). These objectives are also the motivation behind the 4G vision [27] and [28].

MANETs and Cellular networks are two types of existing wireless technologies. Each of them has interesting and also complementary features [29]. MANETs provide high throughput rates while the cost of deploying and accessing the network are low. Furthermore, they rely on multihop communications. However, their communication range is limited. Cellular networks can reach a wide area and thus many more mobile users than a MANET. Additionally, they are more easily controlled with reliable billing and security systems. However, they use a single hop communication and the bandwidth is relatively low. The Third Generation Partnership Project (3GPP) has standardized 3G cellular networks [30]. Therefore, we can talk about 3G/MANET integration, and Multihop Cellular networks (MCNs) are a practical example. Figure 2.5 shows an overview of a MCN.

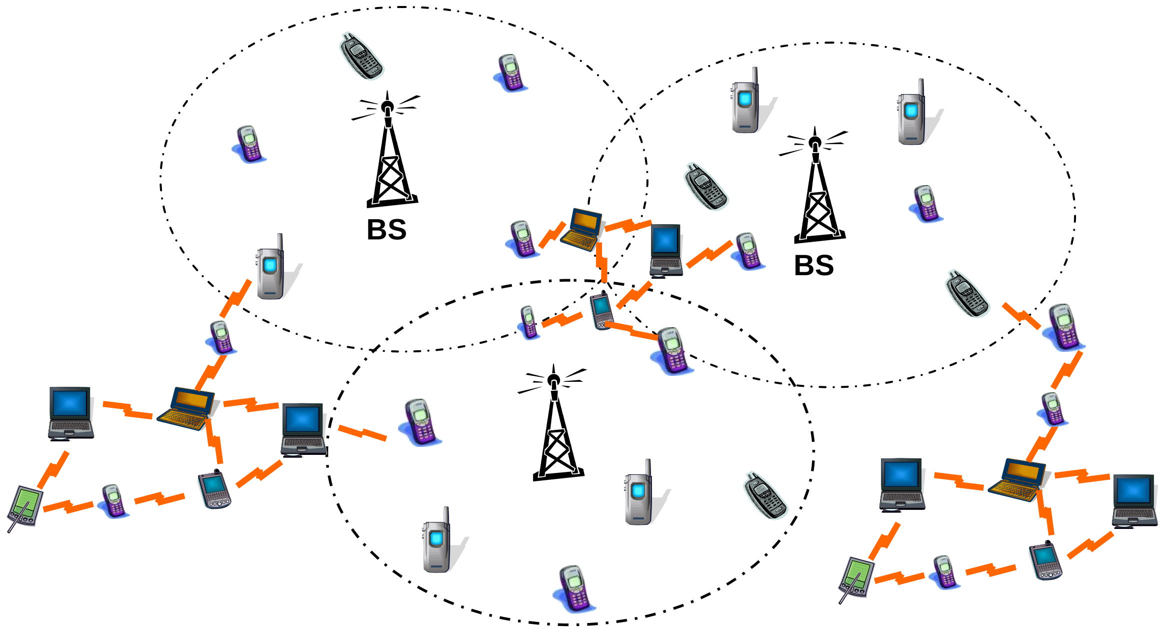


Figure 2.5: Overview of the multihop cellular network

The integration of a MANET and a cellular network takes advantage of the attributes of both networks, and therefore has received much attention from academia and industry. The standardization bodies, 3GPP and 3GPP2, have published several works on the integration of wireless LANs and cellular networks [31], [32] and [33]. Furthermore, other works have been carried out to achieve this integration [34], [35] and [36]. Nevertheless, thus far, only wireless infrastructure-based networks, that happen to be single-hop, have been considered. The integration with MANETs is trickier but has been studied in [37], [38] and [39]. The architectures proposed in these works accomplish the integration at the lower-layer level and aim at balancing the load and extending the cellular networks' coverage. Indeed, the Integrated Cellular and Ad Hoc Relaying Systems (iCAR) [37] introduces *Ad hoc Relay Stations* (ARSs) to share channel resources between cells and then balance the load dynamically. The main goal of the Unified Cellular and Ad-Hoc Network Architecture (UCAN) [38] is to extend the coverage and improve the throughput between the BS and the Mobile Host (MH). This architecture exploits the high throughput of surrounding MHs by using proxy and relay MHs. More recently, in [39] the authors address MANET/cellular network integration from the connectivity point of view. They propose different patterns to realize the connectivity between a MANET and a 3G cellular network. These solutions remain at the lower layers. Higher layers have received growing attention very recently. In fact, the work in [40] suggests a cluster-based architecture for signalling in MCNs while the authors in [41] propose an architecture for media handling for conferencing in MCNs. However, the above-mentioned solutions do not consider the integration at the service layer.

2.1.4 MANET characteristics

MANETs bring new challenges and add new constraints. Several characteristics distinguish them from classical networks [17], [42] and [43]:

- **Infrastructure-less:** basically, ad-hoc networks do not rely on any infrastructure support. The network must operate independently of pre-established or centralized entities. Network management and routing, for example, should be done in a cooperative way. Each node acts as a client, a server and a router in a distributed peer-to-peer mode.
- **Dynamic topology:** because of node mobility and membership changes, the network topology varies continually and frequently. While moving, nodes alter their relation to

their neighbours. Furthermore, new nodes can join at any moment, whereas connected nodes may leave in an arbitrary fashion. Thus, routes break down and unannounced disconnections are to be expected frequently in ad-hoc networks.

- **Heterogeneous nodes:** ad-hoc networks very often consist of a mix of different devices. Indeed, the network is open to any user holding any wireless gadget. As a result, nodes may have dissimilar features, may be of diverse size or may be configured with different software/hardware capabilities. These differences must be taken into account when designing algorithms or protocols for ad-hoc networks.
- **Resource constraints:** devices have become smaller and smaller, and with less resources (e.g. memory, processor speed, battery power). When these limited devices come into an ad-hoc network they bring issues related to these constraints. Efficient algorithms and energy management are called for when MANETs are targeted.

2.2 Service architecture

The economy today is becoming more and more service based rather than manufacturing or even product based [44]. An evolution can be observed in the way functionalities have been specified, provided and consumed. Indeed, the level of abstraction has continued to rise. We have thus moved from modules, to objects, to components, and now to services. The term “service” is used for multiple meanings and can be defined according to various perspectives [45]. However, a common sense definition for a service is a set of goods or valuable functions offered by a service provider to a consumer [46]. Examples of services include conferencing, online gaming; printing; travel booking; weather forecasting; sports results and so on. In the telecommunications industry, a value-added service is any service that goes beyond the classical two-party call service.

A service architecture allows new services and applications to be created and brought to users quickly and reliably. The telecommunication industry has been using the Telecommunication Information Network Architecture (TINA) [47], which is the first service architecture in the domain. TINA was proposed by a worldwide collaborating group of operators and computer equipment suppliers: the TINA Consortium (TINA-C). Throughout this thesis a service architecture is a set of concepts, rules and principles to support the service lifecycle [48]. In addition, the service architecture defines the business model and how it can be applied to the architecture.

2.2.1 Service lifecycle

The service lifecycle was first introduced by TINA-C [46], [49]. It encompasses four main phases:

a. Service creation: All the activities related to service logic production are a part of service creation. Hence, service specification, code design, implementation and testing are the core activities of this phase.

b. Service deployment: During the deployment phase, the service logic is installed in the appropriate network nodes. The service is then activated. Therefore, an adequate deployment strategy has to be prepared.

c. Service usage: services are created and deployed so that they can accessible to and used by users. Many activities are necessary to achieve satisfactory service usage. Besides users' authentication and authorization, this phase contains [50]:

i. *Service description*: is responsible for describing a service in a comprehensive and unambiguous manner. The description uses a clear syntax to specify what the service does, and how and where it can be reached. The output of this process is the *service profile*, which should be machine interpretable and human readable to facilitate both automation and rapid formulation by users. A good service description language may help. For example, the well-known WSDL (Web Services Description language) [51] from the web services world is an XML-based service description language.

ii. *Service advertisement:* by analogy, we can say that service advertisement is the deployment of service descriptions. It allows service descriptions to be reachable by everyone so that users are made aware of the existence of the services. Descriptions can either be published on a central registry, or directly to the other nodes in the network.

iii. *Service discovery:* is an important activity in the service usage phase. By the end of this process the user knows about existing services and how to bind them. Discovery of services can be done in a *pull*, *push* or *hybrid* fashion. The pull method is based on the classical request/reply paradigm. In the push method, the provider does not wait for a user request. It pushes service descriptions to the network periodically or when an event occurs (e.g. a user has joined, a new service is added). The hybrid is a combination of the pull and push methods.

iv. *Service invocation:* deals with the management of the communication between the user and the provider to facilitate the use of services. It includes sending commands, receiving results, maintaining connections, and abstracting details from the user's point of view.

v. *Service interaction:* when many services must coexist, an interaction problem may occur. Services working well alone may crash when put together. This is because a service might modify or influence another service in defining the overall system behaviour. Service interaction is managed and problems related to the interactions between services are solved.

vi. *Service maintenance:* is the same activity as in software engineering. The main actions are bringing changes to the service logic or/and correcting faulty services.

d. **Service withdrawal:** is the deactivation of the service at the network level and/or its removal from the network.

2.2.2 Business Model

The business model concept is not new. Many models are, in fact, used in different domains. From the perspective of economics, a business model is a framework for creating value and capturing returns from that value within a value network [52]. A business model should address four main issues [53]: identify the customer; define the customer value (i.e. the service); describe the underlying logic that ensures customers' value delivery; and explain how money is made within that business. In other words, a business model covers two generic actions connected to doing business. The first part encompasses all of the activities related to producing something: design, manufacturing and so on. The second part contains all the activities related to selling something: finding and reaching customers, transacting a sale, distributing the product or delivering the service. From a technical and ITs perspective, a business model describes the different players or roles involved in service provisioning and their relationship to each other [54]. It is an important part of the service architecture and varies according to the nature of the architecture and its application. Furthermore, a business model is usually considered as a staring point for standardization since it identifies the interfaces between the different roles. In the following section, we present the business model of TINA-C, which has inspired several recent architectures and business models.

2.3 Service provisioning in 3G networks

Service provisioning is the process by which entities advertise their willingness to offer specific services, and discover and run other services. It is a keystone to successful service architectures. The IP Multimedia Subsystem (IMS) [55] is the most important 3G service architecture today, and it is becoming the architecture of choice for the Next Generation Networks (NGN). Therefore, we can and should discuss the IMS architecture from the service provisioning perspective. The objective of IMSs is the convergence of cellular and IP networks. In other words, IMS aims to offer Internet services anywhere and at any time via cellular technologies. Furthermore, IMS architecture will permit the creation of new brand services by allowing operators to provide, combine and integrate services from third parties.

2.3.1 IMS architecture

The IMS uses several protocols, however its driving force is to be based on one common session control protocol, namely, the Session Initiation Protocol (SIP) [56]. The IMS architecture is horizontal and structured in two plans: control and service. These two plans are overlaid on top of a transport layer. The transport layer contains routers and switches and allows different devices to access the network via a variety of network accesses. The control plan contains control servers (SIP servers) for managing the multimedia sessions. The service plan contains the application servers (AS) hosting and executing value-added services.

The IMS architecture rests on SIP servers, each hosting central functions. The most important is the Call Session Control Function (CSCF), which processes SIP signalling messages. Therefore, all of the traffic should go through one or more CSCF. There are three types of CSCFs: Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF) and S-CSCF (Serving-CSCF). The P-CSCF is the entry point to the IMS network. The P-CSCF maintains several functions. For instance, it authenticates users, disseminates users' identity and verifies the correctness of SIP requests. The I-CSCF is a proxy server used by SIP servers to find the next hop, as described in the SIP specification [57]. The S-CSCF is a central node in the IMS architecture. It registers users, controls sessions, provides billing information, processes routing and translations, and verifies users' authorizations and profiles. Besides the CSCFs servers, the HSS is another important server. It is a database of all IMS subscribers and service data. All of the sensitive data related to users, their profiles and data describing their service behaviour and providers are stored in the HSS. The IMS core network contains other media-based functions. The Media Resource Function (MRF) allows media content to be played and controlled. Furthermore, different media gateways are also provided in the IMS network: Media Gateway (MGW) and the Media Gateway Control Function (MGCF). These are interfaces with the legacy circuit-switched networks (e.g. Public Switched Telephone Network - PSTN-). The next section introduces the SIP, since it is at the heart of the IMS network. Figure 2.6 illustrates a simple IMS architecture.

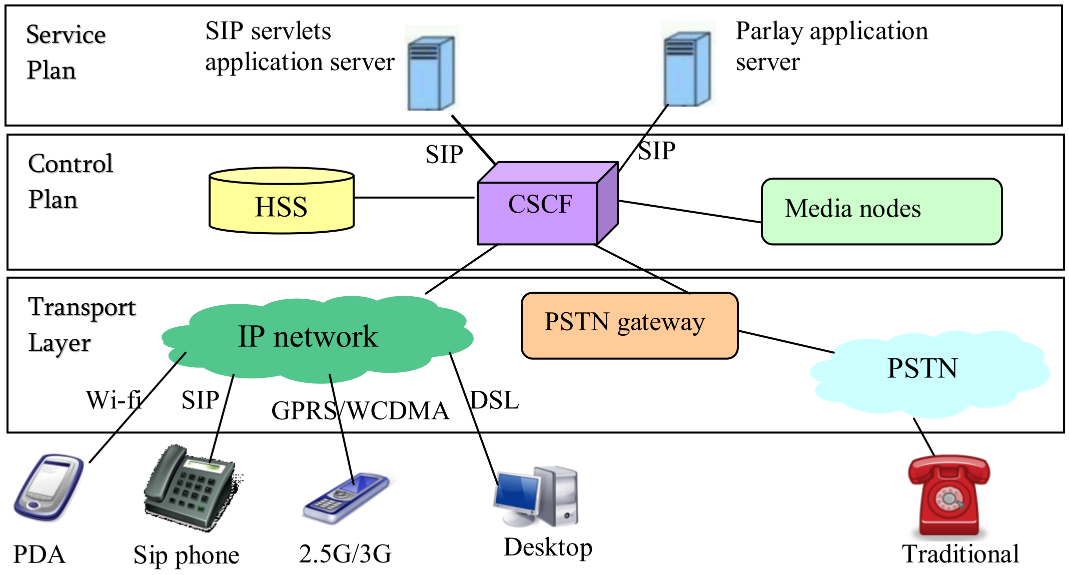


Figure 2.6: Simple IMS architecture

2.3.2 Session Initiation Protocol

The Session Initiation Protocol (SIP) [56] is a widely used signalling protocol for multimedia conferencing over IP. It is independent of the underlying transport protocol. SIP allows establishing, maintaining and terminating multimedia sessions between two or more endpoints. The endpoints in SIP are referred to as SIP User Agents (UA) and act as a UA Client (UAC) or a UA Server (UAS) to create new requests and generate responses, respectively. Furthermore, three types of servers are defined: the SIP registrar server, the SIP proxy server and the SIP redirect server. The SIP registrar server keeps track of users' locations. Users register their location whenever it changes. The SIP proxy server is a SIP router. It receives messages and then forwards them to one or multiple destinations based on specific criteria. The SIP redirect server is also a SIP router, however it has a different behaviour It informs the sender about an alternative location where the destination can be reached.

The SIP is very attractive because of its ease of use and flexibility. It can handle several operations with only a few messages, called SIP methods. The main SIP messages are INVITE, CANCEL, ACK, BYE and REFER to initiate a session, cancel a SIP transaction, acknowledge a successful response, terminate a session and trigger a request sent to a third party, respectively. Table 2.1 summarizes core SIP methods and extensions [55].

SIP methods	Meaning
INVITE	Establishes a session
BYE	Terminates a session
ACK	Acknowledges the establishment of a session
CANCEL	Cancels a pending request
REFER	Instructs a server to send a request
REGISTER	Maps a public URI with the current location of the user
SUBSCRIBE	Requests to be notified about a particular event
NOTIFY	Notifies the user agent about a particular event
OPTIONS	Queries a server about its capabilities
MESSAGE	Carries an instant message
INFO	Transports application information with the signalling
PUBLISH	Uploads information to a server
UPDATE	Modifies some characteristics of a session
PRACK	Acknowledges the reception of a provisional response

Table 2.1: Meanings of SIP core methods and extensions

2.3.3 Service provisioning in IMS

Prior to any message exchange in an IMS network, a user's terminal should perform some operations. The most important is registration. A user is allowed to access a service if and only if it is registered with that IMS network. In the following sub-sections, we assume that the registration step has been done.

2.3.3.1 Application servers

Application Servers (ASs) are core entities in the service plan. They are responsible for hosting and executing value-added services. Multiple ASs coexist in an IMS network, where each AS provides a specific service to end-users. The application servers may implement different technologies (e.g. SIP servlets, SIP Common Gateway Interface - CGI- , Java technology). Nevertheless, a common requirement is to provide an SIP interface to the S-CSCF. Furthermore, an AS may possess additional technologies, such as HyperText Transfer Protocol (HTTP) [58] or Wireless Application Protocol (WAP) [59].

Three types of ASs are proposed by the IMS: SIP Application Servers (SIP AS), Open Service Access-Service Capability Servers (OSA-SCS) and the IP Multimedia Service Switching Function (IM-SSF). The SIP AS is the instinctive application server of an IMS. In fact, new 3G services are developed specifically for the SIP AS. The goal of the OSA-SCS and the IM-SSF is the integration with legacy networks to access existing services. Indeed, the OSA-SCS is a gateway to the existing services in OSA framework servers, while the IM-SSF is a gateway to existing Customized Applications for Mobile network Enhanced Logic (CAMEL) services, as in the GSM environment.

2.3.3.2 Service provisioning

As mentioned above, a user must be registered with the IMS network before accessing the services. A user profile is available in the HSS for identification, authentication, and authorization purposes. The user profile also contains information about the services a user is allowed to access, the criteria for triggering a service, and which ASs are concerned with that service. This information is called the *initial filter criteria*. The user profile contains zero or multiple *initial filter criteria*. The user profile is retrieved by the S-CSCF from the HSS at registration time. Therefore, after registration the *initial filter criteria* are available in the S-CSCF. When the S-CSCF receives a certain type of SIP message, it evaluates the *initial filter criteria* and then decides if an application server is to be contacted or not. If it does decided to contact an AS, the S-CSCF also decides which AS to call, based on the *initial filter criteria*, and that AS executes the service. A service may involve one or several ASs. The S-CSCF evaluates the *initial filter criteria* when it receives SIP messages that are not subsequent requests. In other words, the evaluation takes place with SIP requests that create a SIP dialog or that are stand-alone requests (e.g. INVITE, SUBSCRIBE, OPTIONS).

2.3.3.3 Scenario

Concretely, we will see how a service is provided in IMS. We consider an *interest-based conferencing* service. The service automatically establishes a conference between online subscribed users that share the same interests when a quorum is reached. New users can join the ongoing conference at any time.

Alice, Bob and Carol are three registered 3G users. They all subscribed to the interest-based service and all share the same interests. The information about their interests can be updated via HTTP. The *initial filter criteria* corresponding to their respective profiles are now in the S-CSCF. In the interest of clarity, we assume that each of the three users is linked to the same S-CSCF. We also assume that the service quorum is set to three, by Alice who had initiated the service. Furthermore, we consider a combined AS/MRF entity to handle media conferencing. The collocation of the AS and the MRF is described in the 3GPP specifications [60]. In addition, the P-CSCF and the S-CSCF are combined and referred to as the CSCF. Figure 2.7 illustrates the message flow when the third user (i.e. Carol) invoked the service.

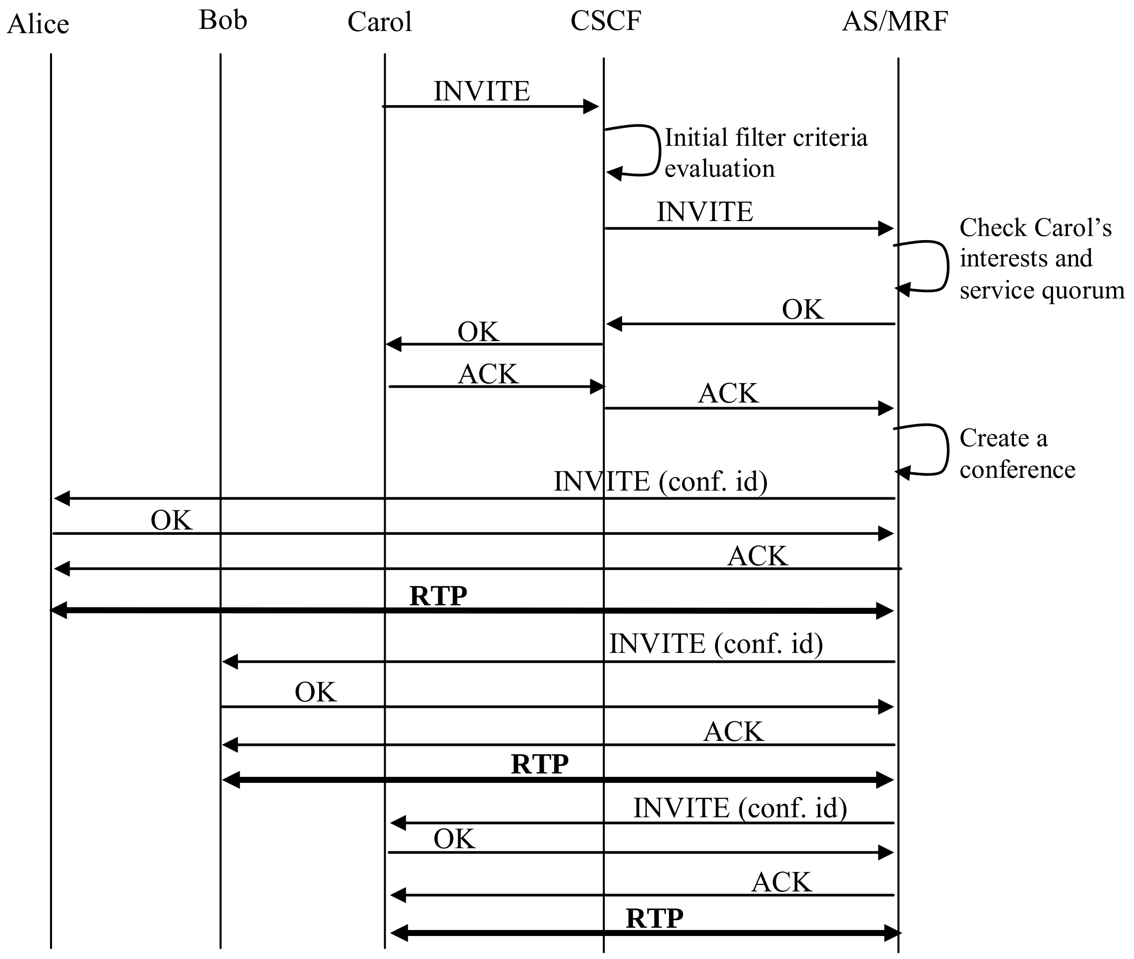


Figure 2.7: An example of service provisioning in IMS: an interest-based service

Alice and Bob have already invoked the service but since the quorum was not yet reached the conference had not started. Now, Carol sends an INVITE message via the P-CSCF. When the S-CSCF receives the SIP request, it evaluates Carol's *initial filter criteria* and then contacts the AS/MRF that executes the interest-based conferencing service. The AS/MRF checks the interests of Carol and matches them with the interests provided by Alice (the service initiator). Since the interests correspond and the quorum has been reached (Carol is the third user to request the service), the AS/MRF creates a conference and invites the users to join it.

2.4 SIP servlets framework

2.4.1 The servlets technology

A *servlet* is a Java application that runs on the server side to handle clients' requests and to provide services. Java Servlets can be seen as a layer between the client and the services. Java Servlets are not tied to a specific protocol. Furthermore, since servlets are Java-based they are independent of any platform. The servlet technology benefits from the power of the Java programming language. They have become a popular and successful paradigm for web development and web-application development [61]. Several characteristics contribute to the current wide adoption of Java servlets' technology (e.g. scalability, reusability, industry-wide support). Java Servlets [62] are managed by a servlets *container,* also called a servlets *engine*. Clients invoke services, implemented by servlets, through the Servlets Engine (SE). The SE is the entity responsible for maintaining servlets' lifecycle: it loads the servlet, initializes it, calls the appropriate servlet method upon the reception of a message and finally destroys the servlet. Figure 2.8 illustrates the servlets lifecycle.

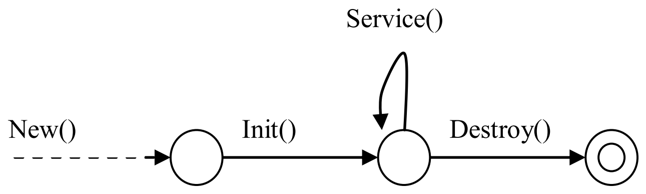


Figure 2.8: Java Servlets lifecycle

The SE provides four methods that implement the servlet lifecycle's functionalities: *new()* to load or create a servlet, *init()* to initialize its parameters, *service*() to run the logic corresponding to the incoming message and *destroy()* to remove the servlet.

Although, Java servlets are protocol-independent they are commonly used with HTTP, since all of the servlets' engine implementations should support HTTP. Servlets technology has mainly been used with HTTP for dynamic generation of web content. The servlet may add new headers, modify existing values or attach content to the response.

2.4.2 SIP servlets

SIP servlets [63] are Java-based applications performing SIP signalling logic. They congregate both Session Initiation Protocol (SIP) and HTTP servlets concepts. Therefore, SIP servlets' APIs extend the functionalities of SIP servers and allow the new services to be created easily [64]. Figure 2.9 presents a simple view of the SIP servlets framework.

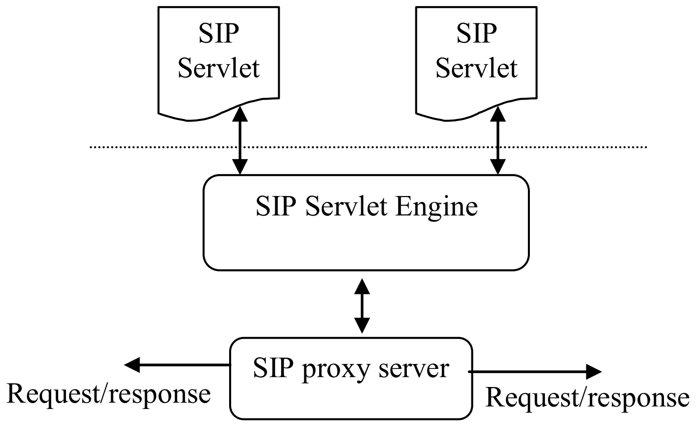


Figure 2.9: Simple view of the SIP servlets framework

The SIP servlets have the same core behaviour as HTTP servlets: When the servlet engine receives an initial SIP message it selects a servlet according to given rules and invokes its *init()* method. Then, each time a subsequent SIP message is received the engine calls the *service()* method of the servlet. This method contains the logic of the service and depends on the type of SIP message. Essentially, the engine runs the do**XXX**() method where **XXX** refers to the type of the received SIP message. For example, if the engine receives an INVITE message, the doINVITE() is called. At the end, the engine calls the *destroy()* method to stop the service and release the resources.

However, there are fundamental differences between SIP servlets and HTTP servlets. SIP servlets have extended features. A SIP servlet is able to respond to an incoming SIP request with zero, one, or multiple responses. A SIP servlet may also proxy an incoming request to one or several destinations, or it may generate a new SIP request. In general, the SIP servlets framework contains three functional entities: applications, SIP servlets and SIP servlet's engine. Applications are the service logic. The SIP servlets are building blocks for developing applications and the SIP Servlet Engine (SSE) is the execution environment that runs the servlets.

As mentioned earlier, SIP is the signalling protocol of choice for the NGN. Therefore, SIP servlets become a promising framework for service provisioning in the networks of the future. In the remainder of thesis we will consider a SIP application server implementing the SIP servlet technology. Figure 2.10 shows a SIP servlets-based application server.

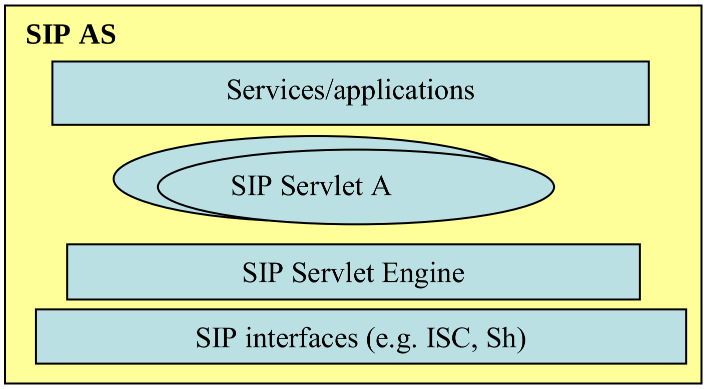


Figure 2.10: SIP servlets-based application server in IMS

2.5 Summary

In this chapter we have presented the necessary background information for our research. Mobile ad-hoc networks have been described, starting from their evolution, the different types of MANETs and their main characteristics which make them challenging environments. Both standalone MANETs and integrated 3G/MANETs have been presented. We have introduced multihop cellular networks (MCNs) as today's major example of integrated 3G/MANET networks.

This chapter then covered the key concepts of service architecture, namely the service lifecycle and the business model. We have elaborated on the service lifecycle's phases and introduced the business model. One of the most important processes for service architectures is service provisioning. We described how this process is performed in 3G networks. The IP Multimedia Subsystem (IMS) was considered, as it is the standard for 3G networks. The IMS architecture was depicted together with its core protocol SIP, and then a scenario was proposed to illustrate the service provisioning process in IMS.

The chapter ends with the SIP servlets framework as a promising paradigm for service provisioning in future networks. The servlet technology and the SIP servlets' architecture were described.

CHAPTER 3: Related Work

Providing value-added services in mobile ad hoc networks is not a straightforward task. The lack of research on the service layer of this environment makes the work more complex. However, certain approaches do exist, ranging from complex service architectures to solutions that address a specific aspect of the service provisioning. The common limitation of these approaches is that they were designed without MANETs challenges in mind.

Deriving a set of requirements and then evaluating the existing solutions in the context of these requirements is the focus of this chapter. The various requirements related to the architecture are presented in the first section, and the second section discusses and analyses the state of the art according to the derived requirements. Both stand-alone MANETs and Multihop Cellular Networks (MCNs) are addressed.

3.1 Requirements

This section is organized into four sub-sections. The overall requirements for service architectures in MANETs are presented first, followed by the requirements specific to each component of the architecture (i.e. business model and service execution framework). The requirements related to service execution frameworks in stand-alone MANETs are detailed in sub-section three. Finally, sub-section four discusses the requirements for service provisioning in MCNs.

3.1.1 Overall requirements for service architectures in MANETs

From the beginning, the service architecture should obviously consider the complete service lifecycle. The architecture should support service creation, deployment, usage and withdrawal. The second requirement is that the service architecture must not rely on any existing infrastructure or any central entity since pure ad-hoc networks are infrastructure-less, self-organizing networks., The architecture should also provide adequate mechanisms for advertisement and discovery. Unannounced and frequent disconnections are a common phenomenon in ad-hoc networks. The fifth requirement is that the service architecture should allow an optimal usage of resources. In fact, many ad-hoc network devices have limited resources (e.g. CPU, power, memory, battery, etc.). Good service architectures should be based on lightweight protocols and algorithms that require less processing. For example, advertisement and discovery mechanisms should be lightweight, and service description language should not engender heavy processing.

Next, the architecture should be able to handle different types of devices with dissimilar capabilities. Indeed, the heterogeneity of devices (e.g. laptops, PDAs, cell phones) is a particularity of MANETs. Thus, services should be able to be discovered and run on different types of gadgets. Finally, the architecture should enable flexible and varied business models. The dynamicity and temporality of ad-hoc networks requires flexible and varying business models. Two specific aspects are considered. First, the business entities may change their roles depending on the context. Second, any individual may wish to join the network and play a business role.

3.1.2 Requirements for business models in MANETs and related publication and discovery mechanism

Here we present business models' requirements for MANETs, including their motivation. The related publication and discovery mechanisms' requirements are then discussed.

3.1.2.1 Requirements of the business model

The first requirement is that the functional entities provided by the business roles should not be infrastructure-based nor centralized since MANETs are infrastructure-less and fully distributed by definition. Second, the business model should be flexible: it should be possible to dynamically discover not only roles, but also functional entities. This will address the dynamic aspect of MANETs where nodes can join and leave at any time.

Third, the business model should rely on individuals rather than on organizations. By individuals we refer to any entity present in a MANET at any given time, and by organizations we refer to business entities such as network operators that own or have control over the network. This requirement is a consequence of the infrastructure-less, the heterogeneity and the dynamic topology characteristics of MANETs. It also opens the network to new business opportunities. However, the provided functional entities should be lightweight enough in order to be offered by individuals with small devices.

In addition, communication between business roles should be done in a pure peer-to-peer fashion because no central entities are allowed in MANETs. Finally, the mechanism for the publication/discovery of the business roles and the functional entities they provide must take into account the characteristics of MANETs.

3.1.2.2 Requirements of the publication/discovery mechanism

First, for greater flexibility and interoperability, the mechanism must be independent from the underlying routing protocols. The mechanism should be fully distributed and it should adapt to the topology's changes. Furthermore, it needs to be lightweight -- introducing minimal overhead. These constraints will ensure that the publication and discovery mechanism conforms to MANET's characteristics. The mechanism should also allow the publication and the discovery of various types of interfaces. In fact, a business model is formed by several business roles and eventually different interfaces between these roles. The last requirement is that the publication and discovery mechanism should support both *push* and *pull* modes. This provides more flexibility to the model and enables an appropriate usage of the scarce resources of a MANET.

3.1.3 Requirements for service execution architecture and corresponding communication mechanism in stand-alone MANETs

This sub-section presents two sets of requirements. The first is related to the architecture for service execution. The second set is associated with the communication mechanism between the architecture's components.

3.1.3.1 Requirements related to the service execution architecture

Since MANETs are open networks where any node offering any functional entity can enter the network at anytime, the first requirement is that the service execution architecture should allow one or more service execution environments to coexist and cooperate in the same network.

Furthermore, the architecture must enable self-organization in order to support the frequent node mobility (i.e. nodes joining) that is part of the dynamicity of MANETs.

Next, the architecture should enable self-recovery to overcome the changing topology. Random failures of the service execution environment's components (e.g. node crashes, nodes leaving deliberately, batteries down) are an especially common occurrence. The architecture should be scalable in terms of the number of the service execution environments it maintains. Finally, the solution should be simple. It should neither take too much time to set up nor be resource-demanding, given the resource constraints of MANETs.

3.1.3.2 Requirements related to the architecture's communication mechanism

First, the protocol should be distributed for peer-to-peer communication. Basically, all the MANET nodes are equivalent (i.e. each node plays the role of a client, a server and a router). Furthermore, centralized control entities are unrealistic for MANETs -- the most suitable communication paradigm in this environment is distributed peer-to-peer.

Second, the protocol should allow self-organization and self recovery in a systematic way. Indeed, MANETs are highly dynamic environments where nodes leave and join frequently --- almost continually changing the topology and the connections between nodes. Therefore, the protocol should maintain the architecture structure with no effort or human interaction. Furthermore, nodes may crash suddenly. The protocol should be able to recover from this situation, making the architecture robust and flexible.

In addition, the protocol should be lightweight. It should be possible for small handheld devices with limited resources to host the protocol.

Furthermore, it should be simple, so that resources and time are used efficiently. Complex protocols with complex functionalities and management are too resource-demanding and introduce long delays. Finally, the protocol should scale to the number of the architecture's nodes, so that the performance should not decrease drastically when the network becomes large.

3.1.4 General and specific requirements for service provisioning architecture in MCNs

We first derive general requirements for service provisioning in Multihop Cellular Networks (MCNs), and then the requirements for a specific and concrete service provisioning architecture. The specific architecture is based on the SIP servlets framework for service provisioning. Furthermore, the 3G/MANET integration is done so that end-users are in the MANET portion and can access services in 3G network.

3.1.4.1 General requirements for service provisioning in MCNs

The first requirement is that the architecture should allow service invocation and service execution regardless of the location of the end-users. Users in a MANET and users in 3G should be able to invoke the same service. This will ensure that 3G network services and services provided in a MANET can be accessed by all subscribers: the overall goal of integration. Following from the first requirement, the second is that the architecture should allow users in 3G to discover available services in the MANET, including those services provided by individuals. These discovered services should then be able to be executed in a MANET or in a 3G to allow as much flexibility as possible. In fact, the network operator may prefer to run a service in a MANET rather than in 3G, or vice versa, depending on the network load or the application type. The architecture should also scale in terms of the services that can be executed simultaneously, since in the MCN, both MANET and 3G users can access and run services. The last requirement is that the service integration should have minimal impacts on MANET and 3G networks. It is always preferable to have fewer extensions and expend less effort to achieve them in order to avoid negative impacts on the overall system.

3.1.4.2 Specific requirements for a SIP servlets-based service provisioning architecture in MCNs

The first requirement is that the architecture should allow the SIP Servelts Engine (SSE), as a service execution environment, to be provided either by 3G service providers or by individual end-users who are in the MANET portion. Allowing individual users in MANET to provide the SSE will open the network to new business opportunities and potentially increase the network capacity regarding the service execution process.

The second requirement is that several SSEs should be able to coexist in the MANET, for scalability, higher flexibility, and to enable fault tolerance. Furthermore, the architecture should allow centralized as well as distributed SSEs. In fact, any entities providing SSE functionalities should be able to take part in the network.

Another requirement is that the service execution environment configuration (i.e. centralized or distributed) in the MANET should be as transparent as possible to the 3G AS in order to keep the AS independent from the MANET configuration. The fifth requirement is that the architecture should introduce minimal impacts on the 3G and MANET networks.

The last requirement is that the AS should know the location (i.e. MANET or 3G) of the end-users in order to decide where to run the service, since the service request can be sent by users in a MANET or in a 3G and we are in the context where all the end-users involved in a service execution are in the MANET portion.

3.2 Critical review of the state of the art

We have derived several sets of requirements that target service provisioning process in MANETs: for both stand-alone MANETs and integrated 3G/MANETs. The existing solutions will be analyzed and discussed from the perspective of the defined criteria, and for each category of requirements. To this end, the current services architectures, business models and service execution frameworks are discussed. There is, however, a lack of research in the service aspects of Multihop Cellular Networks (MCNs). To our best knowledge there is no existing solution for integrating 3G networks and MANETs in order to provide value-added services.

3.2.1 Service architectures

Two categories of service architectures for MANETs can be defined: the classical solutions and the emerging solutions. In first category we consider architectures designed without MANETs in mind, such as the service architecture of TINAC-C, Intelligent Networks (IN), Wireless Application Protocol (WAP), Parlay, web services and the service architecture of IMS. The second category contains a new model for 4G [65] and the I-centric model [66]. We will review both categories according to the general requirements presented in section 3.1.1.

3.2.1.1 Review of the classical service architectures

None of the architectures of the classical solutions meet all our requirements for service architectures in MANETs.

3.2.1.1.1 TINA

Telecommunication Information Network Architecture (TINA) is the first architecture for telephony and non-telephony services. It relies on basic concepts that are widely used (e.g. service lifecycle and business model). TINA is envisioned as a uniform infrastructure. It is based on four principles: object-oriented analysis and design, distribution, decoupling of components, and separation of concerns [67].

In relation to our derived requirements, TINA was the first system to introduce the notion of service lifecycle – it considers the whole phases of the service lifecycle. However, TINA depends in large part on a centralized architecture and a pre-established infrastructure. In fact, a telecommunication network, with its centralized servers and entities, is a prerequisite for TINA service architecture. Consequently, distributing functional entities is not a priority or even a necessity. Indeed, several servers are employed by the service architecture (e.g. TINA information repository). Furthermore, the functional entities are kept centralized (e.g. subscription management). Service discovery in TINA service architecture is performed in a way that is inadequate for MANETs. Indeed, the services descriptions are stored in a central server, and, since TINA service architecture is based on central and powerful entities, there is no need to limit resource usage. Not surprisingly, the architecture's processes are resource-consuming. In addition, TINA architecture is deployed in a controlled network. Therefore, it deals with homogeneous devices with the same configuration and platform.

Finally, the TINA business model is not flexible. The business roles are well known and cannot change their roles dynamically. For example, the *broker* business role cannot provide the functionalities of a *retailer* once the architecture is deployed.

3.2.1.1.2 Intelligent networks

The key concept of Intelligent Networks (IN) is the standardization of the capabilities for building services. In fact, IN describes Service Independent building Blocks (SIB) as capabilities that can be combined in different ways to create a wide range of telephony services. The IN defines the Service Control Points (SCP) in which Service Logic Programs (SLP) run. New services are realized using the building blocks. [68].

However, when evaluated according to our requirements, intelligent networks do not consider the whole service lifecycle. Indeed, service description, service discovery and service publication are beyond the scoop of IN. Furthermore, IN are based on an existing infrastructure such as the physical plane. Furthermore, the IN architecture contains several centralized entities. Obviously, given the central entities, functionalities are not fully distributed. As in TINA, the users must subscribe to the desired services. Discovery and publication of services is not incorporated. In addition, IN entities are high consumers of network resources. IN are based on circuit-switched networks, and so the optimization of resource consumption was not anticipated. Furthermore, the devices should have the same configuration and be of the same type. Finally, the IN business model is not flexible, as roles are defined and deployed initially, and are not subsequently modified. .

3.2.1.1.3 Wireless application protocol

The Wireless Application Protocol (WAP) arose from an industry specification for developing applications that can be used over wireless networks. The WAP architecture includes access points between wireless and wired networks as well as different proxies. The provisioning framework ensures connectivity and disseminates application information. However, WAP devices need to know about the architecture's components in order to use the services they provide. Thus this requires a trusted relationship with the infrastructure [69]. Figure 3.1 presents the basic WAP architecture.

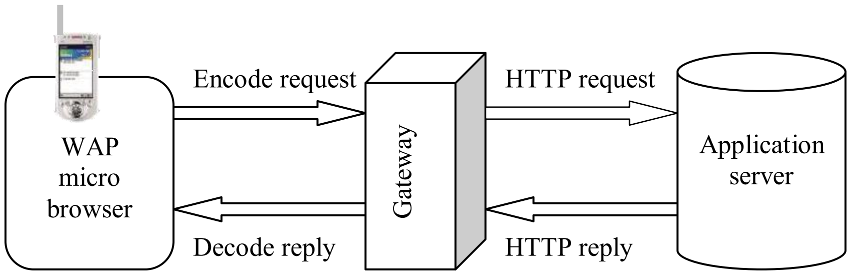


Figure 3.1: Basic WAP architecture

As for our requirements for service architectures, WAP does consider the whole service lifecycle. However, its architecture rests on an application server and a gateway, which constitute central entities, and so the functionalities are not fully distributed. It also relies on a telecommunications infrastructure. Furthermore, WAP architecture assumes that users know in advance the services they wish to access. There are no service discovery and publication mechanisms. WAP does meet our requirements for the optimal usage of resources since it is designed with small devices in mind. However, the architecture is deployed in a very controlled network where homogeneous devices with the desired configuration are targeted. In addition, the roles of the WAP business model are fixed, so that nodes are dedicated to a given role and cannot dynamically change their role.

3.2.1.1.4 Parlay

While applications and services have been located in the operator domain, some research groups such as JAIN [70], [24] and Parlay [71], [25] have worked to open up telecommunication networks to a large community. The logical architecture of Parlay identifies four distinct entities [72]: application servers, Service Capability Servers (SCS), the Parlay/OSA framework, and the core network elements. Figure 3.2 illustrates an overview of the Parlay/OSA logical architecture.

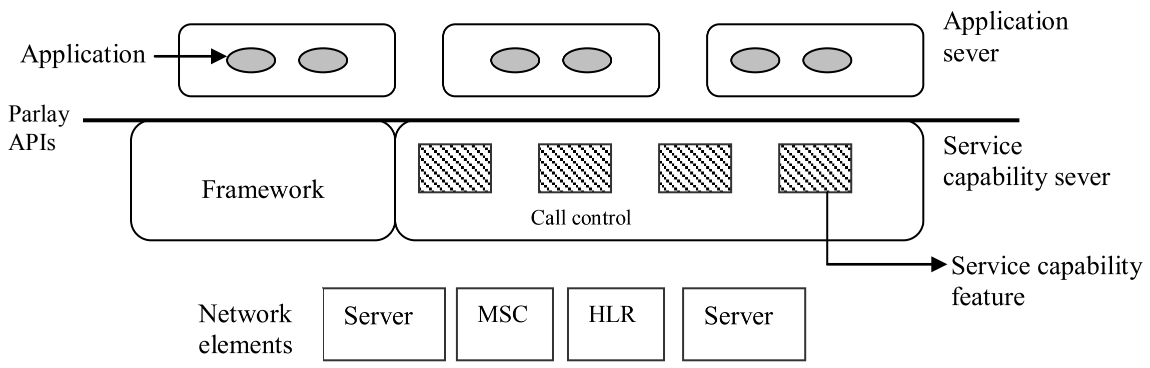


Figure 3.2: Parlay/OSA logical architecture

In Parlay, services refer to network capabilities and not end-user services. Applications are deployed on the application servers and can use Parlay APIs to access the capabilities provided by the service capability servers. It should be noted that the framework controls the access to the service capability features.

Parlay does not meet our requirements for service architecture for MANETs. To begin with, Parlay does not consider the whole service lifecycle. Furthermore, it is a centralized architecture, as it relies on the framework and the Service Capability Servers (SCSs). Theses entities are owned by the network operator and are fixed and centralized nodes. Furthermore, communication in Parlay is Client/Server-based. All the functionalities are centralized in the framework and the SCSs which contain the Service Capability Features (SCFs). In addition, because there is no service description in Parlay, there is no means to advertise services. End-users don't handle service advertisements in Parlay – they subscribe to services.

Parlay architecture assume a reliable network and a continual connection between nodes. It does not take into account devices with scarce resources. Parlay architecture is heavy, mainly because it relies on heavy middleware (e.g. CORBA, Java RMI). Furthermore, in Parlay there is no way to distinguish between devices that constitute the network and those that use services. It treats them in the same way and so each device must conform to the architecture's requirements. As for the Parlay business model, it was conceived for fixed environments and so dynamic roles are not allowed. Furthermore, roles can only be provided by the network operator.

3.2.1.1.5 Web services

The concept of web services [73] stands for a new generation of web applications. It provides a systematic and extensible framework for application-to-application interaction, built on top of existing web protocols and based on open XML standards [74]. A web service is any application that can be published, located and invoked through the Internet.

Web services architecture is based on three fundamental principles: a coarse-grained approach, loose coupling, and both synchronous and asynchronous modes of communication. Put differently, for scalability and efficiency concerns web services architecture should offer high-level interfaces, applications would have minimal interdependencies and the communication system should take into account an application's unavailability [75].

The web services' architecture is based on three entities: the service registry, the requester, and the provider. Providers publish service descriptions in the service registry. Then, requesters can query the registry and get a list of available services that match the query. Finally, requesters choose a service, bind it to the appropriate provider and start using the service.

Furthermore, there are three main parts in the web services architecture [74]: communications protocols, service description and service discovery. These areas are based on different technologies and standards. The communication is based on the Simple Object Access Protocol (SOAP) [76]. The Web Services Description Language (WSDL) [51] is the formal language for web services description and the Universal Description Discovery and Integration (UDDI) [77] repository hosts the service descriptions and then implements the service registry entity. Basically, SOAP is an XML-based protocol for messaging and remote call procedures. It works on top of existing transport protocols such as HTTP, Simple Mail Transfer Protocol (SMTP) or File Transfer Protocol (FTP). WSDL defines an abstract description of services. It is an XML file that contains all the appropriate information for accessing a web service, including location, protocols, message format, and the operations provided by the web service. UDDI is a centralized registry that provides requesters a unified and systematic way to find services.

Web services cannot meet our requirements for service architectures for MANETs. We should mention that web services architecture covers almost all of the service lifecycle except for service withdrawal. Furthermore, web services' specifications allow distributed as well as centralized architectures. Consequently, the functionalities may be distributed. However, there is no mechanism to deal with unforeseen disconnections. Therefore, web services require continual connections. One of the problems that could arise is as follows: a requestor discovers a service and identifies its provider as being at location ***A***. Next, the requestor will try to bind to the provider who perhaps has moved to location ***B*** or is out of communication range. Services can be discovered, but may no longer be reachable. Therefore, the publication and discovery mechanism is not adequate for MANETs.

Although there are some solutions for using web services in the wireless world [76] and [77], with the capabilities and heterogeneity of wireless devices, ad-hoc networks have not yet been considered. The problem with theses solutions is that they only target small devices and bring different platforms for these mobile devices. Finally, the same argument can be made about web services business models as for the previous business models: they are not flexible and individuals are not allowed to provide a business role (except for end-users' roles).

3.2.1.1.6 The service architecture of the IP Multimedia Subsystem

The IP Multimedia Subsystem (IMS) [80] is a 3GPP/3GPP2 standardized architecture of the Next Generation Network (NGN). It aims at filling the gap between the cellular and the Internet worlds. In fact, IMS allows operators to take advantage of the quality and interoperability of telecoms and the innovative development of the Internet [81]. IMS defines a service architecture which can be based on Parlay, web services or SIP servlets.

Along with the drawbacks of Parlay and web services mentioned previously, the service architecture paradigms of IMS share common shortcomings. They cannot meet our requirements for MANETs.

The whole service lifecycle is not supported by the service architecture of IMS. Furthermore, the architecture provides no means for service publication and discovery. End-users must register for given services. It is clear from the architecture that IMS service architecture rests on centralized nodes and pre-established infrastructure. Furthermore, in IMS each functional entity is providing multiple functions -- functionalities are not fully distributed. As mentioned before, there is no adequate mechanism for service publication and discovery. End-users know about off-line services through traditional commercials from their operator or from service providers. In addition, IMS architecture is not optimal when it comes to resource consumption. The core network uses powerful links and data rates, so the processing within the network is very heavy and there is no need for savings in the current context. However, IMS can be accessed regardless of the device type, provided it supports SIP. Finally, the IMS business model does not allow roles to be dynamically provided and discovered.

As shown in this sub-section, the classical service architectures fail to meet the requirements for a MANET service architecture. Table 3.1 summarizes the shortcomings of the existing solutions.

Requirements / Architectures	TINA	IN	WAP	Parlay	Web Services
Consider the whole service lifecycle	Yes	No	Yes	No	No
Central entity, infrastructure	Yes	Yes	Yes	Yes	*Optional*
Distributed functionalities	No	No	No	No	*Optional*
Adequate publication/discovery mechanisms	No	No	No	No	No
Difference between devices	No	No	No	No	No
Optimal usage of resources	No	No	Yes	No	No (but some research)
dynamic business models	No	No	No	No	No

Table 3.1: summary of the shortcomings of the classical service architectures

3.2.1.2 Review of the emerging service architectures

3.2.1.2.1 The emerging 4G model

Fourth generation (4G) networks work with heterogeneous network technologies and try to seamlessly integrate them. The integration of these technologies is usually done at the control and connectivity level. The main objective of the proposed model in [65] is the integration of applications and services in 4G networks.

This model has many interesting characteristics: it is extensible; it allows services and applications to be of "write once, run anywhere" type, and it targets the Personal level networks as well as home/local level and cellular level networks. Personal area, body area and ad-hoc networks are defined at the personal level, Wireless LANs are defined at a local level and UMTS and 3G technologies are at the cellular level. Figure 3.3 illustrates the model.

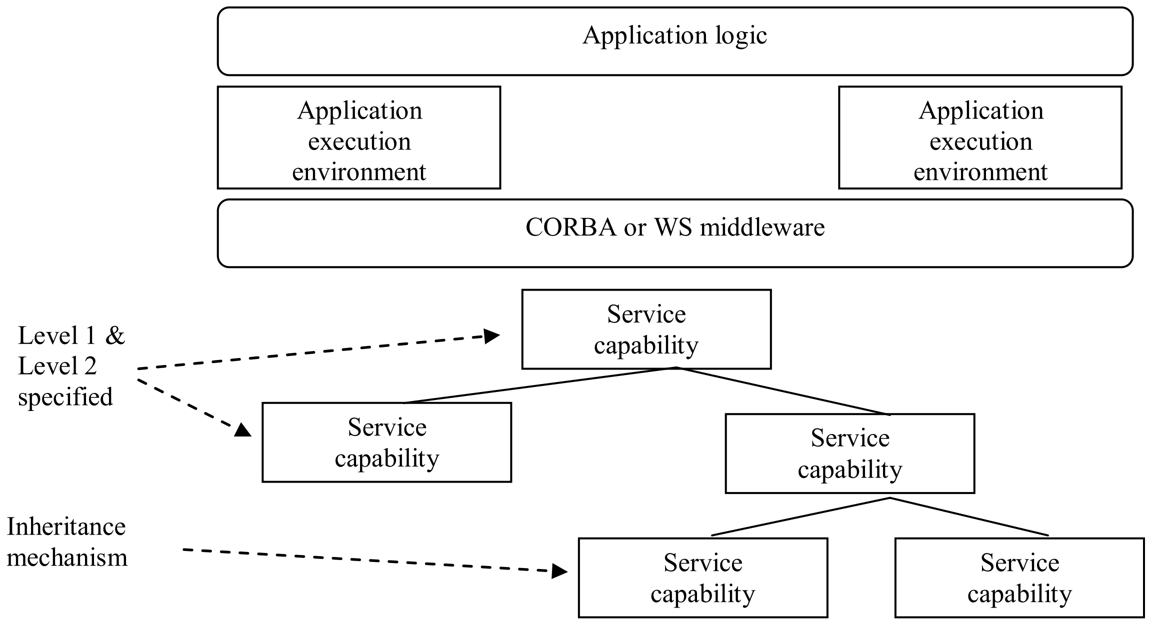


Figure 3.3: A proposed model for services and applications in 4G networks

This model relies on two concepts to achieve its goal: the separation between the application logic and the application execution environment, and the definition of a service capabilities hierarchy. The separation is necessary to develop device-independent applications, and the service capabilities hierarchy makes the model extensible and open to new additions. Two levels of service capabilities are specified within the model, and inheritance mechanisms can be used to add additional levels to the hierarchy.

The first level defines the common capabilities of all services available in 4G networks, while the second level captures the common functionalities of technological families (personal level, home/location level and cellular level networks) inside 4G networks.

Even though this model addresses 4G and MANET networks at the service layer, it cannot meet our requirements for service architectures. In particular, it focuses on service creation and service description and thus fails to consider the whole service lifecycle. In addition, service creation is based on a limited set of basic capabilities. This set may not be complete and therefore, the model is not suitable for any new type of services. Lastly, although the model targets various types of networks, ranging from ad-hoc networks to cellular networks; it is too general to address specific ad-hoc requirements.

3.2.1.2.2 The I-centric model

The authors in [66] propose an I-centric model that puts the individual user in the center of service provisioning. It is a reference model that addresses various issues. The objective is to develop a communication service infrastructure that will take into account each individual's environment with his/her preferences and adapt services to different situations and resources in real-time. The reference model for I-centric communications is presented in Fig.3.4.

The major concepts behind the I-centric model are: a high level of consideration for individual users, a flexible and dynamic business model and the adaptation of services based on ambient awareness and user personalization. Three notions are central to this model: personalization, ambient awareness and adaptation. The goal of personalization is to make service usage easier and to enable tailored services. The purpose of ambient awareness is to gather and use information about the context and the situation around an entity. Adaptation means the ability of services to change their behaviour when circumstances in the execution environment change.

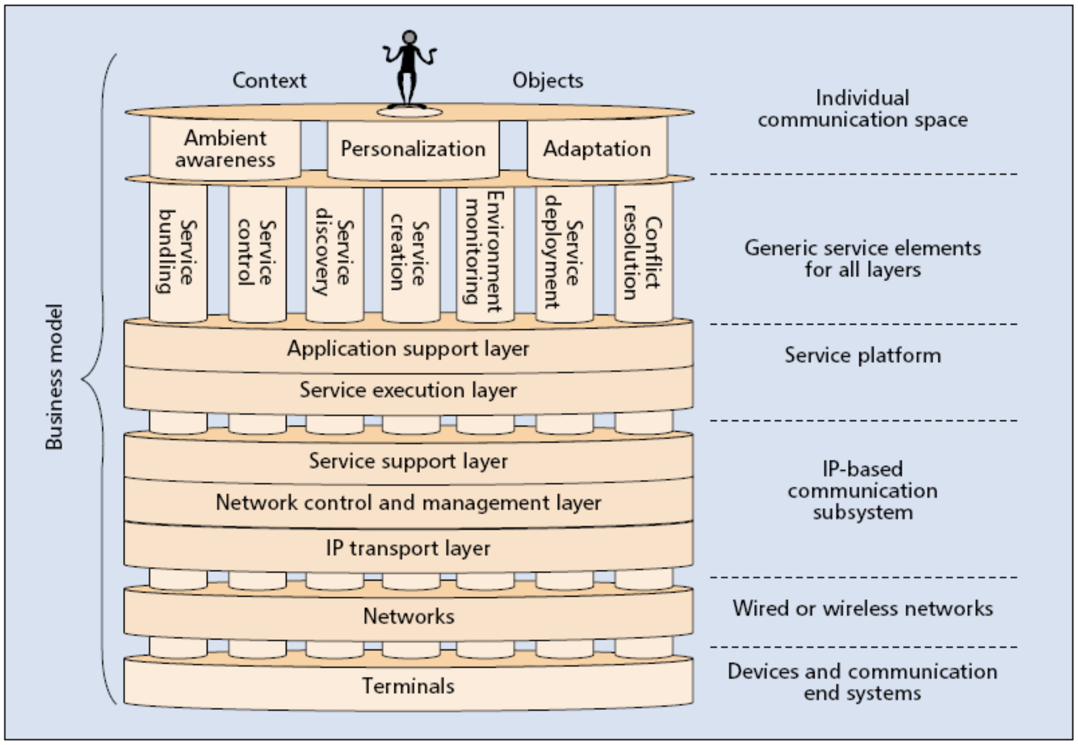


Figure 3.4: The reference model for I-centric communications

The model is very ambitious. However, it is defined at a very high level of abstraction with no concrete algorithms and protocols to show how it can be implemented in an ad-hoc network. Thus far, the I-centric model's concepts have remained at the theoretical level.

3.2.2 Business models and related publication/discovery mechanisms

This section describes the major business models in use today. A critical review is provided for each business model according to the requirements of sub-section 3.1.2.1.Next, the main service publication/discovery mechanisms are discussed based on the requirements of section 3.1.2.2.

3.2.2.1 Business models

In the following sub-section, we present some of the predominant business models:+ TINA-C, web services, parlay/OSA and IMS.

3.2.2.1.1 TINA-C business model

The TINA-C business model is detailed in [54]. The model defines five business roles and the interfaces between them. Figure 3.5 presents an overview of the TINA-C business model.

The *consumer* business role can be either the user of the service or an entity with an agreement for service usage (a subscriber); this role pays for using the available services and is the economical base of a TINA system; The *retailer* is a service provider that has an agreement with the *consumer*, it offers its own services or subcontracted ones; the *broker* business role has the responsibility of fairly providing all the parties with the information required to discover each other and to find services in the TINA system. The *third party service provider* has a business agreement with the *retailer* but no direct communication with the consumers; it supports retailers and other third party providers with services; and the *connectivity provider* owns and manages the overall network.

In order to allow these roles to interact, TINA-C describes a collection of reference points. The Reference Points comprise a set of interfaces describing the interactions taking place between these roles. For example, the standardized reference point *Retailer* (*Ret*) [82] and [81] describes the relationship between the consumer and the retailer. The remaining reference points are: Broker reference point (*Bkr*), Connectivity Service reference point (*ConS*), Client-Server Layer Network reference point (*CSLN*), Layer Network Federation reference point (*LNFed*), Retailer-to-Retailer reference point (*RtR*), Terminal Connectivity reference point (*TCon*), and third-Party service reference point (*3Pty*). Figure 3.5 presents an overview of the TINA-C business model.

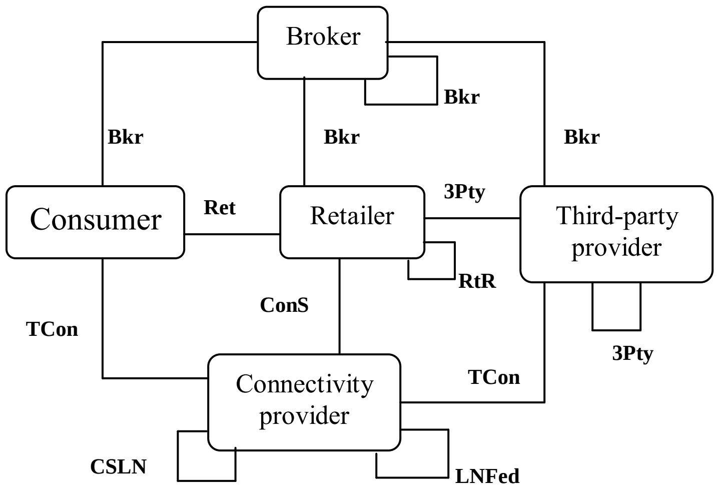


Figure 3.5: TINA business model and reference points

The TINA-C business model does not meet any of our requirements. The retailer, the broker and the connectivity providers play central roles because their lack or disconnection causes the overall system to fail. The consumer and the third-party provider do not affect the functioning of the system when they are not present. TINA business entities can participate in different roles at the same time but, once established, they are not allowed to change their role. Furthermore, by adopting the notion of an administrative business domain that belongs to the enterprise viewpoint, TINA-C makes some roles dependent on organizations. The communication paradigm used in this model is the client/server. TINA-C provides a service discovery via the *Ret* reference point, but it is a very heavy process. It is part of a complex procedure for service access and usage [84]. In addition, the discovery mechanism does not take into account mobility, disconnections or resource constraints. The TINA-C business model is thus not suitable for ad-hoc networks.

3.2.2.1.2 Web services business model

Web services business architecture defines three business roles: the *service requester*, the *service provide*r and the *service registry*. Three types of operations are described to illustrate the relationship between these roles (*publish*, *find* and *bind*). Figure 3.6 presents a global picture of the web services business model.

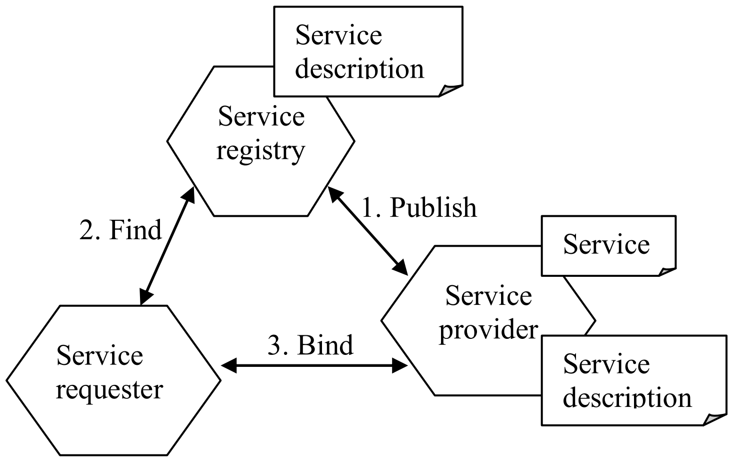


Figure 3.6: Web services business model and its primitives

The *service requester* is the entity interested in using a web service. It uses an agent to interact with the service registry agent in order to discover services, and interacts with the *service provider* agent to make use of the service. The *provider* is the owner of the web service and wishes to share that service with other entities. It can either use services from other providers to construct a new web service or offer entirely local ones. The *service registry* or the broker is a standardized database of service descriptions. It allows providers to publish their web services using the *publish* operation, and requesters to find the desired services using the *find* operation. The *bind* operation is used after the reception of the service description to contact the selected service provider and invoke the preferred service.

The web services business model has interesting characteristics but it does not meet all our requirements for ad-hoc networks. The web service architecture can operate in a centralized way, or have all roles distributed and thus remove the central entity problem. Still, these roles are predefined and entities are not allowed to change their roles dynamically. The World Wide Web Consortium (W3C) specifications define a peer-to-peer approach for web service discovery and usage. As a consequence, organizations are no longer the only possible domains for business roles. Individuals can form a web service architecture operating in peer-to-peer mode with no central entity. However, the underlying publication/discovery mechanism is not suitable for ad-hoc networks. It is clear that the centralized discovery approach is not adequate in such an environment. On the other hand, the distributed approach still has some performance and reliability drawbacks [85]. Furthermore, web services architectures are mostly deployed in their centralized form. Thus web services business model is unsuitable for MANETs.

3.2.2.1.3 Parlay/OSA business model

The Parlay business model is widely inspired by the TINA-C model and contains three main roles: the *client application*, the *enterprise operator* and the *framework operator*. Figure 3.7 illustrates the different business roles and how they are related.

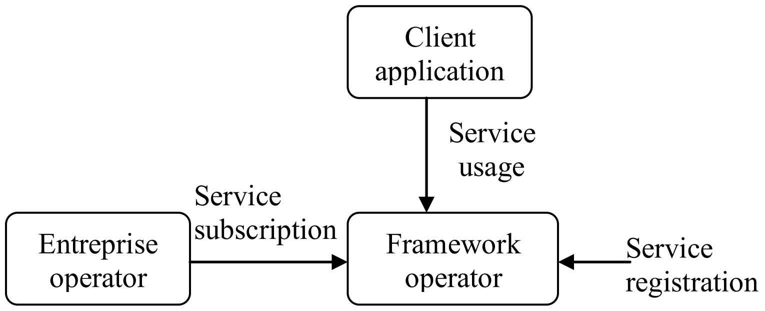


Figure 3.7: Parlay business model

The client application role is the one that consumes the services (network capabilities). It is equivalent to the end-user in the TINA-C business model. The enterprise operator is the entity that subscribes to the services. It has a business agreement with the framework for service usage. This role is equivalent to that of the subscriber in TINA-C. The framework operator provides the initial contact point to the client application to discover the capabilities offered by the network and allows the network' operators (the real service providers) to negotiate with service users and subscribers. The framework operator is equivalent to the retailer in TINA-C. We note here that the service providers (network capabilities providers) are not explicitly considered in the Parlay business model.

This business model cannot meet our requirements. It was designed without ad-hoc network characteristics in mind. Indeed, Parlay rests on a pre-established infrastructure and central role/node (e.g. the framework operator). This represents a serious drawback in a temporarily highly dynamic environment such as ad-hoc networks. The Parlay architecture does not allow entities to switch their role during the execution. In fact, in order to play more than one role, Parlay nodes need to be configured with these roles in advance. Furthermore, except for the client application, roles cannot belong to individuals for security and resource-constraint reasons. The communication between Parlay entities is performed in a client/server fashion and the broker relies on classical CORBA, which makes it heavy-weighted for handheld devices. All of these aspects show that the Parlay business model is unrealistic for ad-hoc networks.

3.2.2.1.4 IMS business model

The IMS business model has three roles: the *end-user*, the *service provider* and the *network operator*.

The end-user owns User Equipments (UE) -- a UE is a client that implements the necessary logic to access, invoke and use services. The service provider owns the application server, as in most deployed IMS systems. The network operator is the owner of the network infrastructure (IMS control nodes). Basically, end-users invoke services through the 3G control nodes while the service providers deploy services in the AS. Later, these can be located either in a third party or in the network operator domain. Figure 3.8 presents the IMS business model for 3G.

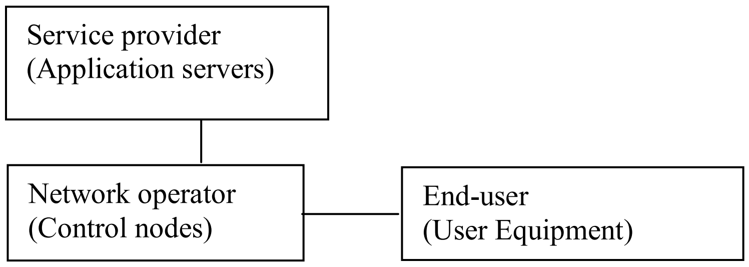


Figure 3.8: IMS business model for 3G networks

Relevant to our requirements, the IMS Business model relies on central units such as CSCFs, the MRFC and the AS. It is also based on a well-controlled and pre-established infrastructure. The locations of the functional entities are known beforehand. No dynamic discovery of the functional entities or roles is provided for. Furthermore, it is improbable that an IMS business model could rely on individuals with small devices. First, the service providers and network operators have no strict resource constraints. Second, the IMS roles are supplying too many functions that are resource-intensive. For example, the SIP AS supplied by the service provider contains services/applications, SIP servlets and the SIP servlets engine. For security reasons, the network operator functional entities cannot belong to individuals. In addition, the communication between the different roles is based on the client/server paradigm and the service discovery process is out of the scoop of IMS. We have clearly shown that none of the main business models is suitable for MANETs.

Table 3.2 summarizes the review of the abovementioned business models.

Requirements / Business models	TINAC	Parlay	Web services	IMS
Central role/infrastructure	Yes	Yes	Optional	Yes
Flexibility	No	No	No	No
Reliance on individuals	No	No	Optional	No
Peer-to-peer communication	No	No	Optional	No
Adequate broker	No	No	No	No

Table 3.2: Summary of the review of the main business models

3.2.2.2 Publication and discovery mechanisms

The publication mechanism relies on two main functionalities: *service description* aims mainly at defining what the service does, how it can be used and from where it can be invoked. This is done using an unambiguous and well-known syntax; *service advertisement* allows service descriptions to be reachable by anyone so that users are made aware of the existence of the services. Descriptions can be either advertised in a service directory or directly to the other hosts. The discovery mechanism implies three key functionalities: *request formulation* which reflects the user's needs and should conform to the service description; a *matching function* that maps requests to equivalent services; and a *communication mechanism* for the interaction between the requester and the provider [55]. An appropriate publication and discovery mechanisms is crucial to the success of the service architecture, especially in a dynamic environment with limited resources such as MANETs.

From the MANET perspective, the existing service publication and discovery solutions can be grouped into three categories: routing-based mechanisms, directory-less mechanisms and directory-based mechanisms.

3.2.2.2.1 Routing-based solutions

These solutions extend and use MANET routing protocols to publish and discover services. Basically, the service messages are piggybacked onto the routing protocol. Examples are the *Lightweight Service Discovery* (LSD) [90] that extends the *Optimized Link State Routing* (OLSR) protocol, the *Zone Routing Protocol* extension [91], *anycast* [92] that extends the *Ad hoc On-demand Distance Vector* (AODV) routing protocol, and a service discovery architecture [89] based on the *Hexell* routing protocol. This last solution uses information from the routing protocol for a service selection that takes into account the network's context.

These solutions cannot meet all of our requirements. Even if they are designed for MANETs and meet several requirements (e.g. lightweight, low overhead, distributed) they fail to meet one important criterion. Indeed, the first and most important requirement is independence from the routing protocol. Without this independence, users and providers in different MANETs using dissimilar routing protocols cannot talk to each other. Furthermore, the above-mentioned mechanism does not consider the *push* and *hybrid* means of service discovery.

3.2.2.2.2 Directory-based solutions

The service publication and discovery mechanisms that use a registry belong to the category of directory-based solutions. Basically, service providers store the services they are willing to offer in a centralized or a distributed directory. MANET users can query the directory to discover the available services.

Several standards have been proposed with a centralized directory. The main ones are the *Service Location Protocol* (SLP) [90], *Salutation* [91], *Jini* [88], *Universal Plug and Play* (UPnP) [89] and, from web services, the UDDI [77]. However, these solutions were designed for fixed and controlled networks and are unrealistic for MANETs. In fact, they are based on a central registry: Directory agent in SLP, salutation manager in *Salutation*, look service directory in *Jini*, control points in UPnP and UDDI in web services. Furthermore, most of these mechanisms rely on heavy protocols (e.g. RMI, SOAP) and only the pull discovery is authorized.

However, mechanisms based on fully distributed directories are interesting solutions for MANETs. Their main advantages are: scalability, rapid service discovery and load balancing. Examples of such mechanisms are: Sailhan's scalable service discovery (SSD) [94] and the *Distributed Service Discovery Protocol* (SDSP) [95]. Neither solution considers the push mode of discovery, and, very important, maintaining a set or distributed directory comes at a cost of extra processing and management. Furthermore, since we are not targeting very large scale MANETs, we believe their cost is not justified for our task.

3.2.2.2.3 Directory-less solutions

This category regroups the publication and discovery mechanisms that do not rely on a directory. Providers store their services in a local and logical registry. Users multicast or broadcast queries and receive responses from providers within the communication range.

This means of service discovery ensures that only services that are available at that moment can discovered. This is an important advantage in highly dynamic networks.

The major service publication and discovery mechanisms in this category are *Konark* [96], *DEAPspace* [96], *Pervasive Discovery Protocol* (PDP) [98] and *Linda In Mobile Environment* (LIME) [99].

Konark is a middleware package for service discovery and delivery, designed for ad-hoc, peer-to-peer networks. The service discovery is based on a fully-distributed mechanism with a cache that allows devices to publish and discover services in the network. *Konark* uses an XML-based service description similar to WSDL, and for service delivery *Konark* proposes a micro-HTTP server based on SOAP. We could think that *Konark* meets all of our requirements. However, its reliance on “mico-HTTP” and SOAP may threaten its lightweight aspect. Furthermore, *Konark* allows semantic searches which then increase the energy consumption, although the responses are more accurate.

DEAPspace targets very short range networks. Indeed, it was designed for single-hop ad-hoc networks. Basically, *DEAPspace* is a solution where all the devices keep track of all known services, called “world view”. Periodically, the devices broadcast their “views” to their neighbors. DEAPspace does not meet our requirements since it is a pure push mechanism. It introduces a large overhead by broadcasting the whole list of services to all of the neighbours and so it is only practical in a very small network.

The *Pervasive Discovery Protocol* (PDP) is a lightweight protocol designed especially for ad-hoc networks. PDP does away with the need for any central entity, and supports the push and pull methods in a straightforward way. One of the main objectives of the PDP protocol is to reduce traffic in the network by minimizing transmissions.

Consequently, The PDP conserves both network bandwidth and devices' resources, particularly for those devices that are very limited. These are central properties for protocol efficiency in ad-hoc networks. Each device is assigned an availability time (the excepted time that the device would remain in the network), a local and a remote memory cache (where owned and discovered services, respectively, are stored). The PDP is based on two agents that discover available services and publish owned services, respectively. The PDP meets all our requirements and is therefore a good candidate for service publication and discovery.

Linda In a Mobile Environment (LIME) is a middleware that extends the coordination model of *Linda* [100]. *Linda* is a fully distributed, in time and space, programming language where programs are a collection of ordered *tuples*. LIME is a coordination middleware that utilizes logically mobile agents running on physically mobile hosts. LIME is based on the concept of *tuple space*. Publication and discovery using LIME is achieved trough the manipulation of *tuple space*: writing and reading from the LIME *tuple space*. LIME has interesting characteristics. It does not rest on any infrastructure, data exchange is time and space independent, mobility is addressed, security issues are dealt with and it allows both pull and push scenarios. Therefore, it meets all of our requirements and is a good candidate for service publication and discovery.

Thus far, PDP and LIME are both promising solutions for developing an appropriate service publication and discovery mechanism. However, a close comparison between PDP and LIME shows that PDP has more advantageous characteristics than LIME. Table 3.3 summarizes this comparison.

characteristics / **Discovery mechanism**	**LIME**	**PDP**
Fully distributed	++	++
Lightweight	+ (346 Ko)	++ (46 Ko)
Adapts to changes	++	++
Optimal usage of bandwidth	+	+++
Security	++	-
Pull and push	+	++
Considers differences in business interfaces	+	++

Table 3.3: Comparison between PDP and LIME

3.2.3 Service execution frameworks and corresponding communication model

This section presents the existing service execution frameworks and then reviews them according to the requirements proposed in 3.1.3.1. The communication model of these frameworks will be compared to the requirements of sub-section 3.1.3.2.

Service execution for MANETS has not yet been addressed in the literature. However, mature standards have been successfully developed for wired and infrastructure-based networks. In the following sub-sections we will review the Service Logic Execution Environment (SLEE) and JXTA as architectures for service execution.

3.2.3.1 Service Logic Execution Environment (SLEE)

SLEE is a well-known concept in telecommunications. It provides an operation system for service execution -- managing and coordinating the execution of services. JAIN SLEE (JSLEE) [101] is the Java standard and component model for SLEE. JSLEE defines four basic elements: resource adapters, events, activity contexts, and the runtime environment.

Resource adapters are responsible for communication with external resources. They receive and send events. When an event is received it is forwarded to the activity context as an object, and then forwarded to the runtime environment. This latter contains the Service Building Blocks (SBB) responsible for processing the events.

However, the SLEE does not meet our requirements, since it is intended for infrastructure-based networks with fixed and stable connections. The specifications do allow several instances of the basic elements of the SLEE to co-exist and cooperate. However, the SLEE does not deal with frequent disconnections and topology changes since self-organization and self-recovery processes are not provided. It is also resource-consuming. Finally, given that the SLEE is infrastructure-based it is not simple to set up. Therefore, it is not feasible to use in a MANET.

The communication between the basic elements consists of event delivery. Nevertheless, the specifications do not define how events are delivered. It is up to the vendors to decide which mechanism to implement.

3.2.3.2 JXTA

JXTA [102], short for *juxtapose*, is an open source platform. It defines a set of protocols that enable any device in a peer-to-peer network to communicate, collaborate and share resources. The platform is organized into three layers and comprises six protocols. The applications layer contains the end-users' applications such as instant messaging. The services layer includes functions commonly required by peer-to-peer environments such as search and indexing, protocol translation, file sharing and so on. The core layer encapsulates the essential primitives for peer-to-peer communication (e.g. discovery, peers, and peer groups). The six protocols are: peer discovery, peer resolver, Rendezvous, peer information, pipe binding, and, endpoint routing. Table 3.4 gives an overview of the JXTA protocols.

Protocol name	Description
Peer Discovery Protocol	Used to discover and advertise peers' resources
Peer Resolver Protocol	Sends a query/response to one or multiple peers
Rendezvous Protocol	Subscribes to a multicast group (propagation service)
Peer Information Protocol	Used to obtain the status information of a peer.
Pipe Binding Protocol	Used to establish a communication channel.
Endpoint Routing Protocol	Used to discover routes (sequences of hops).

Table 3.4: Overview of the JXTA protocols

JXTA does not meet our requirements. Indeed, service invocation in JXTA is not defined in the specifications. JXTA allows self-organization by arranging peers in groups. However, this organization is basically for the purpose of routing and not for service execution. In fact, in order to exchange messages peers must belong to the same group, which is not realistic in MANETs since it can engender high traffic from so much joining and leaving groups. In addition, JXTA does not deal with mobility and unstable connections. In regard to communication, JXTA proposes six protocols which can be too much for a MANET device. The communication model is complex and therefore resource-consuming.

3.3 Summary

In this chapter we have derived a set of essential requirements. Several sets of requirements were presented at different levels: service architectures, business models, service publication/discovery mechanisms, service execution architectures and service provisioning in MANETs and MCNs. We then introduced the relevant works for each level and compared it to our requirements. Consequently, we have shown that none of the existing solutions meets our requirements, except for the service publication and discovery mechanism. Furthermore, existing service provisioning solutions, as they are today, do not meet enough requirements to be considered for a qualitative comparison. Therefore, novel architectures and enhancements to existing frameworks will need to be proposed.

CHAPTER 4: Business model for service provisioning in stand-alone MANETs

Service provisioning in stand-alone MANETs requires a new business model. This chapter presents a novel business model. A general business model is introduced, followed by a refinement of that business model. Afterwards, the chapter presents a mapping between the proposed business model and the SIP servlets framework. The chapter continues by depicting the service description and service discovery mechanism used within the business model. Finally, it draws scenarios that illustrate how the proposed business model is applied to provide services in stand-alone MANETs.

4.1 General business model

This section proposes a general business model for service provisioning in MANETs. First, the different roles of the business model are proposed, followed by a discussion of the roles' interactions. Next, the required functionalities are presented. Finally, the proposed business model is compared to the requirements derived in chapter 3.

4.1.1 Roles of the general business model

To address ad-hoc network characteristics we propose four roles*: end-user*, *service provider*, *capabilities provider* and *execution environment provider*.

In our model, service means any end-user service that goes beyond a two-party voice call. Capabilities refer to the building blocks required to realize services, and execution environments are features that may be needed to run a service.

The *end-user* is the service consumer. It looks for services and invokes those desired. The end-user role does not deal with communication details and it accesses services in a transparent fashion.

The *service provider* owns the service logic. It maintains a list of available services to be offered to end-users. It may require other resources to build its services and should verify the availability of these resources.

The *capabilities provider* owns some service capabilities. It allows service providers to use them and maintains a list of available service capabilities.

The *service execution environment provider* offers its execution environments to the interested entities (i.e. service providers or capabilities providers). It is an important piece of the business model since it is the entity that runs services.

4.1.2 Interactions

In order to use a service within the proposed business model, the service, the appropriate capabilities and execution environment should all be available in the network. The end-user is responsible for discovering service providers and services. The service provider is responsible for discovering the capabilities providers. Then it has to discover the required capabilities for building the services it claims to offer. It is also responsible for initiating the service execution.

Furthermore, a service can be built with specific capabilities. Therefore, the capabilities provider needs an execution environment compatible with its capabilities. However, two different means of interaction are possible. The capabilities provider may be responsible for checking the availability of the appropriate execution environment, or it may delegate the verification to the service provider. However, we believe that it is more efficient in terms of interactions that a capabilities provider be responsible for checking the availability of the adequate execution environment.

Figure 4.1 gives an overall view of the general business model roles and interactions.

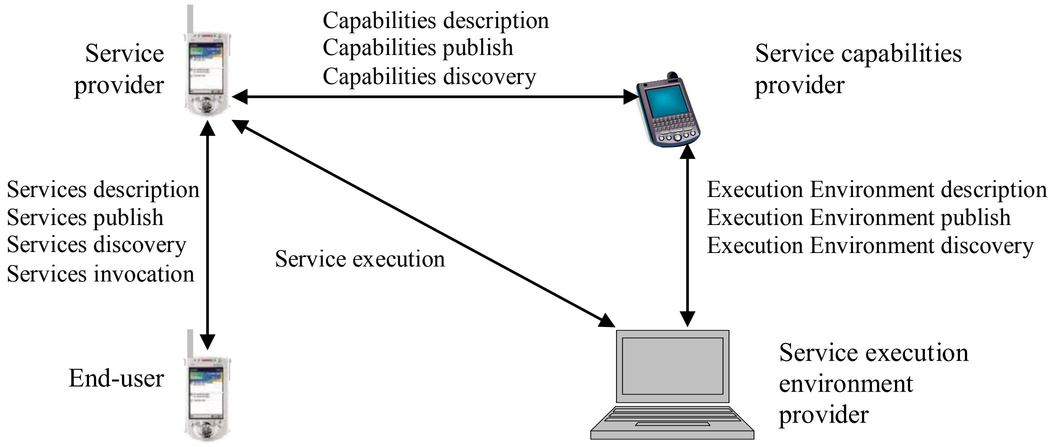


Figure 4.1: An overall view of the general business model roles and interactions

In order to execute a service, the roles' interactions are performed as follows: the end-user discovers the service's provider and then the services. When the service provider receives the end-user's discovery request, it discovers the corresponding capabilities. Upon receiving the discovery request, the capabilities provider discovers the adequate execution environment and replies to the service provider, which in turn sends a reply to the end-user with a list of available services (i.e. services for which capabilities and execution environment are available). At this moment, the end-user is able to invoke a given service. The service execution is then initiated by the service provider and executed by the execution environment provider. Furthermore, the discovery process can be performed in a pull or push mode.

Obviously, the different providers announce their presence in the network and publish their features.

As an example, figure 4.2 illustrates the interactions between the business model's roles in pull mode, where EU stands for End-User, SP for Service Provider, CP for Capabilities Provider, and EEP stands for Execution Environment Provider.

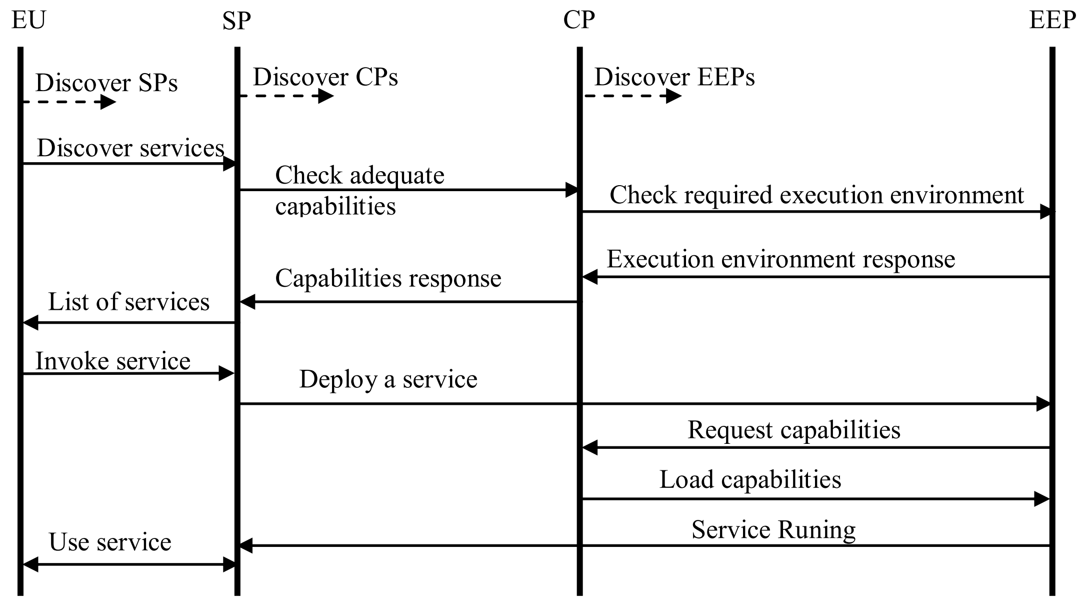


Figure 4.2: The general business model interactions in *pull* mode

The previous chapter showed that two publication and discovery mechanisms are suitable for MANETs. We use one of them to achieve the described interactions.

4.1.3 Required functionalities

From the above business model's description we can state that each role needs a set of functionalities in order to interact with the other roles.

In an ad-hoc network the end-user requires a discovery mechanism to discover available service providers and the services they offer. To ensure transparency, the end-user is not supposed to know about the other business roles.

The service provider requires a publication mechanism, a mapping function and a discovery mechanism to publish its services, map them to their needed capabilities and discover available capabilities providers and appropriate capabilities, respectively. The service provider publishes only those services for which the required capabilities are available.

Similarly, the service capabilities provider requires a publication mechanism, a mapping function and discovery mechanism to publish its capabilities, to map its capabilities to their execution environment and to discover the available execution environment providers, respectively. Only those capabilities for which a well-matched execution environment is available are published.

The execution environment provider requires a publication mechanism to publish the execution environment descriptions. To run the service, a module must interact with the service provider.

In order to describe their respective features, the service provider, the capabilities provider and the execution environment provider need a description language.

4.1.4 Discussion

The proposed business model meets most, but not all of our business model requirements, described in the previous chapter, for service provisioning in MANETs. In fact, it meets all of them except the lightweight requirement for all of the functional entities. The functional entities are distributed and do not rest on a pre-established infrastructure.

Furthermore, each entity can discover not only the features provided by other roles but also the role itself. For example, a service provider discovers the capabilities' providers and then the capabilities. This brings the flexibility to the business model that is required in a MANET. Roles can come and leave with minimal or no adverse impacts on service provisioning. Consequently, any entity can play a role at any time. Communication is performed in a peer-to-peer mode and the selected publication and discovery mechanisms are suitable for MANETs, as shown previously. In addition, the roles provide lightweight functionalities and can easily be provided by individuals, except for the execution environment provider role. The refined business model in the next section will solve this remaining issue.

4.2 Refined business model

The general idea behind the refined business model is to allow individuals with small devices to play any role. The execution environment may constitute a heavy entity. It can, however, be split into many entities; each of which may be a provider of the part of the execution environment it owns. A refined business model is then proposed, based on this possibility. The roles of the refined business model are presented in this section, followed by an exploration of the interactions between the business model roles and the required functionalities for each role to achieve its goal. Finally, the proposed refined business model is discussed according to the requirements from chapter 3.

4.2.1 Roles of the refined business model

The refined business model contains four roles: the *End-User* (EU), the *Service Provider* (SP), the *Capabilities Provider* (CP) and the *Execution Environment Sub-Part Provider* (EESPP). The EU, SP and CP are the same role as described in the general business model. However, a refined role is introduced -- the Execution Environment Sub-Part Provider. The EESP is the owner of a part or a component of the execution environment function. However, the EESP is transparent to the end-user, the capabilities provider and the service provider. In fact, the different components are offered by different providers, but the execution environment sub-part providers collaborate to offer the overall execution service environment. This latter is seen as a unique entity. Figure 4.3 presents an overview of the refined business model.

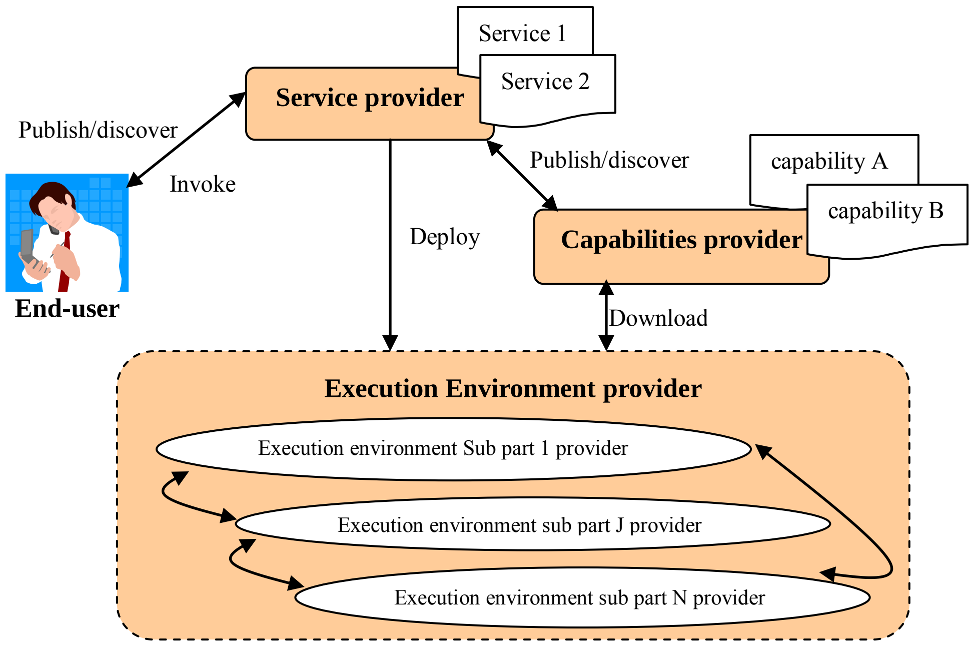

Service 1
Service 2
Service provider
Publish/discover
Invoke
Publish/discover
capability A
capability B
Deploy
Capabilities provider
End-user
Download
Execution Environment provider
Execution environment Sub part 1 provider
Execution environment sub part J provider
Execution environment sub part N provider

Figure 4.3: Overview of the refined business model

The proposed business model assumes that the execution environment is distributed, or that it can be distributed.

4.2.2 Interactions and required functionalities

The EESPPs cooperate to provide the functions of the overall execution environment in a transparent way. Therefore, the others roles' (i.e. EU, CP, SP) interactions with the new entities (i.e. EESPPs) can remain the same. To achieve transparency, one of the EESPPs acts as an entry point to the execution environment. Furthermore, the communication between the EESPPs respects the MANET constraints. We adopt LIME for intra-EESPP communication.

The communication and the collaboration between the EESPPs depend on the execution environment and the distribution scheme of this execution environment. However, certain common features are required. Basically, each EESPP provides a function to publish the description of the sub-part of the execution environment that it 'owns'. Therefore, EESPPs need a discovery mechanism to discover each other. Furthermore, a function to collaborate with the other EESPPs and a function for service execution are required. The interfaces with the other roles are the same as those illustrated in figure 4.3.

An execution environment provider is thus available in the network if and only if all its sub-parts are available, discovered and connected according to an appropriate schema.

4.2.3 Discussion

The refined business model maintains the advantages of the general business model. Furthermore, it enhances the model by enabling the execution environment role to be distributed. A new role, the execution environment sub-part provider (EESPP), substitutes in the execution environment role. Actually, due to resource constraints in MANETs, the new role will allow MANET nodes to provide a small fraction of the functionality of the execution environment. Furthermore, it will make the business model rely not only on organization but on individuals as well. Therefore, the refined model is now meeting all our requirements from the previous chapter, but there is a cost. Distribution means collaboration, which incurs overhead and resource consumption. There is a trade-off to be made between performance and addressing the constraints of MANETs. Our goal is to provide a suitable (e.g. distributed, flexible and lightweight) business model for mobile ad hoc networks while keeping the costs as low as possible. Hence, we believe that using an adequate communication mechanism will limit the impact on performance.

4.3 Mapping to the SIP servlets framework

The business model proposed in the previous section is attractive for MANETs. However, it is described at a high level of abstraction. This section demonstrates how the refined business model is applied in practice. The SIP servlets framework has been chosen as a framework for service provisioning. However, the SIP Servlet Engine (SSE), as an execution environment, must be distributed, since the SSE has to respect MANET constraints. This section first motivates the mapping. Next, it discusses a distribution scheme for the SIP servlets engine. The SIP servlets-based business model is then presented as an outcome of the mapping.

4.3.1 Motivation

Thus far, we have defined the abstract business model and the basic interactions between its roles. A concrete realization of the proposed business model is needed. There are two strategies to approach the problem [65]: an evolutionary strategy and a revolutionary one.

The first approach starts from existing solutions and paradigms, evolves them by refining or reworking them, and then integrates the resulting solution with current approaches. An example of an evolutionary-based solution is Web Services, which reflects new thinking for service provisioning yet rests on existing technologies and standards. The revolutionary strategy is simply an approach that is not an evolution of existing solutions. TINA is a good example of a revolutionary solution. However, TINA gives too little weight to important current technological developments and does not give enough consideration to the installed base systems [103].

The work in this thesis will follow the evolutionary strategy for several reasons. First, the solution will ensure backward compatibility and thus interworking with legacy systems. Second, it will increase its adoption probability since developers and professionals are familiar with existing technologies. Finally, it is more reasonable to take advantage of the current and successful paradigms, especially the mature ones.

The SIP servlets paradigm has proven to be a valuable tool in creating and delivering SIP services in traditional networks with fixed infrastructures. Furthermore, it has spread within a large community that has acquired good expertise in it. In addition, SIP servlets are a mature paradigm based on the SIP, and SIP is the core protocol for next generation networks. Hence, SIP servlets become the primary candidate for service provisioning in the future. Indeed, the IP Multimedia Subsystem (IMS) proposes a SIP servlets-based application server. However, using this paradigm in MANETs for service provisioning requires a signalling layer. A SIP-based architecture for signalling in MANETs has been proposed [40], which makes SIP servlets the best choice. For all of these reasons we chose the SIP servlets paradigm as a basis for implementing the proposed business model.

4.3.2 Distributing the SIP servlets engine

SIP servlets are of prime importance in current and future service provisioning architectures. However, bringing them to MANETs leads to new issues. The SIP servlets framework has a main entity: the SIP Servlets Engine (SSE). This entity may constitute a central node with heavyweight functions and processing. All the drawbacks related to such a configuration (e.g. bottleneck, unrealistic for ad-hoc networks) are thus possible. Even though the proposed business model may deal with central nodes, it is not recommended for MANETs. Several nodes have limited resources (e.g. memory, processing) and they may fail in hosting the entire SSE. Therefore, the SIP servlets framework needs to be extended by distributing the SSE.

We propose a functional distribution scheme for the SIP servlets engine. The SSE is divided into four functional entities that collaborate to achieve the goal of an entire SIP servlets engine. Figure 4.4 presents the SIP servlets framework with a distributed SIP servlets engine.

The components of the distributed SIP servlets engine are:

- *Connector*: The node that provides connectivity to and from the SSE. All SIP messages sent to or received from the SSE must traverse this node. The Connector performs SIP message decoding, parsing and validation. If a message is determined to be valid it is forwarded to the *Controller*. Otherwise, the message is discarded without further action. Likewise, the messages to the SSE's external nodes are parsed, validated and encoded.

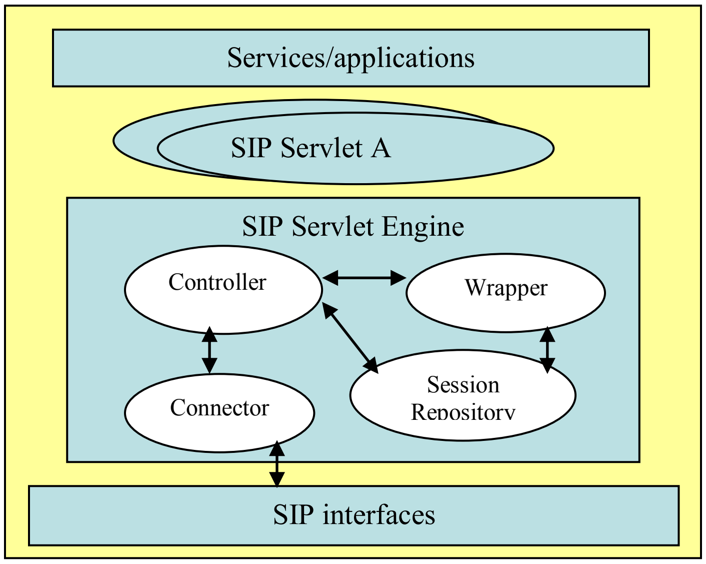


Figure 4.4: The SIP servlets framework with a distributed SIP servlets engine

- *Session Repository*: A repository that stores two types of state information: the overall state of the application represented by SIP application sessions; and the states for individual SIP dialogs (SIP Sessions).
- *Wrapper*: The node that deploys SIP applications. The *Wrapper* extracts the Uniform Resource Locator (URL) of the SIP application's archive. Then, it downloads the application's archive from the servlets repository, loads and instantiates the servlets of the application and manages servlets throughout their lifecycles.
- *Controller*: The node that coordinates all of the other nodes of the distributed SSE. The *Controller* also handles SIP transactions and performs message routing to applications. Furthermore, the *Controller* extracts and stores the rules that specify the conditions that will trigger an application. Finally, it instructs the *wrapper* to download the application's archive.

The distribution scheme described above is the foundation of the subsequent research results.

4.3.3 SIP servlets-based business model for MANETs

From the SIP servlets framework point of view, SIP servlets are the capabilities required to build services. The SIP servlets engine is the execution environment where SIP servlet applications are run. The general business model roles are mapped to the SIP servlets' framework as follows: the end-user is the SIP servlet application's users. The service provider provides the SIP servlet applications and the service logic. The capabilities provider owns the servlets that are offered to the service providers in order to build their applications, and is mapped to the SIP Servlets Provider (SSP). . Finally, the execution environment provider is mapped to the SIP Servlets Engine Provider (SSEP) and it owns the SIP servlets engine provided for the execution of the SIP servlets applications.

However, in the refined business model the execution environment is distributed. Since the SSEP is divided into four components, four roles are derived and mapped to the execution environment sub-part provider. Figure 4.5 illustrates the SIP servlets-based business model for MANETs.

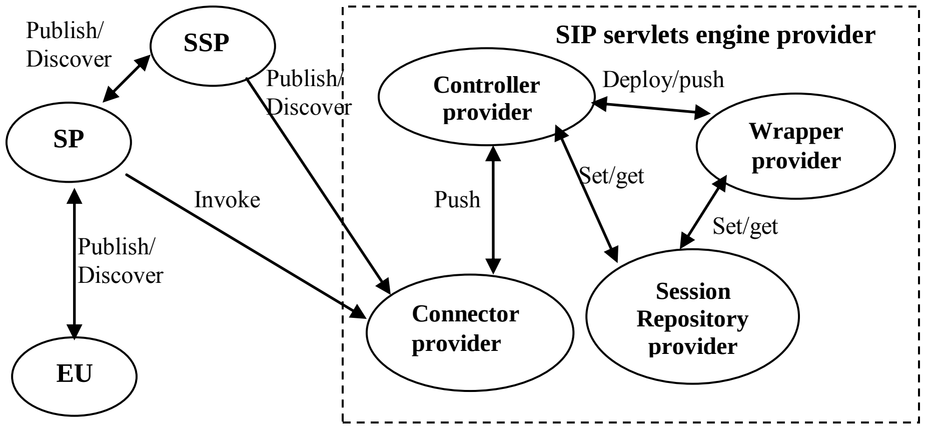


Figure 4.5: The SIP servlets-based business model for MANETs

The *connector provider*, the *session repository provider*, the *wrapper provider* and the *controller provider* are the mapped roles in the SIP servlets context. Each of these roles collaborates to produce the SSEP. In the rest of the thesis we will consider the refined business model.

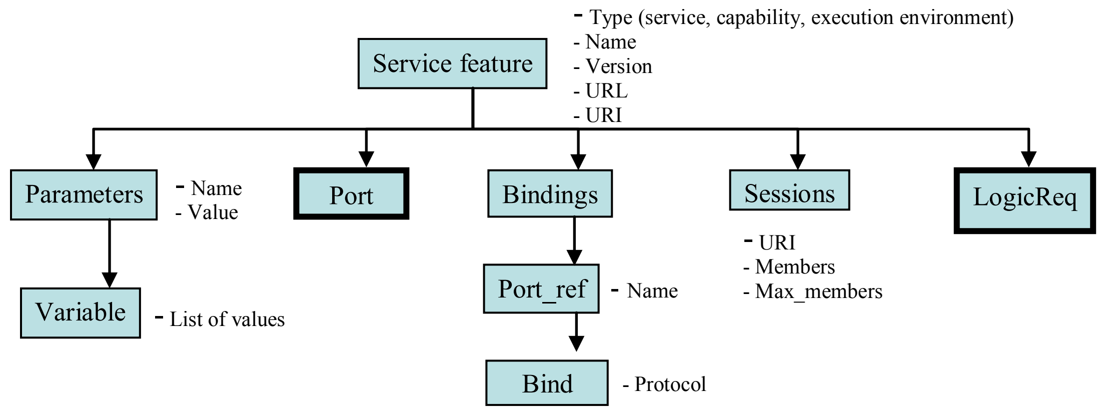
4.4 Publication and discovery

Service description is the starting point for the design of publication/discovery mechanisms. Publication and discovery protocols are discussed afterwards.

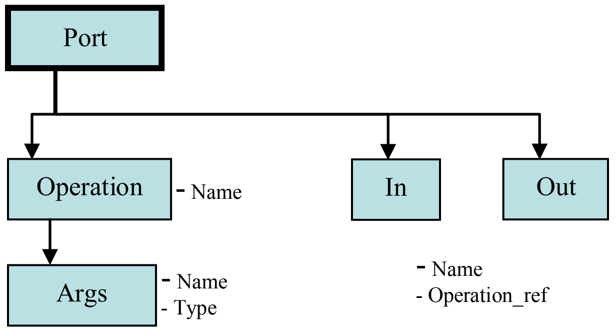
4.4.1 Service description

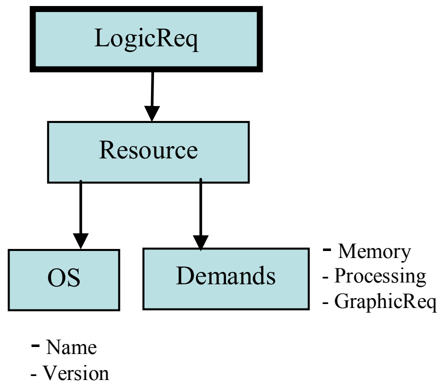
The proposed business model requires a description scheme that allows not only service description but capabilities and execution environment descriptions as well. Moreover, the description language should be machine interpretable to facilitate automation. To meet these objectives we made use of an XML-based scheme. It is a description scheme largely inspired by current approaches, such as WSDL from the web services community. Since this thesis is not focused on description languages design, this solution meets our goals with simplicity, and allows us to consider relevant details related to services, capabilities and the execution environment in ad-hoc networks. Figure 4.6 shows how service features (i.e. end-user service, service capabilities, and execution environment) can be described while taking into account information relevant to MANETs.

Figure 4.6.a presents a service feature as composed of five elements: *parameters, port*, *binding*, *sessions* and *logic requirement*. The *port* and the *logic requirement* elements are expanded in Figures 4.6.b and 4.6.c respectively. The logic requirement's utility is to help to consider an ad-hoc network's characteristics (e.g. limited resource and heterogeneity of devices). The required resources are therefore described in this element (e.g. the operating system and its version, the minimum memory storage, processing and graphical characteristics). The *port* element is similar to the WSDL *operation* element and contains the name, the arguments and the type (input/output) of the function to be invoked to run the service. The *binding* element maps the *port* element to a given port number, IP address and to a supported protocol. *Parameters* describe the service arguments of the service feature. *Parameters* may be of two types: fixed having one value or variable with a set of possible values. The *sessions* element illustrates details about the ongoing sessions.



a. Abstract view of service features description scheme





b. Abstract view of the *Port* element

c. Abstract view of the *LogicReq* element

Figure 4.6: Global view of service features description in MANETs

Furthermore, a service feature is designated by a *type* that defines if it is a service, a capability or an execution environment. Obviously, the feature has a *name*, a *version* and a *URI/URL* so that it can be identified and located.

4.4.2 Publication and discovery protocol

As shown in chapter 3, Linda In a Mobile Environment (LIME) and the Pervasive Discovery Protocol (PDP) are suitable for MANETs. We have experience using both of them. LIME, in particular, was used for communication between the entities of the distributed SIP servlets engine since it is basically designed for distributed and concurrent process communications. Furthermore, it has a motivating characteristic for MANETs: it is not necessary for the sender and the receiver to be connected at the same time and their respective locations are not relevant for exchanging data. Furthermore, LIME introduced the notion of Reactions. A reaction can be registered or deregistered, and fires when a *tuple* matching a given pattern is found in the *tuple space*. Three basic primitives are defined: *out(t)* to add a *tuple t* to the *tuple space*, *in(p)* to read and remove a *tuple* that matches the pattern *p* and *rd(p)* to read but not remove the *tuple* matching the pattern *p* [104].

PDP is used for publication and discovery. The PDP protocol is simple and has two mandatory messages: *PDP_Service_Request* and *PDP_Service_Reply,* to request services and to reply and announce services, respectively. A third, optional message *PDP_Service_Deregister* is introduced to announce that a service is no longer available.

To discover available services in the network, a device makes use of the *PDP User Agent,* and to publish services the *PDP Service Agent* is used. However, some small extensions (i.e. new fields in some message headers) have been performed in order to enable the *PDP_Service_Reply* message to be used for the push mode.

4.5 Illustrative scenarios

We next present some scenarios to demonstrate how the proposed business model can be applied for service provisioning in MANETs.

4.5.1 Distributed SIP servlets engine interactions

The interactions between the components of the distributed SIP Servlets Engine (SSE) are depicted first. Sequence diagrams are presented for an abstract communication flow and for a LIME-based communication flow.

Figure 4.7 presents the abstract view of the distributed SIP servlets engine handling an initial request.

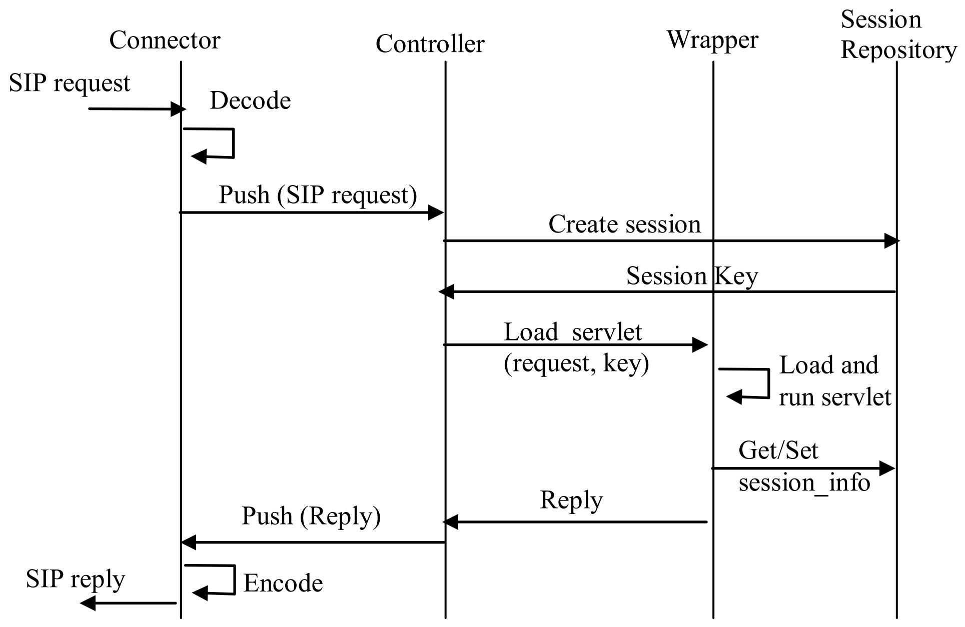


Figure 4.7: Abstract view of the distributed SSE handling an initial SIP request

When received by the *wrapper*, the SIP request is decoded and then forwarded to the *controller*. Because it is an initial message, the *controller* creates an entry in the *session repository* and gets the session key. This key is transmitted to the *wrapper* together with the SIP request. The *wrapper* then downloads and runs the appropriate servlet. During the service execution, the *wrapper* may retrieve or modify the session information using the session key. The reply is generated by the servlet and transmitted to the *connector* through the *controller*. The connector encodes the message and sends it to its destination. Figure 4.8 shows the LIME implementation of the previous scenario.

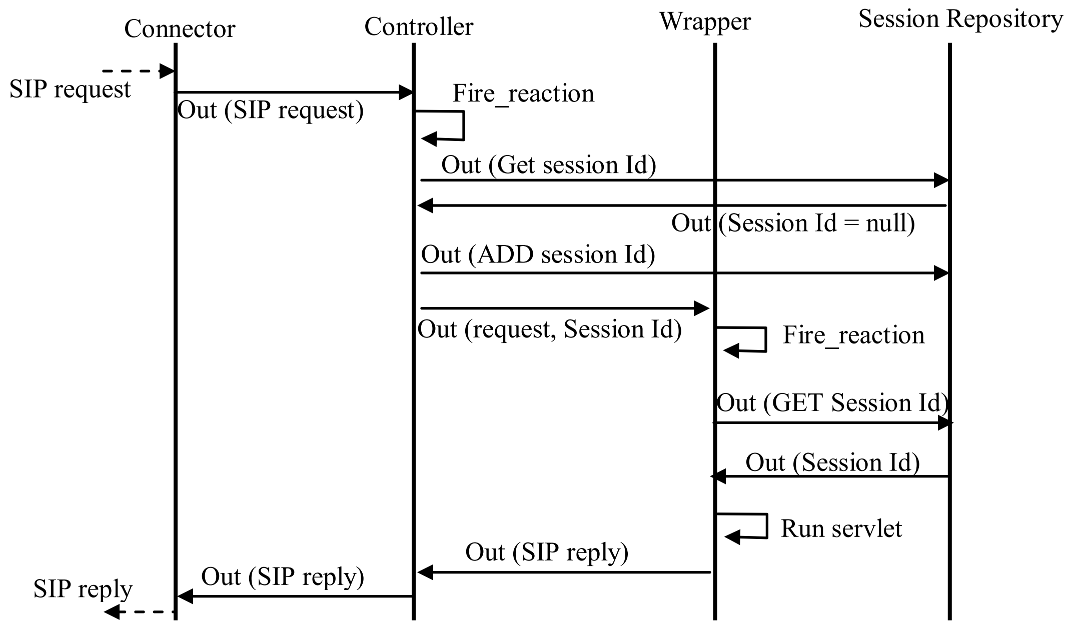


Figure 4.8: Distributed SSE handling an initial SIP message using LIME

The communication between SSE components follows the same scenario as shown in figure 4.7. The LIME primitive *out()* and the concept of *reaction* are employed. For clarity we will model the SIP servlets engine as one box in the remaining scenarios in this section. However, it may be either a centralized SSE or a distributed one. In the latter case, the above mentioned scenarios are applied.

4.5.2 LIME-based scenario

Figure 4.9 presents the sequence diagram of the *pull* mode in a LIME-based publication and discovery scenario. It is a three-phase process. The LIME setup phase prepares the environment. The publication/discovery phase is where services/features are discovered and the invocation phase is where services are used.

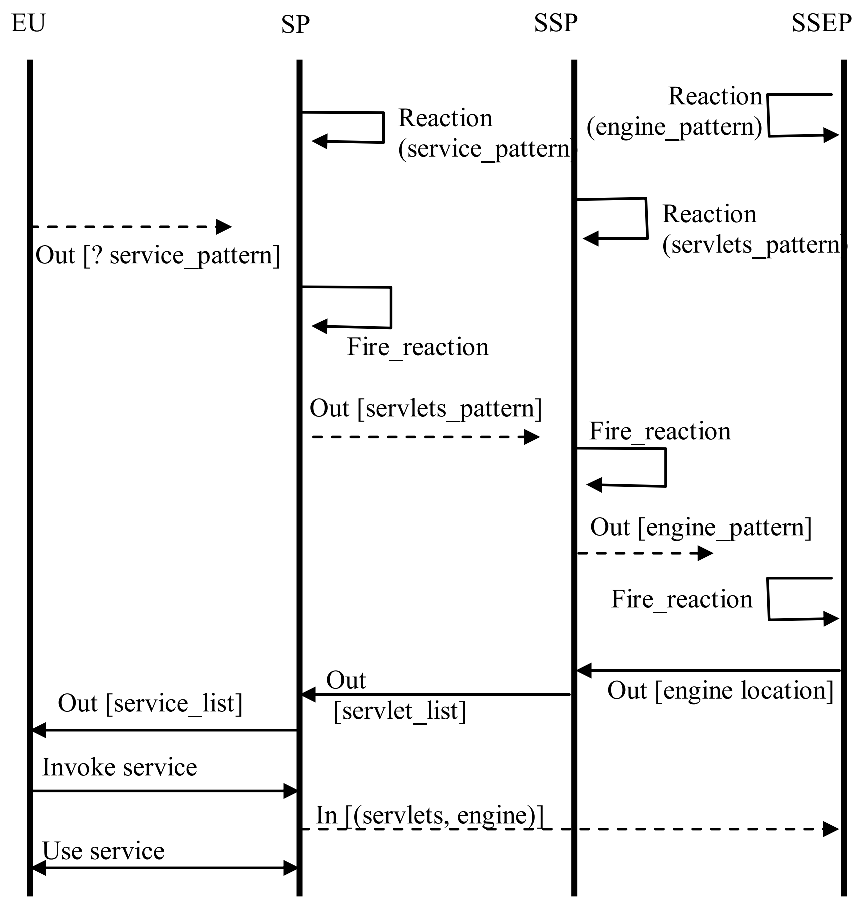


Figure 4.9: LIME-based scenario for publication and discovery: *pull* mode

In the first phase, the SIP Servlet Engine Provider (SSEP), the SIP Servlets Provider (SSP) and the Service Provider (SP) register LIME reactions. A reaction is a code to be executed when a *tuple* matching a given pattern is found in the *tuple space*. This phase allows the different providers to react to the service discovery messages, and takes the place of service publication.

The second phase starts when the end-user requests a service. The SP that owns the matched service fires its reaction. A mapping to the required servlets is performed. Then the service provider discovers the needed servlets using the *out* primitive. As a result, the SSP's reaction fires and the SIP servlets engine discovery is initiated. The *out* primitive is used for discovery, which fires the reaction of the SSEP so that it sends the location of the SIP servlets engine to the SSP. Then the SSP returns the address of the requested servlets to the SP. Finally, the service provider replies to the end-user with the requested service description.

The third phase begins with service invocation. The service provider then contacts the SSEP to run the service.

4.5.3 PDP-based scenario

The same scenario described above is illustrated in figure 4.10 using an extended PDP protocol for publication and discovery.

The PDP request specifies the type of the service feature to be discovered and its name. Therefore, the end-user discovers services, the service provider discovers servlets and the SIP servlets provider discovers the SIP servlets engine. When the different features are discovered, the service provider can send the list of services to the end-user. Each feature has a Time-To-Live (TTL) to guarantee up-to-date information.

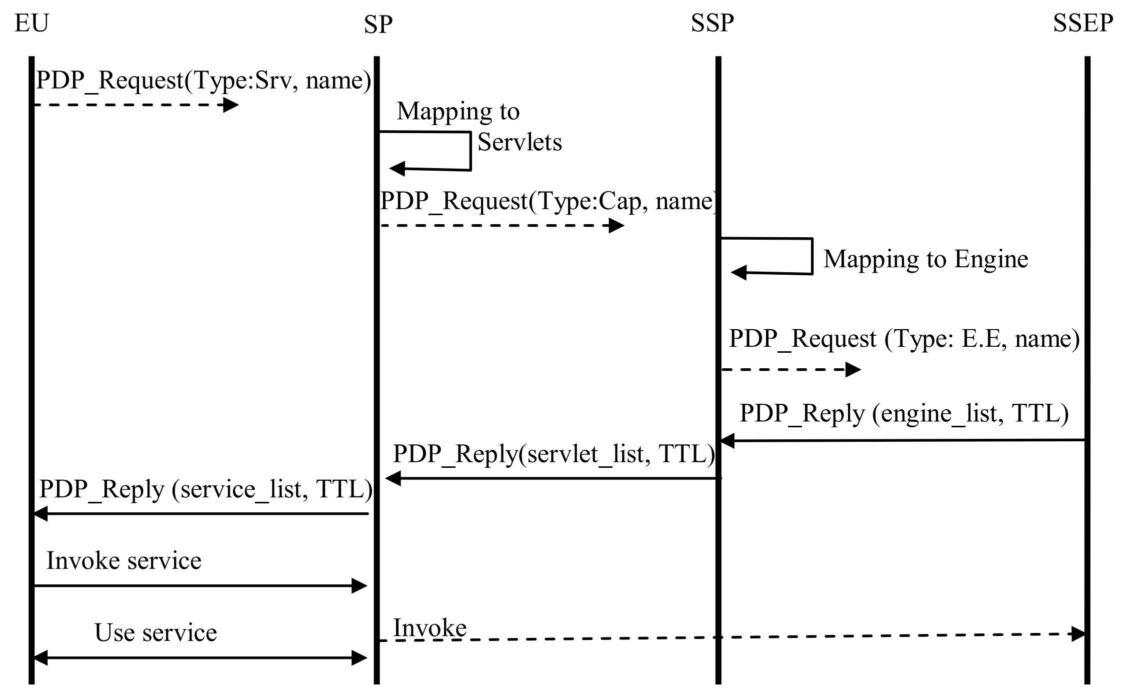

EU
SP
SSP
SSEP
PDP_Request(Type:Srv, name)
Mapping to Servlets
PDP_Request(Type:Cap, name)
Mapping to Engine
PDP_Request (Type: E.E, name)
PDP_Reply (engine_list, TTL)
PDP_Reply(servlet_list, TTL)
PDP_Reply (service_list, TTL)
Invoke service
Use service
Invoke

Figure 4.10: PDP-based scenario for publication and discovery: *pull* mode

Figure 4.11 shows the push variant of the previous scenario.

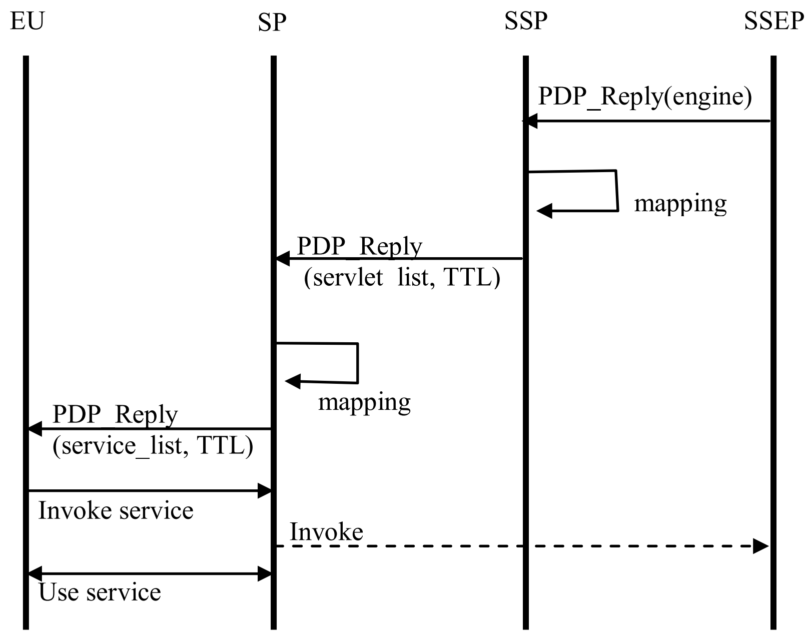


Figure 4.11: PDP-based scenario for publication and discovery: *push* mode

In this scenario, the SSEP pushes the type, name, and pointer to detailed description of its engine to the network. The push is either based on periodicity or events. Upon reception, SSPs select from the owned servlets those that can be mapped to the described engine. They then update their list of available servlets and decide to either perform no action or to push these servlets' descriptions to the network. Later, if they decide to react, the service providers push a list of services that require the received servlets to the end-users. Service invocation remains the same in all cases.

4.6 Summary

In this chapter we proposed a novel business model for service provisioning in stand-alone MANETs. Both a refined and a general business model were elaborated. We described the roles and their interactions. We also demonstrated that the refined business model meets all our requirements for MANETs. The business model was then mapped to a concrete service provisioning framework. Based on our criteria, the SIP servlets paradigm was chosen, and an extension to the framework was presented: a distribution scheme for the SIP servlet engine.

In addition, publication and discovery were discussed in some detail. First, a description language inspired by existing approaches was proposed, which takes into consideration the specific requirements of MANETs. Then, based on the previous chapter's results we selected LIME and PDP as mechanisms for publication and discovery.

The chapter ends by presenting diagrams that demonstrates how all these elements can be put together. Scenarios are presented to illustrate the communication process between the distributed SIP servlets engine components and the service publication/discovery using LIME and PDP.

CHAPTER 5: An overlay network for a SIP servlets-based service execution environment in MANETs

This chapter proposes an architecture for service execution in stand-alone MANETs. A brief introduction is followed by an overview of overlay networks. Next, the proposed overlay network architecture is depicted, with a subsequent discussion of the underlying procedures of the architecture. Then, an overlay network protocol is proposed and detailed. The chapter ends with scenarios that illustrate the overlay network architecture.

5.1 Introduction

The architecture we propose in this chapter is based on the extended SIP servlets framework. The distribution scheme of the SIP servlets engine described in the previous chapter is the starting point. The service execution environment that uses the extended SIP servlets framework is an important component for service provisioning in MANETs. However, in order to become a realistic solution for MANETs, the architecture will be extended with a method to manage the topology changes. Furthermore, to realize service execution in MANETs, the proposed architecture enables several SIP servlet engines (SSEs) and different instances of the same SSE component to coexist. Therefore, many *controllers*, *connectors*, *wrappers* and *session repositories* may be part of a MANET. These multiple instances or nodes form a network of SSEs. The communication between these nodes should be handled to fulfill the SSE goal. Node coordination, self-organization and recovery are new issues to solve. Since the underlying network is a MANET, which adds complexity, an adequate architecture is needed to manage the distributed SIP servlet engines. Peer-to-peer overlay networks appear to be a promising solution. Indeed, they are robust, reliable, and enable self-organization and recovery.

5.2 Overview of overlay networks

Peer-to-peer overlay networks [105] are logical structures on top of the physical network. The logical nodes are mapped to one or more physical nodes. The overlay network comes with its own protocols to build the desired logical structure. The main advantages of the overlay networks are: robustness, because the overlay network changes according to events (e.g. node failure, increasing load), and reliability, because logical links adapt to the physical network changes and scalability. Furthermore, overlay networks require no change to the underlying existing technology.

The peer-to-peer overlay networks come in two varieties: structured and unstructured. In structured overlay networks the data object is placed at well-known locations. The lookup time, in such networks, may be high and may affect the network performance. In unstructured overlay networks nodes are randomly organized in a flat or a hierarchical style. They introduce less overhead than structured networks and they are ad hoc by nature.

Consequently, unstructured overlay networks are an elegant way to organize the SIP servlet engines with no changes to the underlying physical network. We chose this network type to implement the SIP servlet engines for service execution in MANETs.

5.3 The overlay network architecture

We propose an overlay network architecture to manage the distributed SIP servlets engine and thereby realize service execution in stand-alone MANETs. The proposed architecture is based on some assumptions and principles. This section presents the architectural assumptions and principles, and then discusses the overlay network architecture's design.

5.3.1 Assumptions and architectural principles

a. Assumptions: We assume that a SIP servlet engine is available when a *controller* is connected to at least one *connector*, one *wrapper* and one *session repository*. In addition, a *controller* manages zero, one or multiple *connectors*, *wrappers* and/or *session repositories*. *Wrappers* and *session repositories* connect to one or more *controllers,* but a *connector* serves one and only one *controller*.

b. Architectural principles: The starting point of the overlay network is the four components of the distributed SIP servlets engine. The *controller*, the *wrapper*, the *session repository* and the *connector* require close collaboration to provide the SIP servlets engine's functionalities. Indeed, these components offer the "service execution environment" as a service to the rest of the network. Therefore, they are the fundamental nodes of the overlay network.

Furthermore, to each node in the MANET we assign a *type* where *type* ∈ {*Connector*, *Wrapper*, *Controller*, *Session Repository*, *Null*}. *Null* type is used by nodes that are not participating in the SIP servlets engine. In other words, the type defines if a node belongs to the overlay network or not. Basically, each node that hosts an SSE component is an overlay node. Thus, by overlay nodes we refer to nodes of **type*** where **type*** ∈ {*Connector*, *Wrapper*, *Controller*, *Session Repository*}. Moreover, we define nodes of **type***$^+$ as overlay nodes, excluding the controller: **type***$^+$ ∈ {*Connector*, *Wrapper*, *Session Repository*}.

In addition, each *controller* has a well-known capacity. The capacity refers to the number of nodes a controller is able to mange while incurring limited impacts on performance. The *controllers*' capacity is pre-configured and is a property of the *controller* node. Nodes discover each others' type when they join the network. This discovery is a part of the overlay network protocol.

5.3.2 The overlay network design

We describe the structure of the overlay network and then present the overall topology.

5.3.2.1 A two-level overlay network

Regarding the nature of the nodes that compose the SSE, we separate the overlay network into two levels. The first contains *repository nodes,* whose role is limited to data storage and management, and the second level includes *execution nodes* which perform the necessary processing for service execution. *Repository nodes* are the session repositories while *execution nodes* are controllers, wrappers or connectors. Figure 5.1 presents an abstract view of the two levels of the overlay network.

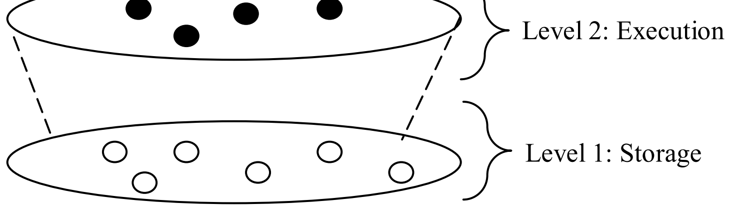


Figure 5.1: An abstract view of the overlay network's levels.

The nodes in level 1 and in level 2 are both overlay nodes. They are distinct entities which are directly mapped to the real ad hoc network. Furthermore, nodes that belong to level 1 of the overlay network should obviously have storage capabilities, be able to comprehend messages from the overlay nodes, and have publication/discovery capabilities.

The second level is made up of three types of overlay nodes: *wrappers*, *connectors* and *controllers*. The common functionalities for these nodes are to understand overlay messages and to publish and discover overlay nodes' *types*.

In addition, *wrappers* should be able to communicate with the SIP servlets provider, and to load and run servlets. The *connectors* should be able to understand, manage and process commands from end-users, and encode and decode SIP messages. *Controllers* should be able to understand and process commands from service providers, route SIP messages to the *wrapper* and handle SIP protocol transactions.

5.3.2.2 The overlay network topology

Repository nodes are fully meshed so as to exchange the data related to the ongoing applications and sessions. The motivation behind full-mesh topology is to simplify failure recovery by enabling data replication. In level 2, the *controllers* are fully meshed to facilitate the exchange of information about the nodes they manage, and to speed the recovery mechanism when a *controller* leaves or crashes. Furthermore, each *controller* is a root of a tree whose leaf nodes are *connectors*, *wrappers* and *session repositories*. The depth of the tree is 1.

Figure 5.2 illustrates the level 2 topology.

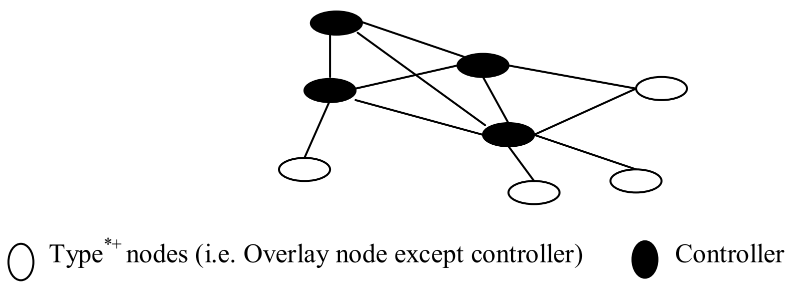


Figure 5.2: Topology of level 2 of the overlay network

Figure 5.3 presents the overall picture of the proposed overlay network. The two levels correspond to the distributed SIP servlets execution environment, which is presented as a value-added service provided to the real MANET's entities. Each node of the overlay network can be mapped to an appropriate SIP servlets sub-part provider.

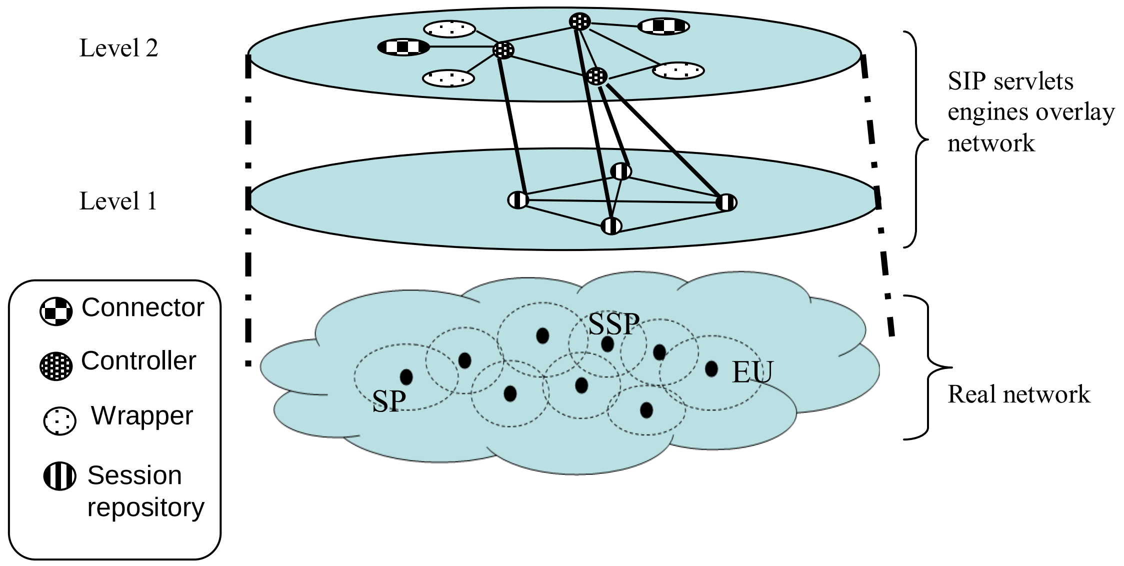


Figure 5.3: Overall view of the SIP servlets overlay network

Thus far we have described the overlay network from the conceptual point of view. However, since we are considering infrastructure-less environments (i.e. mobile ad-hoc networks) we need suitable procedures for the overlay network management.

5.4 The overlay network procedures

The dynamic nature and the unreliable connection links of MANETs require appropriate processes. To address these specific needs of ad-hoc networks, we propose procedures for self-organization and for self-recovery.

5.4.1 Self-organization

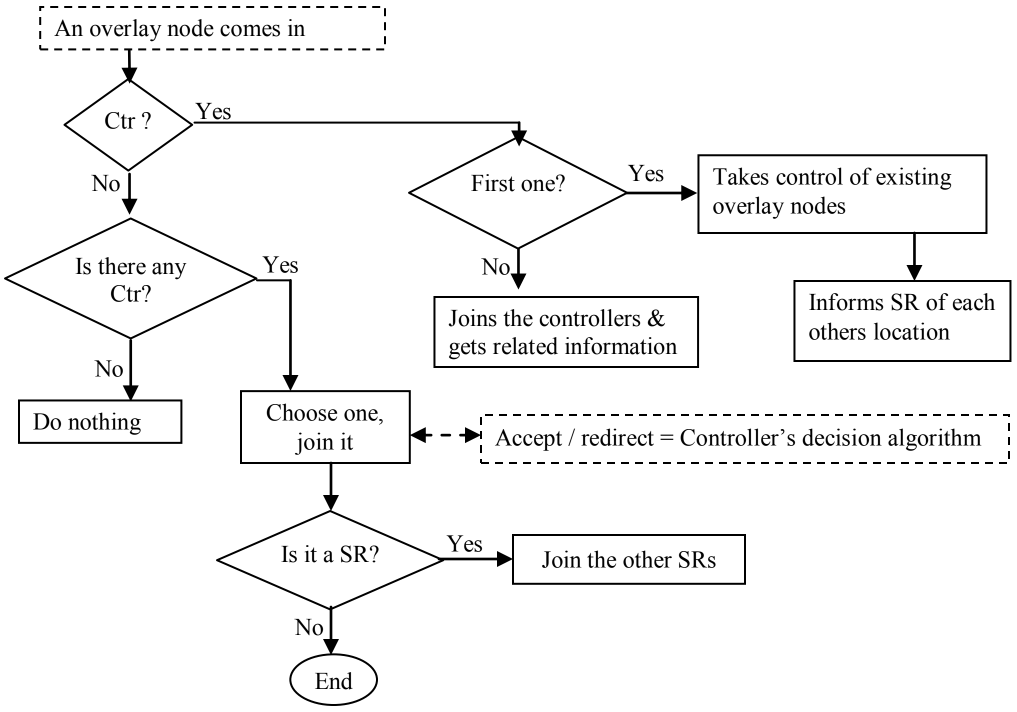
By self-organization we mean the ability of nodes to be structured in the overlay network architecture defined in the previous section, and their ability to maintain this structure automatically. First, we discuss the self-organization procedure, and then illustrate it through scenarios.

When a node comes into a MANET it publishes its type and discovers the other nodes' types. The process of self-organization depends on the node joining the network. The goal is to connect nodes of a certain type^{*+} (i.e. *connectors*, *wrappers*, *session repositories*) to a given *controller*. This is motivated by the fact that a SIP servlets engine is defined when a *controller* is connected to a *connector*, a *wrapper* and a *session repository*. Furthermore, self-organization should ensure that *session repositories* are fully meshed, as well as the *controllers* are. The procedure is as follows:

- If the joining node is a *controller*: if it is the first one (i.e. no other *controller* is in the network): it informs the overlay nodes (i.e. nodes of type^{*+}), if any, to join its logical control area. The joining node is then the *controller* of each node in its logical control area. Next, the *controller* notifies the *session repositories* of each others' location in order to get a full mesh connection between them. However, if it is not the first *controller* (there is at least one controller in the network), it establishes a full mesh connection with the existing *controller(s),* and then gets the list of the managed overlay nodes from each of them.

- If the joining node is of type^{*+}: If there is no *controller* in the network, it does nothing. If there is at least one *controller,* the joining overlay node randomly chooses one *controller* and joins it. The chosen *controller* will decide either to accept the joiner or to redirect it to another *controller,* based on the information it has about the *controllers*. The decision algorithm should consider the *controllers*' capacity and the need to balance the nodes among the *controllers*. For example, to try to ensure that all *controllers* have at least one *connector*, one *session repository* and one *connector*. Furthermore, if the joining node is a *session repository* then the *controller* sends it the list of the existing *session repositories* so that a full mesh connection can be established between them.

Figure 5.4 presents the overall self-organization process for the proposed overlay network, in which SR is the Session Repository and the *Ctr* is the controller.



An overlay node comes in
Ctr ?
Yes
No
First one?
Yes
Takes control of existing overlay nodes
Is there any Ctr?
Yes
No
Joins the controllers & gets related information
Informs SR of each others location
No
Do nothing
Choose one, join it
Accept / redirect = Controller's decision algorithm
Is it a SR?
Yes
Join the other SRs
No
End

Figure 5.4: The overall self-organization process

A controller therefore should implement a decision algorithm. The algorithm checks the nodes' balancing and guarantees that the capacity of the selected *controller* is not exceeded. The algorithm's input is an overlay node of type^{*+} and the output is a *controller* to which the overlay node will be connected. Figure 5.5 shows the controller decision algorithm.

```
Let :
L_Ctr: set of controllers
N_Ctr: the number of controllers in the network
ONet: an overlay node ∈ type*+
Ltype_Ctr(i): the list of nodes' types of the ith controller
Ctr(i): the ith Controller – Ctr(0) = this controller
C_Ctr(i): the capacity of the ith controller
N_m_Ctr(i): the number of managed nodes by the ith controller
S_Ctr: the selected controller

Input = ONet   ;    Output = S_Ctr

Start
    S_Ctr = Null
    T = new Table(N_Ctr,2)
    For i = 0 → N_Ctr
        occ= occurrence(type(ONet), Ltype_Ctr(i))
        if C_Ctr(i) > N_m_Ctr(i) + 1 then
                T[i,0]=occ
                T[i,1]=Ctr(i)

    End For
    Sort_occ (T)
   If T[0,0] = occurrence(type(ONet), Ltype_Ctr(0))
    AND C_Ctr(0) > N_m_Ctr(0) + 1
                 Then
                        S_Ctr=Ctr(0)
                 else
                        S_Ctr = T[0,1]

End
```

Figure 5.5: The *controller*'s decision algorithm

All the information required to run the algorithm is available locally, so no extra message exchange is required. The algorithm starts by identifying, for each *controller,* the number of nodes it has in its control area. The algorithm counts only the nodes that are the same type as the input node. The occurrences and the corresponding *controller*'s addresses are stored in a two-dimensional table. Next, the table is sorted according to the occurrences, from lowest to highest. Only controllers with adequate capacity are kept, which permits balancing of the nodes among the *controllers*. Indeed, controllers with a small set of nodes of the same type as the input node have a greater chance to be connected to the input node. Finally, the selected *controller* is the one in the first line of the table. However, to avoid unnecessary message exchanges over the network, the algorithm makes sure that the current *controller* (i.e. the controller running the decision *algorithm*) does not have the same occurrences as the selected *controller*. In such a case the selected *controller* is the current *controller*.

5.4.2 Self-recovery

This section proposes a procedure to deal with network failures. Basically, this procedure allows the overlay network to re-organize automatically upon a failure. Failure can occur when an overlay node becomes unreachable or unavailable. Some sources of failure are: nodes deliberately leave, nodes crash, nodes go out of the network's range, and a node's battery goes down.

Here we need to distinguish between two major cases: expected failures (i.e. nodes deliberately leaving the network by announcing their departure) and unexpected failures (e.g. a node's sudden crash).

Self-recovery depends on the node that fails and the nodes present in the network when the failure happens. Let's re-state that a *wrapper* or a *session repository* may be connected to more than one *controller*. However, a default *controller* is identified for each node. Figure 5.6 illustrates the overall self-recovery process, where *Ctr* refers to the *controller* and a busy node means that the node is involved in a service session.

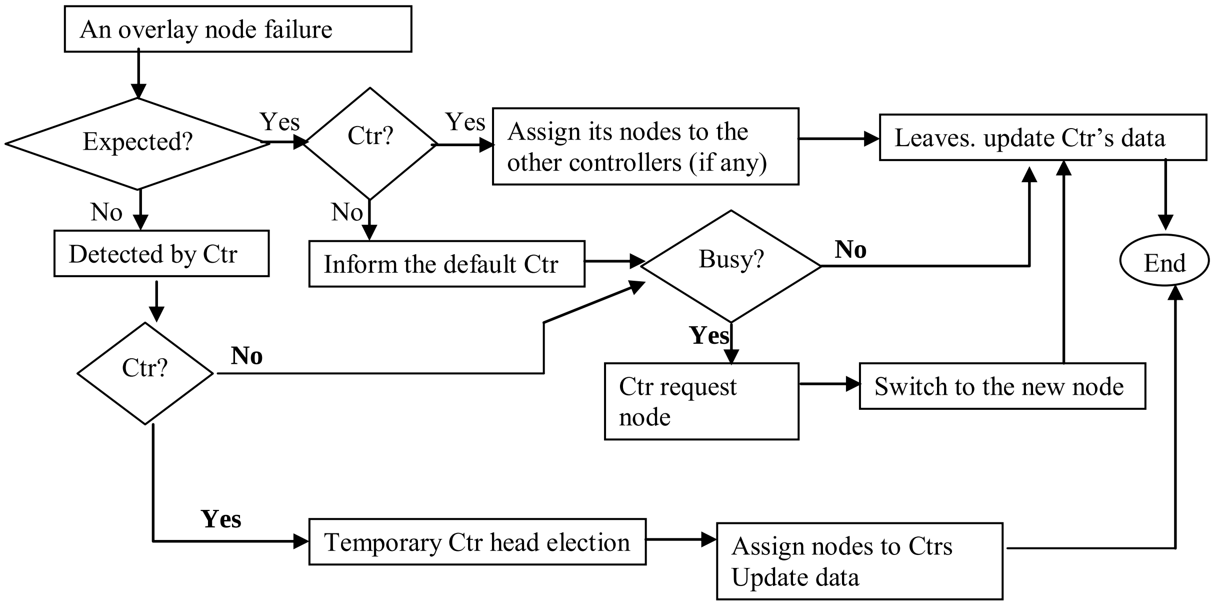


Figure 5.6: The overall self-recovery process

5.4.2.1 Expected failures

As far as expected failures are considered, the process of recovery is as follows: If the leaving node is a *wrapper* or a *session repository*, it informs its default *controller*, which then notifies the other *controllers* to update their entries. If the leaving node is involved in a service session, the *controller* will ask the other *controllers* for a node of the same type as the leaving node. Furthermore, if the leaving node is a *connector*, the end-user is notified with an alternative access point.

When the leaving node is a *controller*, it informs the existing controllers in the network and assigns them the nodes it manages.

5.4.2.2 Unexpected failures

With unexpected failures, a procedure for failure detection is needed. Any heartbeat protocol can be used to achieve this goal. A heartbeat protocol has been proposed for failure detection in MANETs [106].

With the failure detection protocol, unexpected failures, in general, are handled in the same way as expected failures, with slight but pertinent adjustments. For *wrappers* and *session repositories* failures, the only difference is that the default controller of the failed node is responsible for the failure detection. After that, the process remains the same as for expected failures.

When the *connector* goes down suddenly, the failure is detected both by the default *controller* and by the end-user. From the default *controller's* perspective, it notifies the other *controllers* about the failure and requests a *connector* if the crashed node was involved in a service.

It is more complicated from the end-user's point of view, since this *connector* was the access point to the execution environment. To solve this problem, when an alternative *connector* is found by the default *controller*, the service provider is informed. The service provider then sends the new access point address to the end-user.

The most complex case is when the *controller* crashes. In this situation, the *controllers* should elect a temporary head to handle this situation. The head organizes the network and assigns the unattached overlay nodes to the remaining *controllers*.

In order to limit message exchanges and therefore reduce the network overhead, the *controller* head election is based on a simple algorithm. Indeed, the head is the *controller* that has the highest IP address. Since all the *controllers* know each others' address the election is done automatically following the failure detection.

In the self-recovery process the nodes managed by the failed *controller* should be assigned to other controllers. The *controller decision algorithm* discussed previously is then used. However, some of these nodes may be connected to more than one *controller*. Therefore, to avoid overloading the network, these kinds of nodes are not assigned since they will still be connected to at least one *controller*.

5.5 The overlay network protocol

In order to make the recovery problem easy to solve, the *session repositories* should exchange their information about ongoing sessions and applications. Furthermore, the *controller*s should have a global view of the overlay network. They especially need to know the types of the nodes controlled by each controller and their status (are they involved in a session or not). The status is very important in the case of *connectors* because a *connector* can only be connected to one *controller* at a time.

The overlay network should have redundancy at the first level (i.e. the *session repositories* level) and a collaboration of *controllers* at the second level. The self-organization and self-recovery processes require a protocol in order to be realized. In this section we first present the data format and protocol messages, followed by the state diagrams.

5.5.1 Data format and protocol messages

Messages are required for two different purposes. Messages are necessary for the data exchange between *session repositories* and between *controllers* as explained above. Also, a consistent set of messages is required to perform self-organization and for self-recovery operations. The self-organization and self-recovery processes make use of both sets of messages.

5.5.1.1 Data format

Each *session repository* maintains a table where each row refers to the information managed by another session repository. The table of the *session repository **j*** (*SR_j*) is illustrated in Table 5.1 where *n* is the number of *session repositories* in the network at a given time.

Node ID	Session info	Application info
SR_1	$session_{1,1}$(…), $session_{1,2}$(…),…	$Appli_{1,1}$(…), $appli_{1,2}$(…),…
SR_2	$Session_{2,1}$(…), $session_{2,2}$(…),…	$Appli_{2,1}$(…), $appli_{2,2}$(…),…
…	…	…
SR_j	$Session_{j,1}$(…), $session_{j,2}$(…),…	$Appli_{j,1}$(…), $appli_{j,2}$(…),…
…	…	…
SR_n	$Session_{n,1}$(…), $session_{n,2}$(…),…	$Appli_{n,1}$(…), $appli_{n,2}$(…),…

Table 5.1: Session repositories data table

This table should be updated when the other session repositories send new information. For example, the *SR_j* multicasts the line *j* of its table to session repositories in the network whenever entries are added, deleted or modified. Other techniques can be used to reduce the load, such as using clusters or only updating neighbours.

Regarding the *controllers*, each one should maintain the list of the *controllers* and their capacity, and of the nodes in their logical control area. Each *controller* should inform any joining *controller* about the nodes it controls. Furthermore, *controllers* need to exchange their related information for a data update. This is done by sending the line that corresponds to their managed nodes. For example, Ctr_i should send the line *i* when required. The data table of the controller ***i*** (Ctr_i) is shown in Table 5.2 where *p* is the number of available *controllers* in the network at a given time. The data is stored in the form: *node_type(IP,status)*.

Node ID	**Capacity**	**Controlled nodes**
Ctr_1	α	connector$_{1,1}$(@,free), connector$_{1,2}$(@, busy)…
…	…	…
Ctr_i	β	connector$_{i,1}$(@,free), SR$_{i,1}$(@,free)…
…	…	…
Ctr_n	δ	SR$_{n,1}$(@,free), wrapper$_{n,1}$(@,busy)

Table 5.2: Controllers' data table

5.5.1.2 Protocol messages

We propose a set of messages that can either be a part of a new protocol or become an extension for existing protocols.

For the data exchange *between session repositories* and *controllers,* the proposed messages are: *add_entry()*, *remove_entry()*, *update_entry()* and *get_entry()*.

- *Add_entry()*: is used to add an entry for a session repository or a controller that has recently joined the overlay network.
- *Remove_entry()*: to delete the entry of a leaving or unavailable session repository or controller.
- *Update_entry()*: to update the information related to a given session repository or a controller.
- *Get_entry()*: is used by a session repository or a controller that has just joined, in order to get information from the other session repositories.

Regarding the self-organization and self-recovery operations, the proposed messages are: *info(x)*, *Join(src, dest, type)*, *Refers(y)*, *Add(x,type)*, *Bye()*, *Request_node(type)*, *Node_reply(x)*, *Disconnect()*, *Ok()*.

- *Info(x)* is sent by the controller to the connector, the wrapper or the session repository. It is an invitation to join node *x,* which is necessarily a controller. This message is also sent by a session repository to another session repository and has the same meaning. It allows session repositories to establish full mesh connections.
- *Join(src, dest, type)* is sent by any overlay node to the controller. It can also be sent by a session repository to another session repository. It means that the source *src* having the type *type* wants to join (i.e. establish a link) with the destination *dest*. This message is usually sent following the reception of the *info(x)* message.
- *Refers(y)* is sent by a controller to nodes of type^{*+} (i.e. non-controller overlay nodes). The destination is informed that it is redirected. The destination is invited to join node *y* (necessarily a controller). The message is sent as a result of the controller's decision algorithm execution. A *Refers(y)* message is also sent by a leaving connector to the

end-user, which is thereby informed about an alternative connector (i.e. *y*) to use to access the SSE.

- *Add(x,type)* is sent by a controller to another controller. The sender requests that the destination controller adds the node *x* of type *type* to its managed nodes list (i.e. its logical control area).
- *Bye()* is sent by any overlay node to its default controller to announce its departure.
- *Request_node(type)* is sent by a controller to another controller to request a node of type *type*.
- *Node_reply(x)* is the reply to the previous message with the node *x* matching the requested type.
- *Disconnect()* is a message sent by a controller to a node of type^{*+} or by a connector to the controller to remove the sender's related information from the receiver's list. It is an update message for overlay nodes. In fact, each node of type^{*+} keeps a list of controllers it is connected to and identifies the default controller. This list needs to be updated in some cases (e.g. the default controller leaves). Furthermore, since the connector is connected to one and only one controller, this message is required when the connector has to change its controller following a re-organization process.
- *Ok()* is used to acknowledge *Bye*, *Add*, *Join* and *Refers*.

The proposed messages for self-organization and self-recovery operations are summarized in Table 5.3, in which *CTR* refers to the controller, *CONN* refers to the connector, *SR* refers to the session repository and *WR* refers to the wrapper. *CTR* → {*CONN*, *WR*} means that the message is sent by a controller to either a connector or a wrapper. {*CONN*, *WR*}→ *CTR* means that the message is sent either by a connector or by a wrapper to a controller.

Messages	**From → to**	**meaning**
Info(x)	*CTR* → {*CONN*, *WR*, *SR*}; *SR*→*SR*	Its an invitation to join the node x
Join(src,dest, type)	{*CONN*,*WR*,*SR*,*CTR*}→*CTR* ; *SR*→*SR*	*src* wants to join *dest*; *type* is the *src* type.
Refers(y)	*CTR* → {*CONN*, *WR*, *SR* }; *CONN*→*EU*	1- Destination is informed that he is redirected to join y. 2- Informs the end-user about the new connector x
Add(x,type)	*CTR*→*CTR*	Add node x to the receiver's list of controlled nodes.
Bye()	{*CONN*, *WR*, *SR*, *CTR*}→ *CTR*	I am leaving
Request_node(*type*)	*CTR* → *CTR*	Request a node of type *type*
Node_reply(x)	*CTR*→ *CTR*	Reply with the address of the requested node.
Disconnect()	*CONN*→*CTR* ; *CTR*→{*CONN*, *WR*, *SR*}	Remove the sender from the receiver's list.

Table 5.3: Proposed messages for overlay network organization and recovery operations

As mentioned earlier, these messages can either be part of a new protocol or extend an existing one. Next, we propose a possible mapping between the proposed messages and SIP messages. Headers should be extended to reflect message meanings.

Info() can be implemented using the SIP REFER message. *Join()* can be implemented using SIP INVITE. This latter may also implement the *Refers()* overlay network message. *Add()* may be implemented as a SIP REGISTER message. *Request_node()* and *Node_reply()* can be implemented by the SIP INFO and SIP OK messages, respectively. *Disconnect()* and *Bye()* may best be mapped to SIP BYE, and *Ok()* to SIP OK.

Table 5.4 presents the messages for data exchange.

messages	**meaning**
Add_entry()	Add an entry in a session repository or a controller table.
Remove_entry()	Delete the entry of a leaving or unavailable session repository or controller
Update_entry()	Update the information related to a given session repository or a controller
Get_entry()	Retrieve information from a remote session repositories or controller

Table 5.4: Proposed messages for data exchange

5.5.2 State diagrams

In this section we present the state diagrams that illustrate the behaviour of the proposed overlay network. Each entity in the overlay network behaves differently according to the protocol. However, at an abstract level the different entities composing the global system go through the same abstract states. Figure 5.7 presents the abstract state diagram of the global system behaviour.

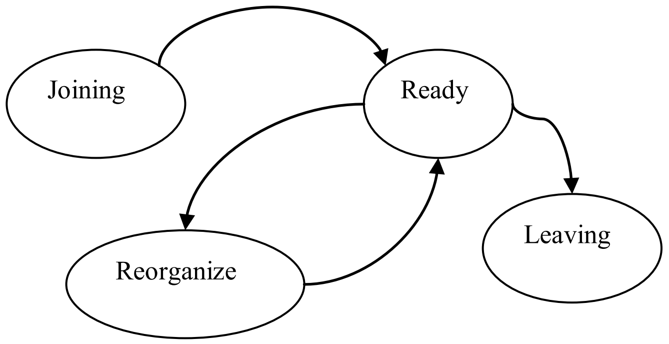


Figure 5.7: The abstract state diagram of the global system

Initially, the overlay nodes explore the networks to discover the existing nodes and publish their *type*. Then a process is started to either join an existing *controller* or to wait for an invitation to join a *controller*. The various data are updated accordingly.

Afterwards, the nodes reach the *Ready* state where they contribute to service executions. If any change (e.g. a node leaving, a node joining) occurs, certain nodes (e.g. *controllers*) start the reorganize process to maintain the logical structure of the overlay network and return to the steady state *Ready*. Other nodes of a different *type* may be involved in the reorganization process. The node that wishes to quit moves to the leaving state and then disconnects. The *Leaving* state also deals with unexpected failures.

To illustrate this behaviour more clearly, we present the complete state diagrams of the overlay network entities (i.e. wrapper, connector, session repository and controller). Conditions are between brackets, question marks indicate message reception and exclamation marks indicate outgoing messages.

5.5.2.1 The wrapper state diagram

Figure 5.8 presents the state diagram of the *wrapper*, where *ControllerExist* equals 1 if a controller is present in the network at that moment, and 0 if not.

First the node gets the list of existing nodes in the MANET. If there is no *controller* in the network then the *wrapper* moves to an idle state waiting for a message from a joining *controller*. If it finds a *controller A* in the network it sends out a *join* message, waits for a reply and updates its data. The reply may be either an *ok,* meaning that *controller A* accepts the *join,* or a *Refers,* meaning that *controller A* redirects the node to another *controller*. Now the *wrapper* is ready to participate in the service execution. The *wrapper* can receive a *Refers*, an *Info* or a D*isconnect* message for reorganization purposes. The *wrapper* receives the R*efers* message when it is in the logical control area of a failed *controller* (i.e. a leaving or a crashed *controller*). The *Info* message is received from a *controller* when that *controller* gets the wrapper's address as a reply to a *Request_Node* message. A *Disconnect* is received from the last *controller*. When this *controller* leaves, it sends a *Disconnect* to its managed nodes.

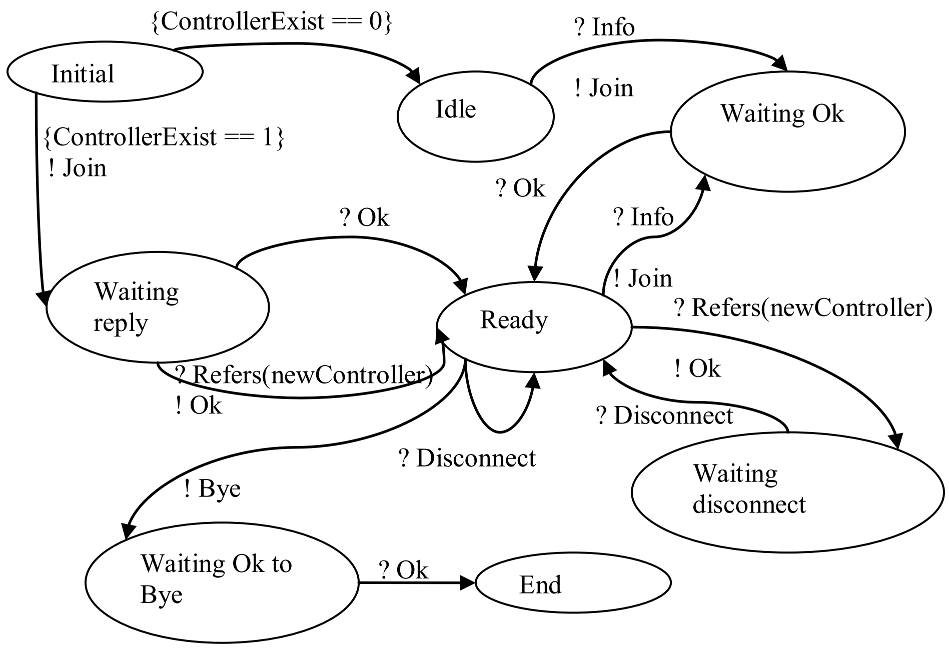


Figure 5.8: The wrapper state diagram

Upon reception of these messages in the *Ready* state, the *wrapper* updates its data, replies and eventually returns to the *Ready* state. The service execution, if any, is resumed.

To leave the network properly, the *wrapper* sends a *bye* message to its *controllers* and waits for the reply from its default *controller* before it disconnects.

5.5.2.2 The connector state diagram

The connector state diagram is similar to the wrapper state diagram. However, there are some fundamental differences. Actually, a *connector* can be managed by one and only one *controller* at a given time. Thus, a *connector* can only accept one join invitation. It has to verify if it is already connected to a *controller* before it accepts. Furthermore, for any reorganization purpose it should disconnect from its current *controller*, if possible (i.e. it is not involved in a service session), before it connects to another *controller*. Figure 5.9 shows the *connector* state diagram.

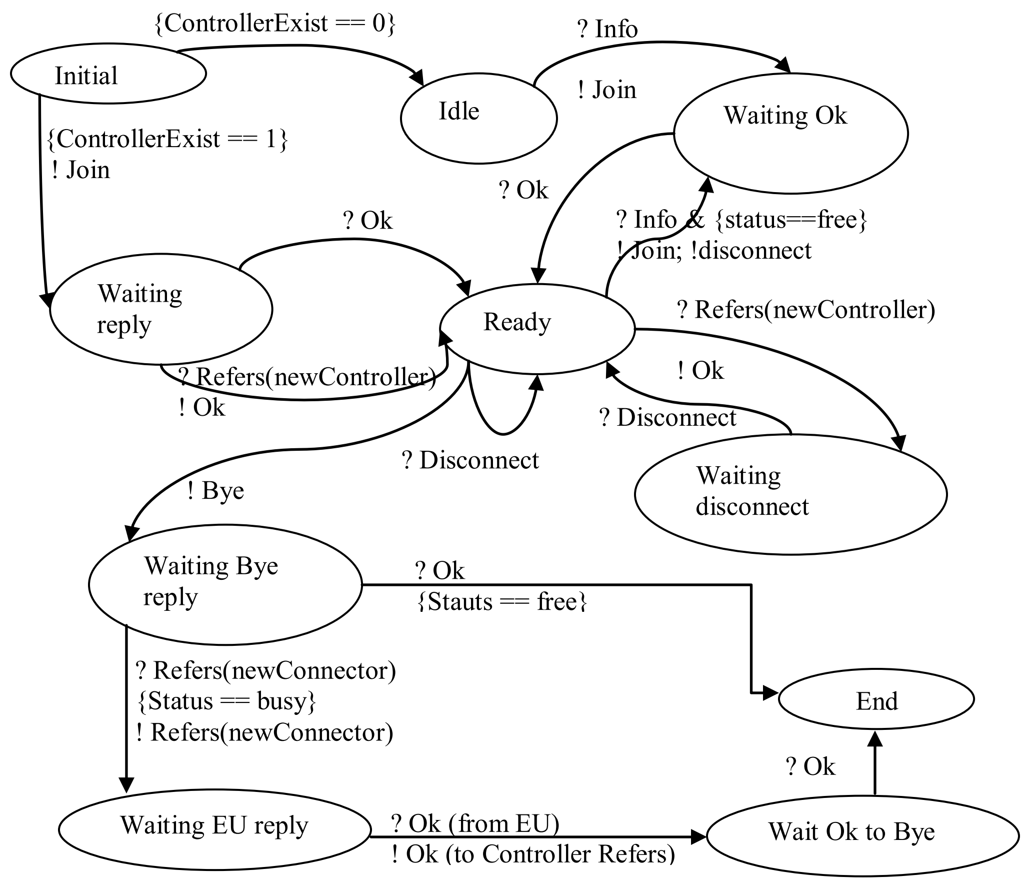


Figure 5.9: The connector state diagram

Another fundamental difference between a *wrapper* and a *connector* is the leaving process. Indeed, when a connector decides to leave it sends a *Bye* to its *controller*. If a connector is free (i.e. is not involved in a service session) then it receives an *Ok* and quits. However, if a *connector* is involved in a service session, it receives the address of an alternative *connector* via the *Refers* message. The leaving *connector* then informs the end-user of the alternative using *Refers*, waits for an acknowledgement (i.e. *Ok* message), sends an *Ok* to the received *Refers* and then quits.

5.5.2.3 The session repository state diagram

Figure 5.9 presents the state diagram of the *session repository* entity. Some parameters are needed to express the conditional transitions. Therefore, *SR* refers to Session Repository, *Ctr* refers to Controller, *N_SRs* refers to the number of SRs in the overlay network at that moment, *list_SRs* refers to a list of addresses of SRs and *k* is a counter.

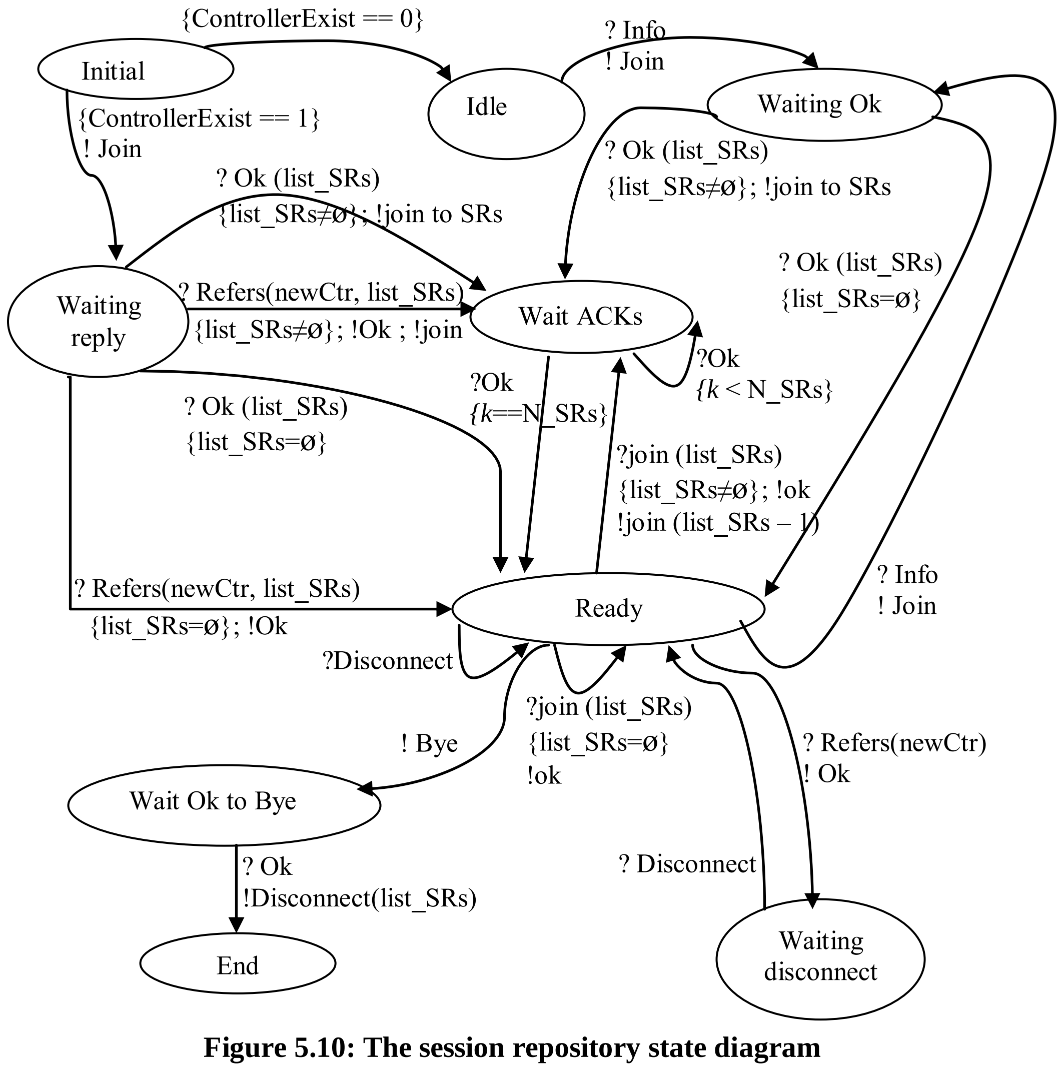


Figure 5.10: The session repository state diagram

Session Repositories (SRs) have to establish a full mesh connection in order to exchange their sessions and applications information. Therefore, when an SR joins the overlay network, if *controllers* (Ctr) are founded it randomly joins one of them. The chosen *Ctr* replies/accepts using the *Ok* message, or it redirects the SR to a new *controller* (newCtr) using the *Refers* message. However, in both cases the reply also contains the list of SRs available in the network. The SR then sends a join to all the SRs on the list and moves to the *Wait ACKs* state. There, it waits for acknowledgements (i.e. *Ok* messages). When all of these have been received, the SR moves to the *Ready* state. If no SR existed before in the joining session repository, the list is empty. In that case, the behaviour is similar to that for the *wrapper* and the *connector*.

The joining SR moves to an *Idle* state when no *controller* is found. Then the SR waits for a first *controller* to arrive. When this first *controller* replies to a *join* message it should include a list of the existing SRs. Since no connection has yet been established, the *controller* sends an empty list to all the SRs except for a chosen one (e.g. the one with the highest IP). The selected SR runs an algorithm to establish a full-mesh link. The motivation for this procedure is to avoid duplicate messages between SRs when establishing full-mesh links.

The algorithm is executed by the SR when it is in the *Waiting Ok* state or in the *Ready* state and it receives a non-empty list (*L*) of SRs. In these cases, the SR that runs the algorithm chooses a session repository, say *SR1*, from the list *L* (e.g. the one with the highest IP) and removes that one from the list. The resulting list is *L'*. It then sends a *join* message with an empty list to all the SRs on the list *L'*, and sends to *SR1* a *join* message with the *L'* list (i.e. the initial list except for the chosen SR). The initial SR then moves to the *Waiting ACKs* state. This is the *full mesh connection algorithm*. At a given round of this algorithm, the list will be empty and each SR will have a link with all the other SRs.

Figure 5.11 shows the *SR full mesh connection algorithm.*

```
Let :
L_SR: List of session repositories
SR(i): the ith session repository ; SR(0) = this session repository
S_SR: the selected session repository
Empty_L: empty list

Input = L_SR   ;  Output = none
Start
    Remove SR(0) from L_SR
    S_SR = highest_IP (L_SR)
    Remove S_SR from L_SR
    For i  in L_SR
        Send join(Empty_L)  to SR(i)
    End For
    Send join(L_SR)  to S_SR
End
```

Figure 5.11: The SR full-mesh connection algorithm

In the *Ready* state, *Refers*, *Bye*, *Disconnect* and *Info* messages are handled the same way as for the *wrapper* state. The *Disconnect* may be received from either the last leaving *controller* or from a leaving *session repository*. Information is updated accordingly. When a *join* is received from an SR, the associated list of SRs is checked. If the list is empty then the message is acknowledged and the SR remains in the *Ready* state. However, if the list is not empty, the SR uses the *full mesh connection algorithm* to send the corresponding *join* messages and moves to the state *Wait ACKs*.

5.5.2.4 The controller state diagram

The *controller* is the entity responsible for managing the overlay network nodes (i.e. nodes of type^{*+}). Therefore, it has a complex state diagram. For clarity, we present the *controller* state diagram in three separate parts: *Joining*, *Ready* and *Recovery*. *Joining* illustrates the *controller*'s behaviour when it first comes into the MANET. The *Ready* part presents the *controller*'s behaviour in response to events (e.g. nodes joining, nodes leaving). The *Joining* and *Ready* parts illustrate the self-organization aspect of the overlay network. The *Recovery* portion shows how the *controller* acts prior to its departure or when it detects unexpected node failures. This part illustrates the self-recovery aspect of the overlay network.

The global *controller* state diagram is obtained by sequentially combining the three parts. States with the same name refer to the same state.

Figures 5.12, 5.13 and 5.14 illustrate the different parts, where: *N_Nodes* is the number of overlay nodes in the network at a given time, *N_Ctrs* is the number of existing *controllers* in the network at a given time, *L_SRs* is the list of existing *session repositories* and *k* is a counter.

- **The controller's *Joining* part**

Figure 5.12 presents the state diagrams of the controller *Joining* part.

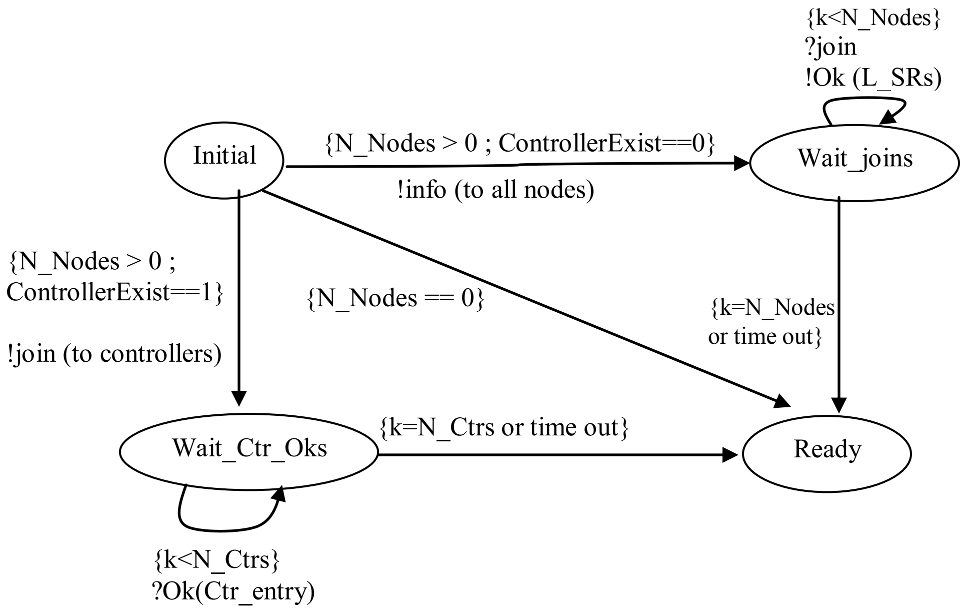


Figure 5.12: The *Joining* part of the controller state diagram

When a *controller* comes in it may be the first node in the overlay network, in which case it moves directly into the *Ready* state. However, some overlay nodes may already be in the network. In this case, if the existing nodes are all of type^{*+} (i.e. no *controller* has joined before) then the *controller* invites those nodes, waits for the corresponding *join* message, acknowledges them and moves to the *Ready* state. The *controller* includes the list of existing *Session Repositories* (SRs), if any, in only one acknowledgement to a chosen SR, which will execute the *SR full mesh* algorithm, as explained previously.

If a joining *controller* is not the first one in the MANET, it joins the existing *controllers* and waits for the replies before it moves to the *Ready* state. The *controllers'* replies contain entries that constitute the *controller's* data table, as shown in Table 5.2.

- **The controller *Ready* part**

The *Ready* part is illustrated in Figure 5.13. For clarity reasons we split the figure into two pieces, Figures 5.13.a and 5.13.b. The former basically presents the *controller* handling joining nodes, while the latter mainly shows the *controller* handling leaving nodes. In these figures, *accept* equals 1 if the *controller* decides to accept a joining node, equals 0 if it decides to redirect it, and *reply* is true if the controller can reply, but false if not. Indeed, in Figure 5.13.a the *controller*'s behaviour depends on the type of the joining node. If *controller A* receives a *join* from another *controller* then it sends an *Ok* to that node with the information about the nodes associated with *controller A*.

If *controller A* receives a *join* from a node of type^{*+} and accepts to add it to its logical control area, then it sends an *Ok* and informs the other *controllers* in order to update their tables. The *Ok* message is sent with the list of existing session repositories when the joining node is a session repository. However, if *controller A* decides to redirect the *join* message to another *controller,* then it informs that *controller* using the *Add* message. After receiving the *Add* acknowledgement, *controller A* informs the joining node about its new *controller* using *Refers*. The *Refers* message contains the list of existing SRs when the node considered is an SR.

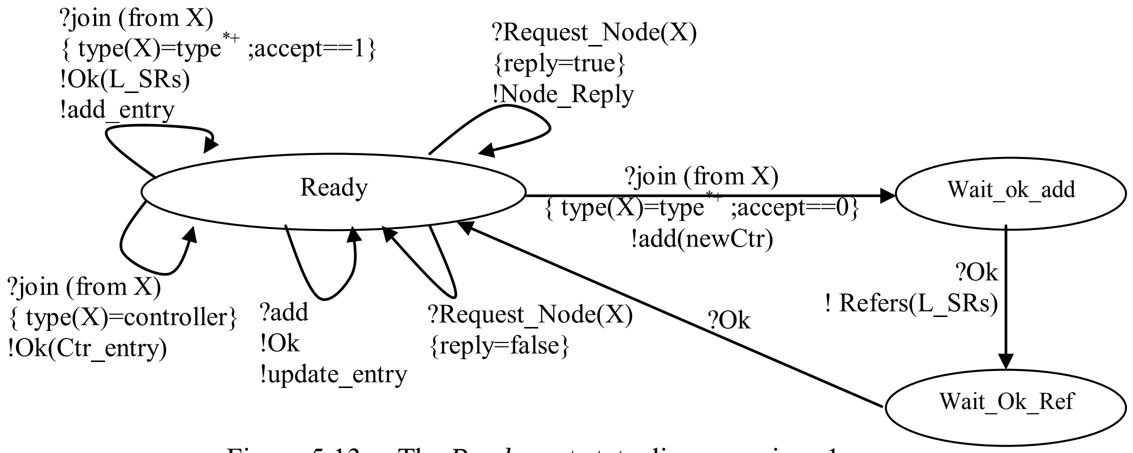


Figure 5.13.a: The *Ready* part state diagram: piece 1

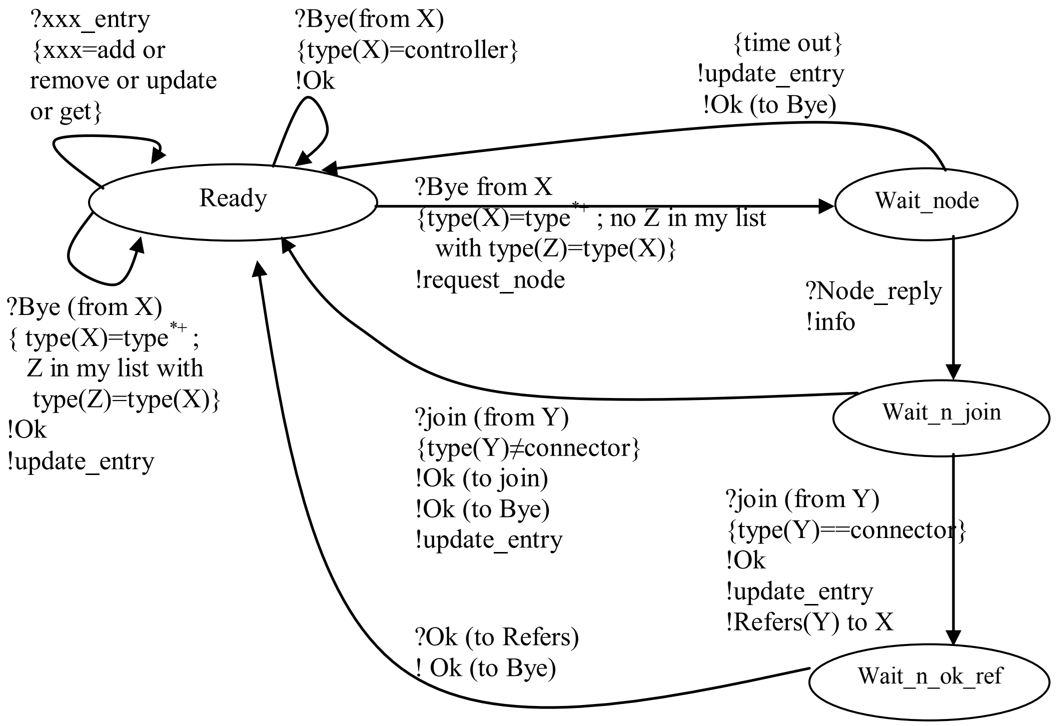


Figure 5.13.b: The *Ready* part state diagram: piece 2

Figure 5.13: The *Ready* part of the controller state diagram

Upon receipt of an *Add,* the *controller* sends an *Ok* and asks the other *controllers* to update their tables accordingly. Finally, a *controller* may or may not reply to a *Request_Node* message. If the *controller* has the requested node in its logical control area it responds, otherwise it ignores the message.

In Figure 5.13.b, the *controller* manages the expected failures of nodes (i.e. when nodes depart voluntarily). The simple cases are when *controller A* receives a *Bye* from another *controller* or from an overlay node *X* of type^{*+} such that there is a node of the same type as node *X* in the logical control area of *controller* A. In this situation *controller A* does not need to request a node and simply acknowledges the *Bye*. When the leaving node is of type^{*+} it sends an update message to the other *controllers*.

In the other cases, the *controller* should request a node of the same type as the one that is leaving. Since a SIP servlets engine cannot exist unless a *controller* is connected to at least one *connector*, one *wrapper* and one *session repository*, requesting a node has two advantages. First, it ensures service continuity and second, it allows a *controller* to form a distributed SIP servlets engine.

After requesting a node, the *controller* either receives a reply or times out. A reply is only received when a node of the same type is available in the network. That node is then invited to join the *controller* which sent the request. At that level, if the leaving node is a *connector C*, the *controller* also sends a *Refers* message with an alternative *connector* (i.e. the one received in the reply) to *connector* C. This latter then forwards this message to the end-users as an alternative access point to the SIP servlets engine. Finally, the *Controllers'* data tables are updated.

▪ **The controller *Recovery* part**

This part handles the volunteer departure of a *controller*. Furthermore, it illustrates how the controller acts upon unexpected failure detection. Figure 5.14 shows the recovery part of the *controller* state diagram, where: *last_Ctr* is true if the *controller* is the only *controller* in the network when it decides to leave and false if not; and *Card(X)* is the number of *controllers* the node *X* is linked to. *Card(X)*=1 means that the node *X* is connected to only one *controller*. *L_Card_1* is the list of nodes *Y* such that Card(Y) = 1, while *L_Card_n* is the list of nodes *Y* such that Card(Y) > 1. *N_S_Ctrs* refers to the number of selected *controllers*. A *controller* is selected through the decision algorithm for assigning nodes. Finally, the parameter *Crash* = 1 means that we are in the case where the *controller* is handling another *controller*'s crash (i.e. an unexpected failure).

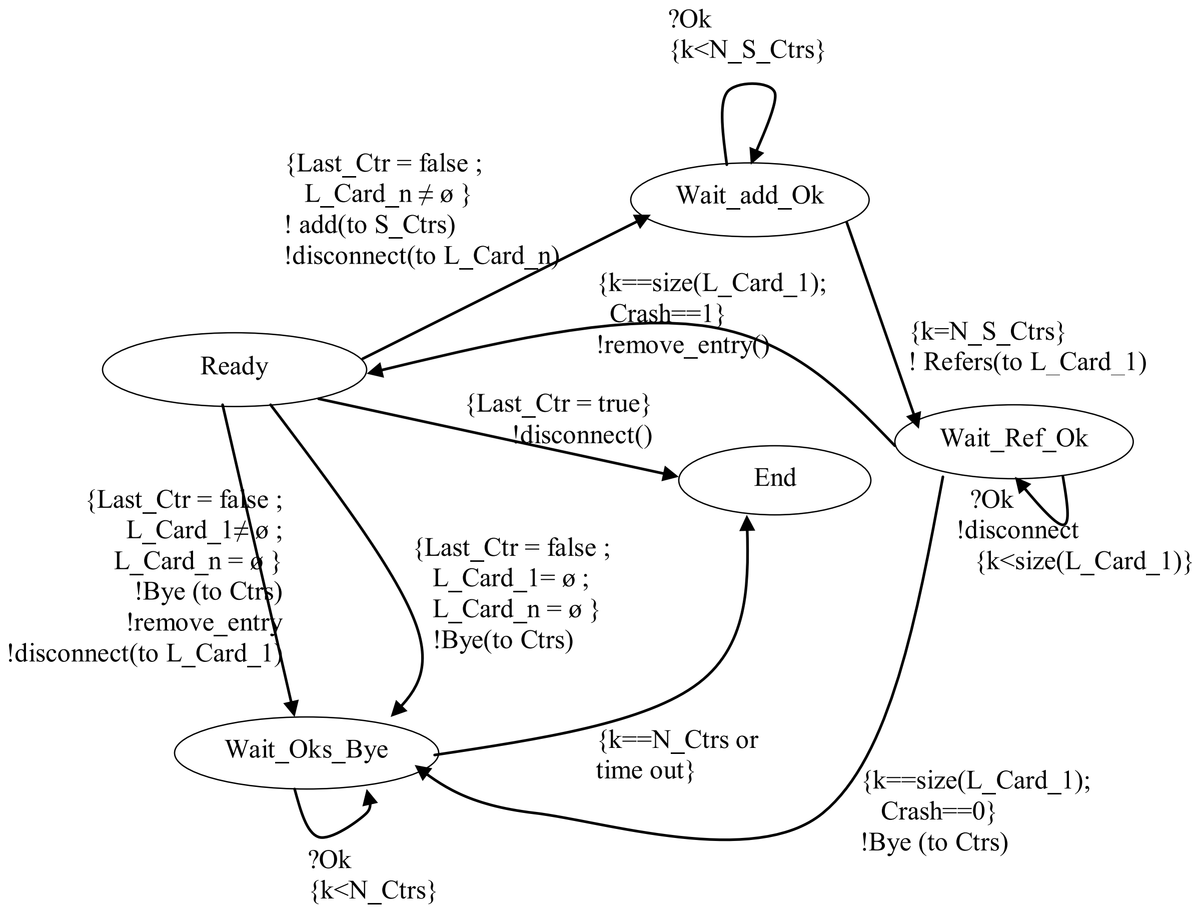


Figure 5.14: The *Recovery* part of the controller state diagram

A *controller* that decides to leave has to assign its nodes to the existing *controllers*. However, if there is no other *controller* in the network at the moment of its departure the *controller* informs its managed nodes via a *Disconnect* message and quits. Furthermore, not all managed nodes are assigned. Actually, nodes that have the leaving *controller* as a unique *controller* (i.e. no other *controller* is managing that node) are the only ones to be assigned. Therefore, the remaining nodes (i.e. nodes with more than one *controller*) will be informed using the *Disconnect* message so they can update their table.

When no node reorganization is required, the leaving *controller* sends a *Bye* to the existing controllers and waits for the acknowledgement. However, when some nodes need to be assigned, the *controller* runs the decision algorithm to select the target *controllers*. It then sends *Add* messages and waits for responses. Afterwards, it informs each node about its new *controller* using the *Refers* message. When the acknowledgment is received the *controller* is allowed to send *Bye* to the other *controllers.* Data tables are updated accordingly.

For crash detection, the *controller* detects not only its nodes' crashes, but also the other *controllers'* crashes. When a node of type^{*+} crashes, its controllers detect it. Therefore, they act as if they have received a *Bye* from that node (see figure 5.13). If another *controller* crashes it is detected by all the other *controllers*. However, it is the one with the highest IP address (i.e. a temporary head) that initiates the recovery. Basically, the procedure is similar to a *controller* leaving procedure (figure 5.14). Indeed, the concerned states are *Ready*, *Wait_add_Ok* and *wait_ref_Ok*. The temporary head plays the role of the crashed *controller* that decides to leave. At the *wait_ref_Ok* state the temporary head goes back to the *Ready* state instead of sending a *Bye*.

The three Figures, 5.12, 5.13 and 5.14, present the behaviour of the *controller*. Compared to the global system behaviour (Figure 5.7), the *Joining* part corresponds to the *Joining* state, *Ready* part matches *Ready* and *Reorganize* states and *Recovery* details the *Leaving* state.

5.6 Illustrative scenarios

This section presents examples of flow diagrams illustrating some of the cases discussed above. A scenario for self-organization and two scenarios for expected and unexpected failures are presented.

5.6.1 Self-organization

Figure 5.15 shows the interactions between the overlay nodes when a node of type^{*+} (in this case a *connector*) joins the network.

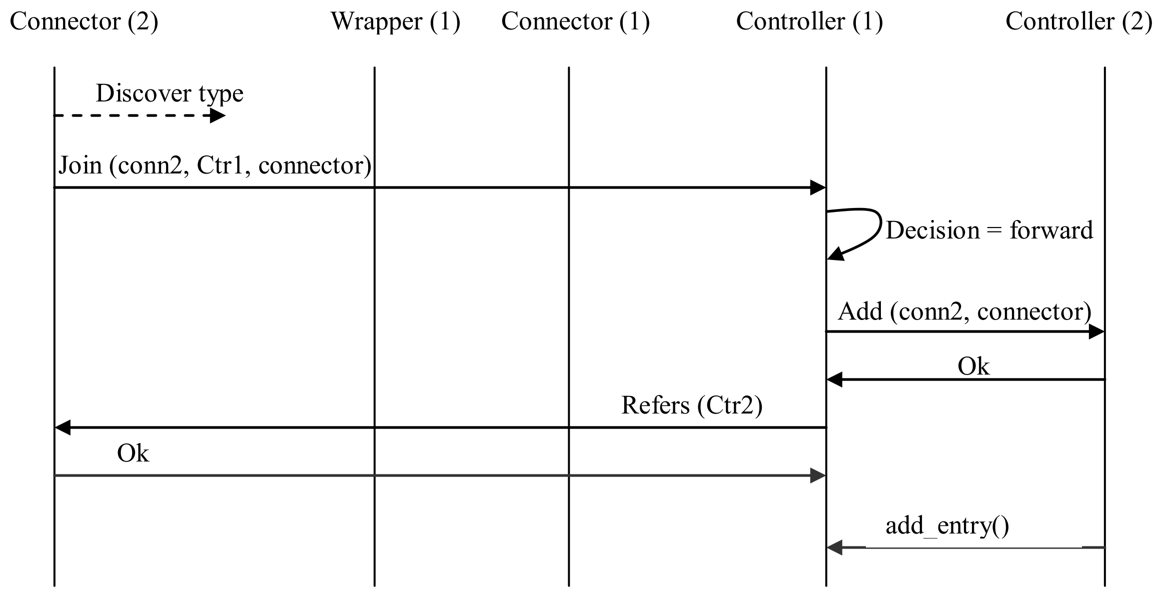


Figure 5.15: Interaction following a connector joining the overlay network

In this scenario we assume that Connector(1) and Wrapper(1) are under the control of Controller(1) (*Ctr1*), while Controller(2) (*Ctr2*) has no overlay nodes attached to it.

When Connector(2) (conn2) joins the overlay network, it discovers the list of overlay nodes (Connector(1), Wrapper(1), Controller(1), Controller(2)) and then chooses randomly to join *Ctr1*.

Upon the reception of the *join* request, Controller(1)(*Ctr1*) acts temporarily as a *head* for the group of *controllers*. It then verifies the list and the type of the overlay nodes attached to each existing *controller*. In this example, *Ctr1* can see that it already controls a *connector* while Controller(2) has no overlay node under its control. Therefore, to give every *controller* the opportunity to play its role and form a SIP servlets engine, Controller(1) decides to redirect the *join* request to Controller(2). It sends an *add* request to the chosen *controller* and informs Connector(2) of this operation using *Refers*. At the end, Controller(2) takes control of Connector(2). The other controllers are informed to update their data table.

5.6.2 Self-recovery

First let us consider a voluntary departure. In this scenario, Controller(1) controls Wrapper(1), and Controller(2) controls Wrapper(2).

Wrapper(1) decides to leave. Therefore, it sends the *Bye* request to its *controller*. Controller(1) then multicasts a request for a free wrapper to the community of *controllers* and receives a reply with the address of the available *wrapper*.

Controller(1) invites Wrapper(2) to join it via the *info* request. Wrapper(2) then joins Controller(1), which multicasts an *update_entry()* to the other *controllers* to update their tables.

Figure 5.16 presents the corresponding flow diagram.

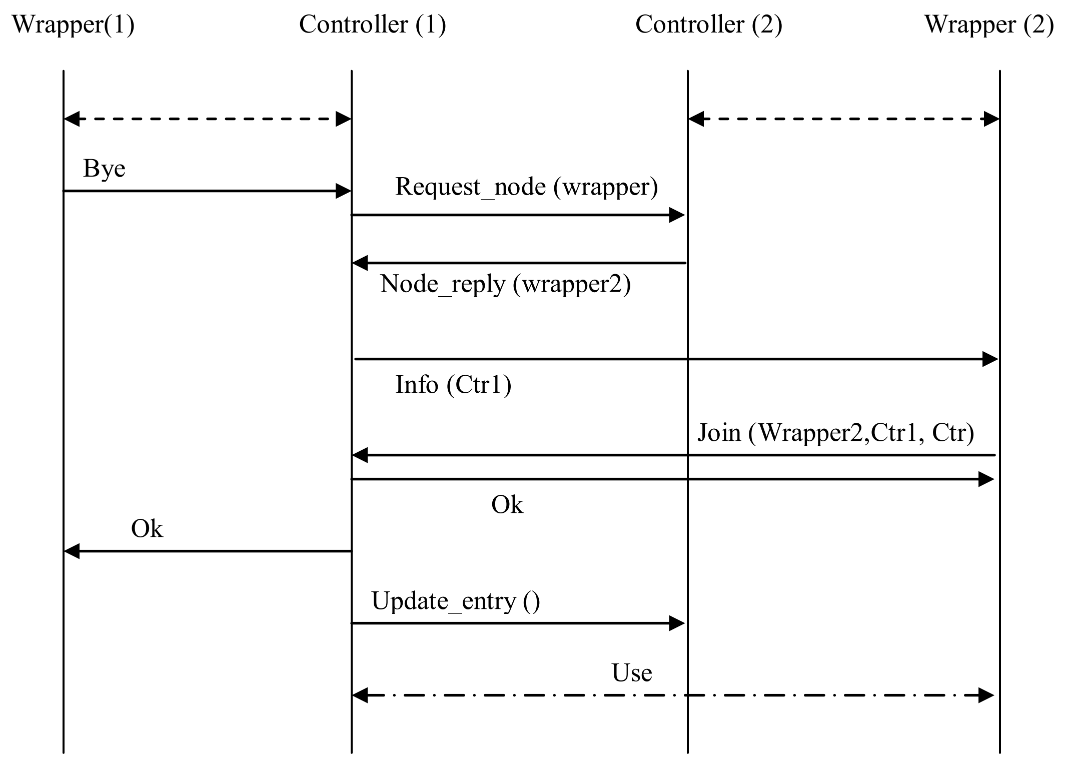


Figure.5.16: A wrapper voluntarily leaving the overlay network.

Figure 5.17 illustrates an unexpected *controller* failure. In this example, Controller(1) controls Wrapper(1), Controller(3) controls Wrapper(2) and controller(2) has no node. Controller(1) crashes suddenly.

Since all the controllers know each other's address, the head election is done automatically after the failure is detected. The temporary head, say Controller(3) in this example, decides to which *controller(s)* each node of the failed *controller* will be assigned. Controller(3) runs the decision algorithm to balance nodes among *controllers.* As a result and taking the *controllers'* capacity into account, Wrapper(1) has only one *controller* that crashes. Then, it must be assigned to an alternative *controller*.

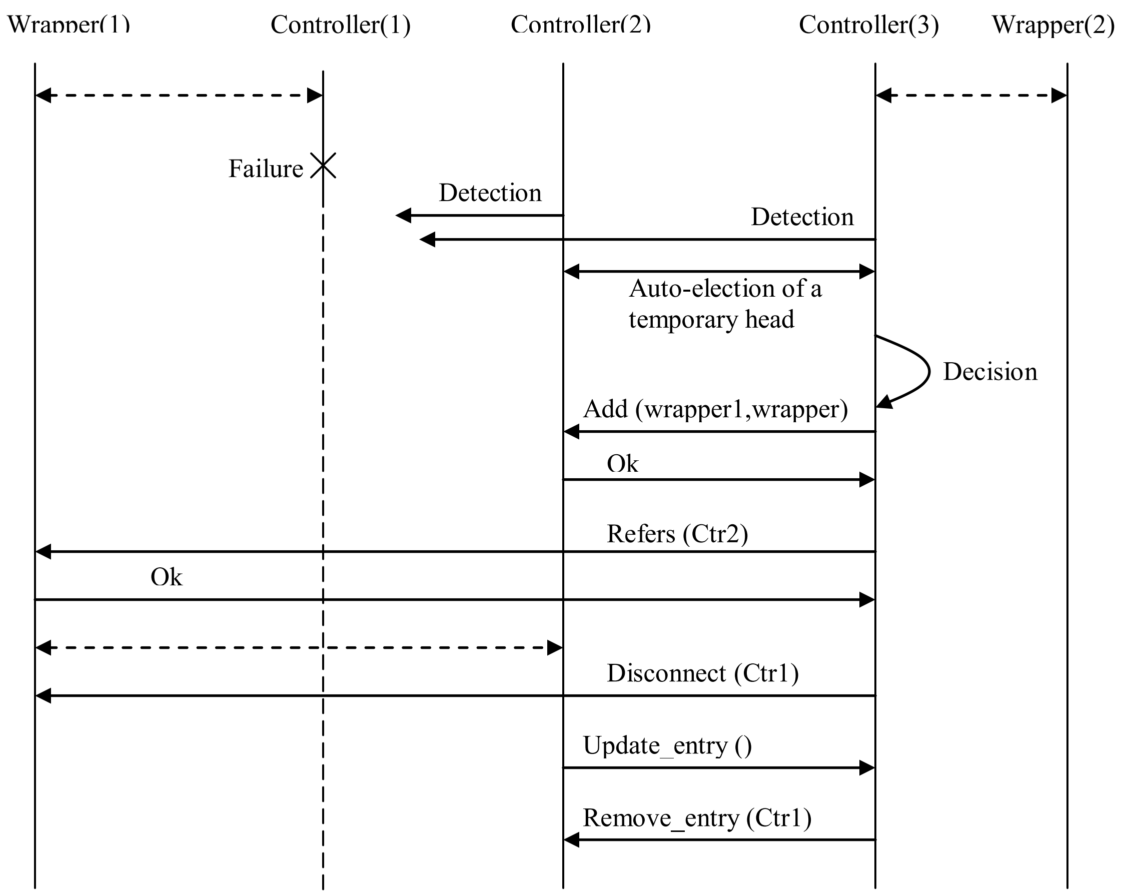


Figure 5.17: An unexpected *controller* failure.

Controller(3) assigns Wrapper(1) to Controller(2) and informs Wrapper(1) that its new *controller* is Controller(2). The temporary head also instructs Wrapper(1) to disconnect from Controller(1) since it is no longer available. This is done by updating Wrapper(1)'s table. The *controllers* then update their tables accordingly.

5.7 Summary

In this chapter we have proposed an overlay network for service execution environment in MANETs. It is based on a distributed SIP servlets engine. The motivations behind the proposed architecture have been discussed and an introduction to overlay networks was provided.

The overlay architecture has been depicted in detail. The architectural principles and assumptions were presented and the architecture design discussed. Furthermore, the overlay nodes have been described and procedures to construct, maintain and re-organize the overlay architecture elaborated. A protocol for the overlay network operations has been proposed. The data format of the exchanged information and the protocol messages were discussed. The corresponding state diagram for each overlay node has been elaborated. Finally, scenarios illustrating some examples of the flow diagram were described.

CHAPTER 6: A SIP servlets service provisioning architecture for integrated 3G/MANET networks

Integrated 3G/MANET networks have been explored with great interest thanks to their numerous benefits. However, the service aspects of these networks remain unexplored. This chapter proposes an architecture that is based on the SIP servlets paradigm for service provisioning in Multihop Cellular Networks (MCNs). The chapter starts with an introduction to the integrated 3G/MANET service provisioning, followed by a description of the SIP servlets framework in IMS. Then, it presents an exhaustive view of high-level architectural alternatives for service integration based on SIP servlets. The alternatives are discussed and the most interesting ones are identified. A detailed architecture is then proposed to realize one of the most promising alternatives.

6.1 Introduction

The integration of 3G and MANET networks is an important application of the 4G vision. The main goal behind this integration is to create a new network that has the advantages of both MANET and 3G networks. Indeed, MANETs are known for their ease of deployment, low cost, high bandwidth and multi-hop routing, while 3G are infrastructure-based, easy to manage, have a billing system and take security issues into account. Therefore, there has been more than enough justification for professionals to elaborate solutions for this integration. Indeed, MCNs enable new business opportunities by opening the 3G network to MANET users and vice versa. Furthermore, potential performance gains are expected by taking advantage of the high throughput of MCNs, and service execution times may be enhanced with an appropriate integration solution.

3G/MANET integration for service provisioning entails the choice of the 3G architecture and the service provisioning framework. The IMS network is considered since it is a 3G standard based on SIP and its deployment is growing. It is a promising architecture for next generation services. Furthermore, we propose an integrated architecture based on a SIP servlets framework for service provisioning, since SIP and SIP-based protocols are the prime signalling protocols for 3G, MANETs and integrated 3G/MANETs. In addition, SIP servlets are a part of IMS service provisioning, and we have already proposed a SIP servlets-based architecture for providing services in MANETs.

6.2 SIP servlets framework in IMS

The SIP servlets service provisioning framework in MANETs was discussed in chapters 4 and 5. This section details the framework usage in IMS architecture.

Service provision in IMS involves three main entities: the HSS, the CSCF and the SIP AS. The main data stored in the HSS is composed of user identities, registration information and security information. However, the *user profile* is the most important part because it determines the services that will be provided to each user and states the rules for service triggering. A *user profile* contains a set of information related to a particular user. The initial filter criteria is the most important element for service provisioning because it describes when and which services are to be invoked, under which conditions and in which order. The S-CSCF downloads the user profile or part of it (i.e. the initial filter criteria) from the HSS when the user registers for the first time with that S-CSCF. This same S-CSCF evaluates the initial filter criteria and contacts the proper application server. The communication between the HSS, the S-CSCF and the AS is accomplished through standardized IMS interfaces. Figure 6.1 shows a simplified view of the SIP servlets service provisioning model in IMS.

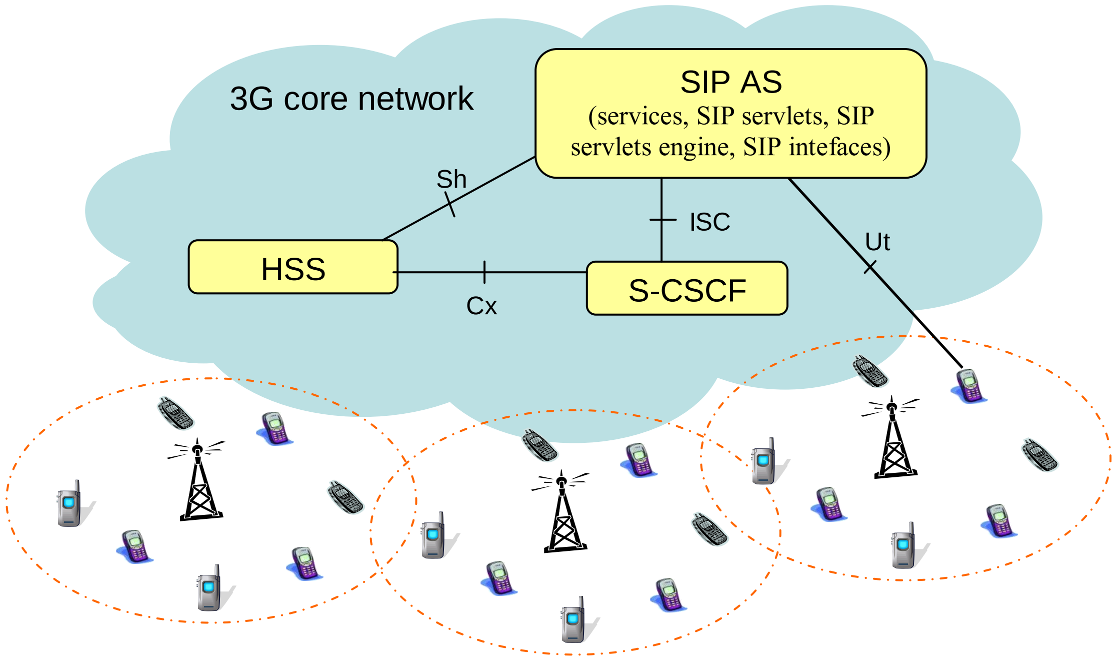

3G core network
SIP AS
(services, SIP servlets, SIP servlets engine, SIP intefaces)
Sh
ISC
Ut
HSS
S-CSCF
Cx

Figure 6.1. Simplified view of the SIP servlets service provisioning model in IMS

6.3 SIP servlets-based service provisioning in Multihop Cellular Networks: high-level architectural alternatives

The SIP servlets-based service provisioning process requires four key entities: a service, a party interested in that service (i.e. the user equipment), SIP servlets and a SIP servlet engine. Any of these entities can be hosted either in the MANET or in the 3G portion of the MCN.

The architectural alternatives are defined by the different possibilities for hosting these entities. Table 6.2 presents all the possible options.

Entities	**MCN sub-network type**					
Service	3G	MANET	3G	MANET	MANET	3G
User equipment	3G	3G	3G	MANET	MANET	MANET
SIP servlets	3G	3G	MANET	MANET	3G	MANET
SIP Servlets engine	MANET	MANET	MANET	3G	3G	3G

Table 6.1. All the possible options for hosting the SIP servlets framework in MCNs

However, the SIP servlets' location has no significant impact on service provisioning. Indeed, they are loaded at run time from their respective locations. Any file transfer protocol can be used. Therefore, we will focus on the service, user equipment and SIP servlets engine locations.

We classify the alternatives according to where the service is executed (i.e. where the SIP servlets engine is hosted). This gives us two categories. In the first category the SIP servlets engine is hosted in the MANET while in the second category it is hosted in the 3G. In each category three alternatives can be considered; these refer to the allowed options for hosting the remaining entities. For instance, when the service is executed in 3G, the alternatives are: *user equipment and service logic hosted in the MANET*; *user equipment in the MANET and service logic in 3G*; and *user equipment in 3G and service logic in the MANET*.

We assume in each case that all of the interactions between the 3G and the MANET sub-networks are done via a new entity we call the Service Gateway (SGW). The alternatives are described and discussed next.

6.3.1 Services executed in the MANET portion

In this category, all the invoked services are executed in the MANET sub-network. In other words, the execution of the services provided by MANET service providers or by 3G service provider is performed in the MANET portion.

The MANET is seen as an execution environment, which is especially interesting for a network operator anticipating a performance. In fact, running a service in a MANET instead of in a 3G can speed up the service execution time: remote S-CSCFs and AS communications are avoided while peer-to-peer connections are promoted. Furthermore, this option can be used for load balancing when the 3G network nodes and particularly the ASs are overloaded.

Another impetus to run services in a MANET is when the connection to the 3G network is not reliable or it cannot be maintained for a long time. In battlefields or emergency situations, for example, it would be better to run a service in the MANET since the connection to the 3G cannot be guaranteed throughout the service execution time. This is practical when all the involved users are in a MANET. The different alternatives for this category are described below.

6.3.1.1 User equipment and service logic are in the 3G portion

This alternative is a remote service execution. A user in 3G can access his or her 3G services, but a service provider decides to run its service in a MANET. Therefore, the application server may contain the service logic only. The execution environment (i.e. the SIP servlets engine) is provided by a user in the MANET (i.e. SSEP). The service execution can be routed to a MANET via the service gateway. Figure 6.2 illustrates this alternative.

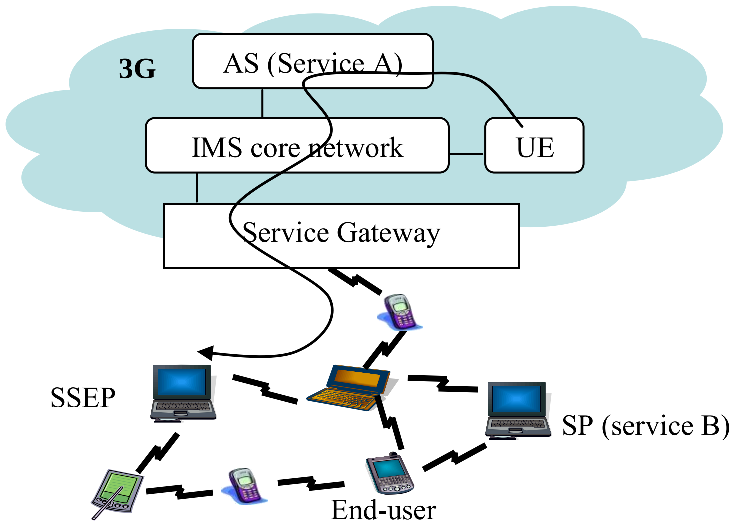


Figure 6.2. Service execution in MANET: UE and service logic in 3G

The end-user is in the 3G sub-network and invokes a service from its user equipment (UE). Then the service provider redirects the execution to the MANET. Several criteria can be defined and implemented in the AS to redirect a service execution to the MANET.

6.3.1.2 User equipment is in MANET and service logic is hosted in 3G

In this alternative, end-users in the MANET sub-network access and run 3G services. The services are hosted in the 3G network. This alternative is an interesting option for 3G operators to extend their network coverage using MANETs. Indeed, 3G users that are out of the network coverage can use the MANET sub-network to access their services and run them in the MANET. Furthermore, this option helps to achieve *service continuity*. Service continuity happens when a user moves from a 3G home network to a 3G visited network but the only connectivity between these two networks is ensured by a MANET. Figure 6.3 illustrates this alternative.

The end-user in the MANET sub-network accesses its 3G services using the application server (AS) hosted in the 3G. This access is done through the service gateway. The AS checks the criteria for service execution and decides to redirect the execution to the MANET. The SIP Servlets Engine Provider (SSEP) is then reached and the execution initiated. The service gateway hides the nature of the SIP Servlets Engine (SSE), which may be centralized or distributed.

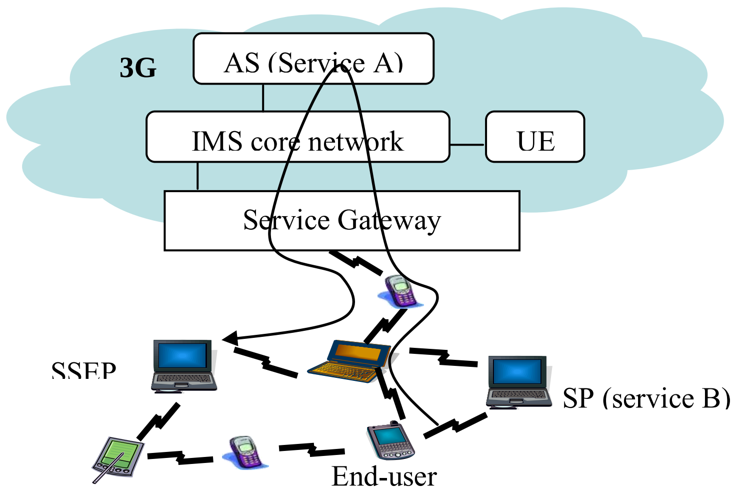


Figure 6.3. Service execution in MANET: UE in MANET and service logic in 3G

6.3.1.3 User equipment is in 3G and service logic is hosted in MANET

In this alternative the user is in the 3G network and the service is hosted in a MANET. MANET services can be provided either by the network operator or by individuals. This option is economically promising since it opens the 3G network to totally new services by allowing individuals in the MANET portion to provide a range of new services.

The 3G users discover the MANET services through the service gateway, which also plays the role of an application server providing all the services from the MANET.

When a MANET service is invoked, the S-CSCF redirects the request to the service gateway that forwards it to the appropriate MANET service provider.

Furthermore, users that move from the MANET portion to the 3G portion can access and run the services they have discovered in the MANET. This is achieved transparently thanks to the qualities of this alternative, illustrated in Figure 6.4.

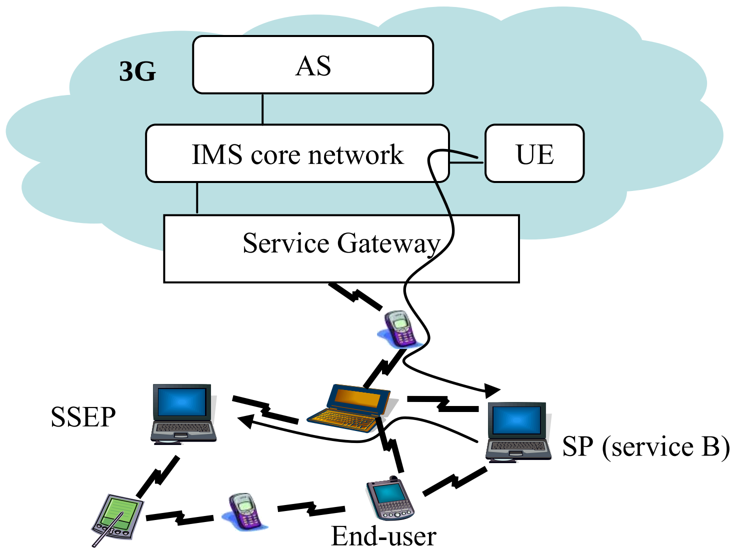


Figure 6.4. Service execution in MANET: UE in 3G and service logic in MANET

Using its user equipment (UE), the end-user in the 3G discovers and accesses the *service B* hosted in the MANET via the service gateway. The service is provided by a Service Provider (SP) in the MANET. The SP then contacts the SIP Servlet Engine Provider (SSEP), which executes the service.

6.3.2 Services executed in the 3G portion

In this category the services provided by MANET or 3G service providers are executed in the 3G sub-network. The network operators may decide to run a service in the 3G portion in order to save the MANET resources (i.e. bandwidth, devices' memory, processing and battery). By running services in the 3G portion, an operator ensures: better security, reliability, control over the service provision process, and frees the MANET resources from heavy processing. In particular, this category is attractive for services that require a high level of security. In such cases, it is better to run the service in a secure environment (i.e. a 3G sub-network) but at the same time the service can be provided by any user (e.g. a MANET service provider) which guarantees openness and service diversity. The possible alternatives under this category are described below.

6.3.2.1 User equipment and service logic are in the MANET

The service logic and the user equipment are in the MANET portion while the service is executed in the 3G network. Given that the service provisioning process starts in the MANET, the appropriate service publication and discovery mechanism is used to obtain the list of available services. The MANET is then considered as a service creation environment while the 3G is considered as a service execution environment. Figure 6.5 illustrates this alternative.

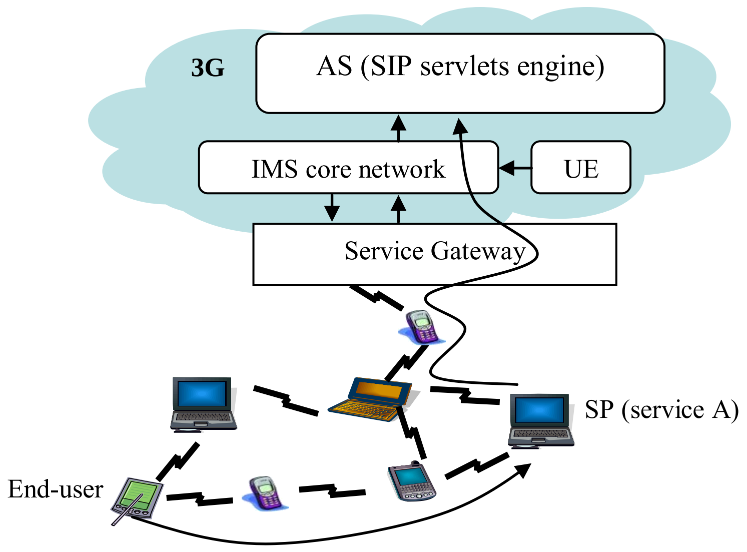


Figure 6.5. Service execution in 3G: UE and service logic in MANET

This alternative allows individuals in the MANET to provide innovative services without concern for execution environment issues (e.g. security, billing). The MANET will play the role of a service creation environment: an open environment for interested parties, while services are executed safely and with the required performance in the 3G.

The service provisioning starts in the MANET sub-network as described in chapters 4 and 5. When the service provider is reached it decides to run the service in the 3G sub-network. The service gateway ensures transparency and plays the role of an application server calling another application server (i.e. the one with the SIP servlets engine). A 3G service provider may also wish to provide an SSE as a service through its AS.

6.3.2.2 User equipment in MANET and service logic hosted in 3G

Users in the MANET sub-network access and run the services hosted in the 3G sub-network. Typically, this alternative allows users that are out of the 3G sub-network coverage to access and run their 3G services. It also permits service continuity since users in the MANET still have access to their 3G services. Figure 6.6 illustrates this scenario.

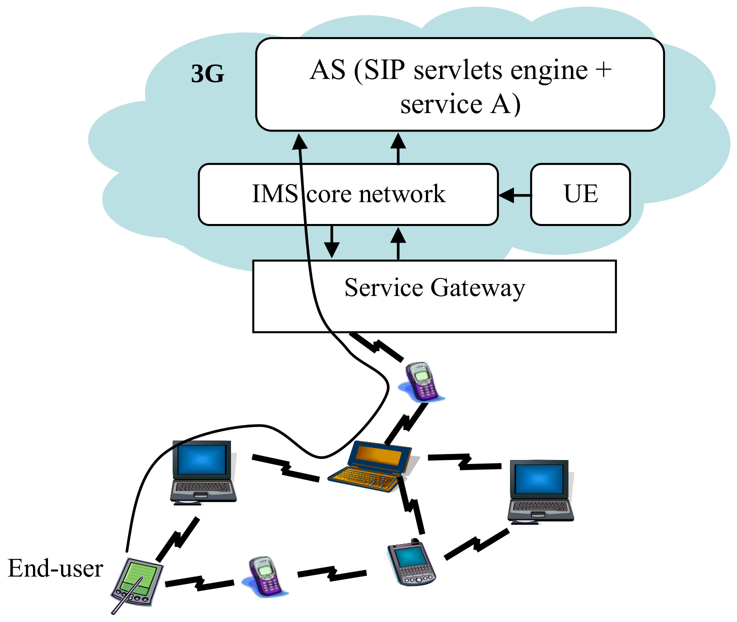


Figure 6.6. Service execution in 3G: UE in MANET and service logic in 3G

The MANET end-user accesses the 3G services it subscribed to via its user equipment (UE). The MANET sub-network ensures the connectivity while the service gateway plays the role of user equipment for the requested service. The AS evaluates the criteria to determine where to execute the service. Another option is to pre-configure the AS with a given location (i.e. 3G in this case). The AS runs the service locally, which may involve users in the MANET and/or in the 3G sub-network.

6.3.2.3 User equipment is in 3G and service logic is hosted in MANET

In this alternative the service is hosted in a MANET but accessed from 3G user equipment and executed in the 3G sub-network.

With this alternative, users in the MANET portion with very limited resources are allowed to provide services to users in 3G. The networks are thus opened to services developed by individuals with no special consideration for the execution environment (i.e. the SIP servlets engine). The execution is performed in the 3G portion, which preserves the limited resources of the MANET. This alternative is illustrated in figure 6.7.

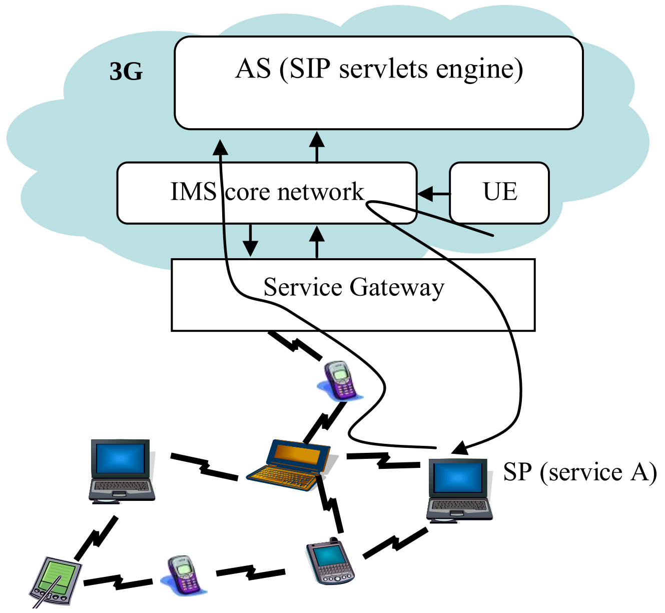


Figure 6.7. Service execution in 3G: UE in 3G and service logic in MANET

The 3G end-user discovers the services provided in the MANET sub-network using its user equipment. The discovery is done through the service gateway, which plays the role of an AS providing the MANET services. Another situation is when the end-user has moved from the MANET to the 3G and kept its list of discovered services in the MANET. Therefore, the service provider in MANET redirects the service execution to the 3G AS, based on specific criteria.

6.3.3 Alternatives analysis

As we have seen, each alternative responds to various needs. The network operator is free to configure the network with the desired alternatives according to its needs and the expected benefits. Table 6.2 summarizes the different alternatives.

	Alternative 1	**Alternative 2**	**Alternative 3**
Service execution in **MANET**	UE and service logic in 3G	UE in MANET and service logic in 3G	UE in 3G and service logic in MANET
Service execution in **3G**	UE and service logic in MANET	UE in MANET and service logic in 3G	UE in 3G and service logic in MANET

Table 6.2. Summary of the high-level architectural alternatives for SIP servlets-based service provisioning in MCNs

MCNs were traditionally deployed for coverage extension and throughput improvement. However, the proposed alternatives introduce new benefits for the MCNs. Indeed, from the network operator point of view, the benefits expected from the different alternatives include:

▪ *3G services invocation by end-users that are out of coverage*: The *user equipment in MANET and service logic in 3G* alternative is a 3G coverage extension using MANETs. This alternative allows end-users that are out of the 3G network coverage to access and invoke their 3G services. When the service is executed in a MANET, this scenario combines coverage extension with the advantages listed above regarding service execution in MANETs (i.e. ,more rapid service execution, load balancing, overcoming 3G/MANET link failures). Furthermore, the network operator maintains control over the service provisioning process.

▪ *Individuals offering services in 3G settings*: The *user equipment in 3G and service logic in MANET* alternative opens the 3G networks to a new brand of services and a new business model. Individual users can make services available in a 3G setting where new business opportunities are promoted. MCNs then become very interesting economically. However, the users in 3G should already know about the existing services in MANETs.

▪ *Speeding up the service execution process*: The *end-user and service provider in 3G* alternative with execution in a MANET is advantageous when performance is important. Indeed, remote S-CSCFs and application servers' communications are saved while peer-to-peer connections are promoted. This can avoid both bottlenecks and overloaded application servers.

▪ *Providing a reliable execution environment for users and service providers that are in the MANET*: The *end-user and service provider in MANET* alternative with execution in 3G allows individuals in a MANET to provide innovative services with no consideration for the execution environment issues. Therefore, executing services provided by users in a MANET becomes secure and reliable. Furthermore, the scarce resources of the MANET are conserved.

Tables 6.3 and 6.4 review the advantages of service execution in MANET and in 3G, respectively. The relevant MCN configuration is shown with its corresponding benefits.

	Advantages	**Relevant configuration**
Coverage extension	Service execution Speed up	All or most of users are in the MANET
	Avoid 3G-MANET connectivity problems	All the users are in the MANET
	Load balancing	Any 3G/MANET combination : users any where

Table 6.3. Advantages of service execution in MANET

	Advantages	**Relevant configuration**
Coverage extension	Security	Any 3G/MANET combination : users any where
	Save MANET resources	Any 3G/MANET combination : users any where

Table 6.4. Advantages of service execution in 3G

However, the different alternatives introduce several issues. The issue common to all the scenarios is the need to extend the application servers. In fact, services may be executed either in 3G or in MANET. Therefore, the network operator has to establish the criteria for service execution for each service or category of services and then choose to run them either in MANET or in 3G depending on the desired objective. For example, if the ASs are overloaded for a period of time, the network operator may decide to switch execution to the MANET as long as the situation continues. Similarly, if a given security level is required for a category of services then the execution may be routed to the 3G.

In certain scenarios the application servers do not need to use or even implement all of the SIP servlets functionalities. For instance, when the services are executed in the MANET, the AS does not need to use the SIP servlet engine functionalities and so it should be able to activate or deactivate some of its functions.

From the above discussion and from the network operator point of view, the most interesting category is when services are executed in the MANET portion. Furthermore, the MCN configuration that takes the maximum advantage of this category is when all or most of the users are in the MANET. Finally, the scenario where the services are offered by 3G service providers results in only small impacts on the existing 3G and MANET networks. In fact, all the security and reliability problems are avoided.

6.4 Provisioning services in MCNs when the end-users are in the MANET portion

We will focus on the alternative where the end-users are in the MANET and the services are hosted in the 3G but are executed in MANET. This section proposes a detailed architecture for this alternative.

This solution has to deal with several issues. The first issue is the application server criteria required to redirect the service execution to the MANET. The architecture has to define this criteria and the process by which the AS makes a decision. The second issue is related to load balancing. The solution should describe how to get load information and from which entities to get it. The third issue concerns the users' location. The solution should ensure that all users are in the MANET portion. However, where can this information be found? Furthermore, since the execution is done in the MANET, a distributed SIP servlets engine should be able to be used, as well as a centralized engine.

The proposed architecture is detailed next. Architectural assumptions and principles are discussed along with the architecture's functional entities and procedures. Furthermore, a scenario is proposed as illustration.

6.4.1 Architectural assumptions

Some assumptions must be made to keep the solution clear and simple, and to produce a sketch for a more complex and complete architecture. Therefore, we assume that the decision of running a service in MANET or in 3G is made by the 3G application server. This latter should have enough information to make the right decision for each service. Furthermore, since we are in the scenario where all the end-users are in the MANET, we assume that users' locations are stored in the 3G Home Subscriber Server (HSS). Actually, in terms of user location, the current 3G HSS contains, among other data, the *location information* data type. However, this information is related to the GSM/GPRS users' location and it does not consider MANET users' locations. Basically, this information indicates if a user is in the Circuit Service (CS) domain or in the Packet Service (PS) domain. By analogy we assume that the HSS contains information that mentions if a user is in the MANET portion of the MCN or in the 3G portion.

There are several ways to use SIP in MANETs: using clusters [40][107], full-mesh [107], using the underlying routing protocol [108] or using a distributed SIP [111]. Therefore, we assume that end-users in a MANET establish SIP sessions directly with other MANET users in a full mesh or by using clusters, since the full mesh is more suitable for small MANETs and the clusters approach has proven its efficiency and has been proposed for the integrated 3G/MANET.

The last assumption is that at any given time at least one SIP servlets engine is in the MANET. This assumption is of utmost importance since the services are executed in the MANET by the SIP servlets engine. Otherwise, the service execution cannot be processed in the MANET.

6.4.2 Architectural principles

The proposed architecture relies on several principles. Indeed, the architecture is based on a Service GateWay (SGW). An SGW is a functional entity that plays the role of a proxy when used by MANET end-users to access the 3G sub-network. For better flexibility, the SGW can be hosted either in the 3G or in the MANET portion. Furthermore, the 3G S-CSCF and the MANET SIP Servlets Engine (SSE) are connected to the SGW. This latter is treated as the entry point to the execution environment in MANET (i.e. the SSE). The SGW, therefore, should be involved in the service execution process when the SSE in the MANET will be used. In addition, for improved performance, several instances of the service gateway may be available in the MCN. The 3G S-CSCF can either discover the existing service gateways or be pre-configured with one or multiple gateways. In this work we assume the pre-configuration option. The different SGWs, when they are available, do not need to communicate with each other.

The service execution environment hosted in the MANET, namely, the SIP servlets engine, is to be provided either by MANET end-users which make it available for 3G use, or it is pre-installed by the network operator in a dedicated MANET node. The second option will ensure the availability of the SSE at any given time while allowing individuals to provide their own SSE. Finally, in order to introduce minimal impacts on 3G and MANET sub-networks, and since SIP is the major signalling protocol in these networks, the communication between the different functional entities of the proposed architecture is performed using SIP-based interfaces.

6.4.3 Functional entities

Six functional entities are involved in the proposed architecture: the End-User Agent (EUA), the S-CSCF, the Enhanced SIP Application Server (E-SIP AS), the Enhanced SIP Servlets Engine (E-SSE), the Enhanced Home Subscriber Server (E-HSS), and the Service Gateway (SGW). Figure 6.8 gives an overview of the architecture.

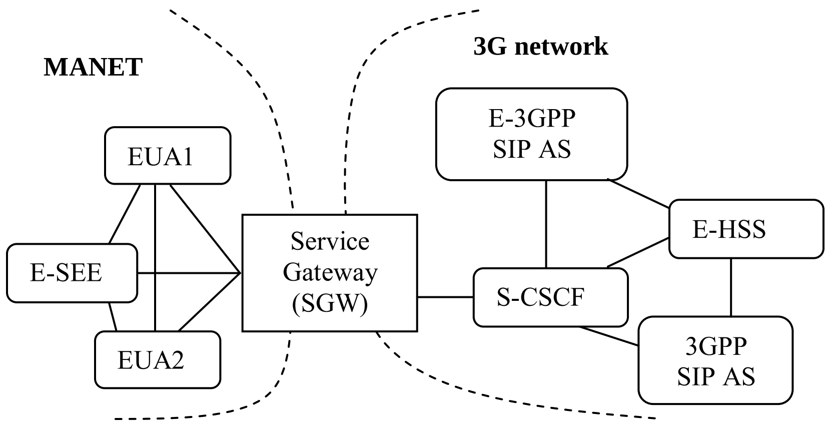


Figure 6.8. An overview of the proposed architecture

The EUAs are 3G subscribers that implement the application portion of the User Equipment defined in the 3GPP standard [60]. They are located in the MANET portion of the MCN.

The S-CSCF is the main entity in the 3G network defined in the 3GPP standard. No changes are required at the S-CSCF level since the interface to the service gateway is based on the SIP.

The 3G SIP application server, as defined by the 3GPP, hosts services, SIP servlets and the SIP Servlets Engine (SSE). However, in the alternative studied here, the service is executed in MANET. Hence, the SIP ASs may not use the SSE they own but instead use the SSE hosted in the MANET portion.

In our architecture we propose an enhanced 3G SIP AS (E-3GPP SIP AS) which can decide to run a service in the MANET portion even though it has a SIP servlets engine. Therefore, the enhanced AS implements *decision making logic* described in the next sub-section. This logic will allow the E-AS to dynamically choose to run a service locally or in the MANET. Furthermore, the E-AS requires a mechanism to collect the network load, which is considered as a criterion that affects the AS service execution decision.

In addition, the enhanced AS is also implementing an interface and a server-side software to allow an SSE hosted in the MANET to download the required SIP servlets for service execution.

The SIP Servlets Engine (SSE) is the entity described in [63] and is responsible for service execution, which is provided by a SIP Servlets Engine Provider (SSEP). The SSEP is either a MANET user or the 3G network operator. We extended the standard SSE with new functions. Indeed, the Enhanced SSE (E-SSE) implements several mechanisms. First, it provides a mechanism to register its SSE function with the SGW. In fact, since the SGW is the entry point to the MANET execution environment (i.e. the SSE) it should be identified by available E-SSEs. Then, these E-SSEs register with the service gateway. Therefore, the E-SSE should implement a SGW discovery function which is used prior to the registration process. Furthermore, in some cases the service gateway may need to discover the E-SSEs hosted in the MANET portion, and so the E-

SSE provides a publication function to make the E-SSE available for the SGW. An additional function provided by the E-SSE is a client-side software for downloading SIP servlets. Finally, since we are in a dynamic environment, a function to inform the SGW when the E-SSE gracefully leaves the network is also required.

The HSS is the 3G subscribers' database as defined in the 3GPP standard [60], augmented with the end-users location (i.e. 3G or MANET) as per our assumption, thus becoming an enhanced HSS (E-HSS).

The architecture introduces a new entity in the middle. This entity makes the service execution in MANET transparent to 3G users by hiding the SSE details. We call this entity the Service Gateway (SGW). The SGW is used to manage the SSEs, especially when multiple SSEs are offered. The mobility, availability and the addresses are maintained transparently. The SSE configuration, distributed or centralized, is handled and kept transparent by the SGW. Each SGW implements SIP to communicate with the S-CSCF and the SSE. The SGW is seen as an AS from the S-CSCF where services are executed or as 3G end-users that request 3G access. Furthermore, it implements publication and discovery functions to allow SSEs in MANET discover the service gateway and to discover the available SSEs in MANET, respectively. Finally, the SGW processes registrations and de-registrations from SIP servlets engines and periodically checks their availability.

6.4.4 Procedures

A SIP based interface is used between the different functional entities. However, two more protocols are required for publication/discovery and for file transfer to download SIP servlets files. Since we use the publication/discovery protocol in the MANET portion only, the protocol should take into account MANET constraints, especially resource limitation. Therefore, any discovery protocol suitable for MANET can be used. We chose the Pervasive Discovery Protocol (PDP) [98], a light-weight protocol designed especially for ad-hoc networks and to preserve bandwidth. As for SIP servlets downloading, we can reuse any relevant file transfer protocol. We opted for HTTP since it is already supported by 3G ASs.

The procedures related to our architecture for service provisioning take place at three different levels: before service execution, at the service execution runtime and at any given time.

6.4.4.1 Before service execution

The service gateway uses the publication and discovery protocol (e.g. PDP) to publish its presence in the MANET. When a SIP servlets engine provider comes in to the MANET or when the pre-installed SSE is activated by the 3G network provider, it uses the same protocol (i.e. PDP) to discover the SGW and registers the SSE with it. The SSE capacity and approximate Time To Live (TTL) are provided to the SGW. The TTL indicates the estimated time an SSE is willing to stay in the MANET. TTL is used as a guideline only. The SSE can update the TTL parameter in the SGW at any time via the push mechanism of PDP. The SIP REGISTER method can be used for SSE registration with the SGW. The capacity and the TTL are added to the REGISTER message. Furthermore, unregistered SIP Servlets Engine Providers (SSEP) can also be present in the MANET (i.e. an SSE with expired TTL that has not re-registered, any inactive SSE). These SSEs publish their function, capacity and TTL in order to be discovered by the SGW when needed. The SSEP can provide either a centralized SSE or a distributed SSE. With a distributed SSE, the SSEP is responsible for the managing the SSE, as described in the previous chapter.

Two other possibilities can be envisaged for the communication between SSEs and SGWs. One is to remove the registration step. Indeed, the SGW will discover the SSE at the appropriate time. The other is to only use the registration framework. We believe that the approach we chose (i.e. a combination of these two approaches) combines their respective advantages: fault tolerance, time saving and controlled management.

6.4.4.2 At the service execution runtime

When an end-user invokes a service via the service gateway, the S-CSCF forwards it to the appropriate application server. The AS evaluates certain criteria and then dynamically decides where to run the service (i.e. in the MANET or in the 3G portion). We fix the criteria as follows: the users' location, the preferences of the service provider, the network load and the reliability of the link between the 3G and the MANET sub-networks.

The AS retrieves the end-user's location (i.e. 3G or MANET) from the HSS through an extended *Sh* interface as per our assumption. This information is relevant to the current alternative where all end-users are in the MANET portion.

The service provider preferences are defined in the AS while the network load can be obtained using context information that has been included in extensions to the IMS architecture [111]. The context information is stored in a Context Information Base (CIB) and collected from context sources. Any IMS entity can subscribe or request the context information. The architecture offers the current network load and the current network status (i.e. regular or crisis) as context information. The AS in our architecture can subscribe to the network status information (the SIP event notification framework is used here) and uses it for load balancing decisions.

Basically, the decision to run a service in the MANET addresses three main objectives: speeding service execution, improving load balancing and 3G/MANET link reliability. The first criterion depends on the Service Provider (SP) preferences. The SP chooses a service, or all services, or a category of services to run in the MANET. The load balancing criterion is variable and depends on the network traffic. Finally, the last criterion is pre-defined. Indeed, the reliability of the link between 3G and MANET is an MCN property.

The *decision making* process starts when all the end-users involved in the service execution are in the MANET. Figure 6.9 shows the *decision making algorithm* at the AS level.

```
- SvcA: the service to run
- NetLoad: the network load
- 3G_MANET_Link: the link property between the 3G and MANET.
- SP_pref(X): the preference of the Service provider regarding execution of the service X
If all users are in MANET
then If SP_pref(SvcA) <> "in MANET"
    then If NetLoad == "Empty"
            then NetLoad = request load information from the CIB.
    If NetLoad == "regular"
        then If 3G_MANET_Link == "reliable"
                then run the service in 3G
             Else run the service in MANET
    Else run the service in MANET
    Else run the service in MANET
```

Figure 6.9. The AS decision making algorithm

The algorithm can be described as follows: The SP preferences regarding a *service A* are checked first. If the SP has no preference regarding running the service in MANET then the AS looks at the network load obtained from the CIB. If the current network load is regular, for example (i.e. load balancing is not required), then the link property between 3G and MANET is checked. If this is deemed weak, then the AS will decide to run the service in MANET; otherwise the service is executed in the 3G sub-network.

When the service is executed locally in the 3G, the execution process is handled as usual, with the current 3G settings. However, when the AS decides to run the service in the MANET portion, it adds the SIP servlets' location address to the SIP request and forwards it to the SGW, which then selects an SSEP from the list of registered servlets engine providers, providing an SSE with the longer TTL and greater capacity. If the list is empty (i.e. no SSEP has registered or all the SSE's TTLs have expired), the SGW discovers available inactive SSEPs in the MANET. The SGW then sends the SIP request to the selected SSEP. The SSE then runs the service as described in the previous chapter.

6.4.4.3 At any given time

In order to keep the service execution environment in MANET transparent to 3G users, the service gateway manages the SSEPs. It receives PDP update messages pushed by the SSE. In fact, when a given parameter is updated at the SSE level (e.g. an extended TTL, a critical level of battery power, a change in capacity), the SSE informs the SGW. Furthermore, the service gateway maintains the list of registered SSEPs and ensures that they are available. Any heartbeat message can be used. However, the message frequency should be as low as possible since this is only a preventive measure.

When the SIP servlets engine provider wants to leave the MANET, its SSE pushes a PDP message with a TTL value equal to zero. Then the SSEP providing this SSE is removed from the SGW list and considered to be unavailable. If an SSE fails abruptly, the MANET self-recovery procedure takes place, as depicted in the previous chapter. The SGW is then informed about the new SSE address.

6.4.5 Illustrative scenario

The interest-based conferencing service is chosen to illustrate the SIP servlets-based service provisioning process in the MCN. This service establishes a conference with participants that share the same interests. The service is implemented using the SIP servlets framework. Participants register their interests with the service provider. The registration specifies the minimum number of participants required to start the conference. The service provider manages the fields of interest and requests the initiation of a conference between participants that share the same theme. The SIP servlets location is transmitted within the invitation message to the SGW, which forwards it to the registered SIP servlets engine. The SSE downloads the required SIP servlets and runs the logic that establishes the conference. New parties can join later when invited by the service provider. The conference participants can leave at any given time.

Figure 6.10 shows the message flow between the different entities in the MCN when the service runs in the MANET, based on the application server's decision. We assume that the publication/discovery and registration processes have finished and that the conference is fully meshed. Media handling issues in MCN are beyond the scope of this work.

Following the registration phase, the extended AS (E-AS) finds that EUA1 and EUA2 have matching interests, gets their locations from the extended HSS (E-HSS) and decides to run the service in the MANET. The E-AS uses an SIP extension (i.e. DIAMETER) to communicate with the E-HSS. In this example, the decision is based on the fact that both users are in the MANET, and on the service provider's preference. The E-AS sends the conference creation request with the address of the SIP servlets to the SGW via the S-CSCF. This service gateway transmits the request, together with the network address of the participants, to the extended SSE (E-SSE) that downloads the servlets and runs the service.

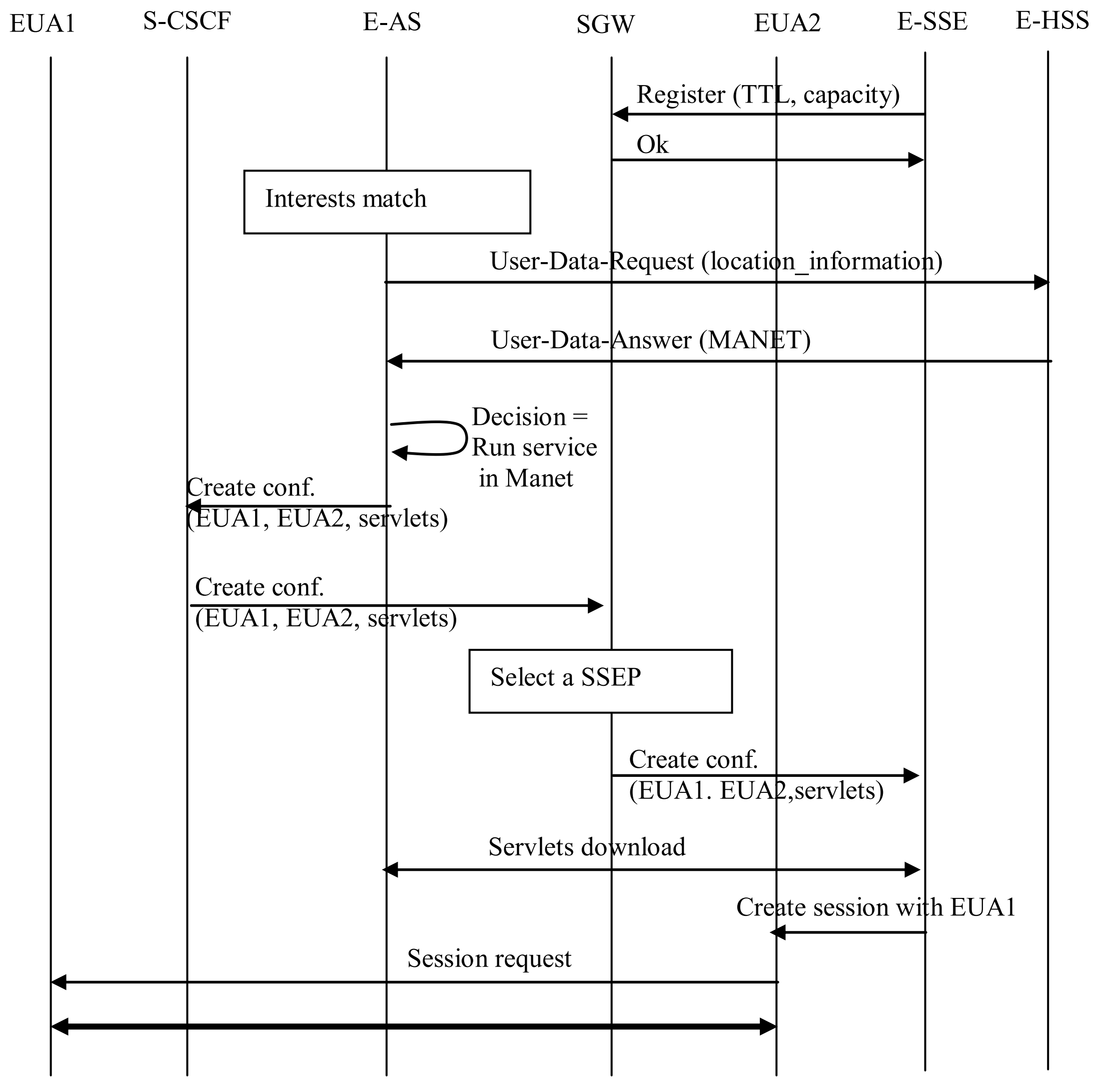

EUA1
S-CSCF
E-AS
SGW
EUA2
E-SSE
E-HSS
Register (TTL, capacity)
Ok
Interests match
User-Data-Request (location_information)
User-Data-Answer (MANET)
Decision = Run service in Manet
Create conf. (EUA1, EUA2, servlets)
Create conf. (EUA1, EUA2, servlets)
Select a SSEP
Create conf. (EUA1. EUA2,servlets)
Servlets download
Create session with EUA1
Session request

Figure 6.10. Conference establishment between two MANET users in MCN

Since the SSE is in the MANET, as are all the users, it can reach the MANET participants directly and thus save time and bandwidth.

We assume in this scenario that the SIP servlets are hosted in the E-AS, but they may be located anywhere. Furthermore, in this example the required minimum number of participants sharing the same interests to start a conference is two.

6.5 Summary

In this chapter we have presented and motivated the service provisioning issue in integrated 3G/MANET networks. Multihop Cellular Networks (MCNs) were considered as an example of such integration.

We have described and discussed an exhaustive set of high-level architectural alternatives for providing services in MCNs. The alternatives were grouped into two categories: service execution in MANET and service execution in 3G. The advantages of each alternative were elaborated.

Furthermore, the chapter presented a concrete and detailed architecture for service provisioning in MCNs. The architecture is tailored to the alternative where all end-users are in the MANET and the services are provided by 3G service providers and executed in the MANET. This alternative is the one most interesting from the network operator viewpoint. The assumptions, principles, functional entities and procedures have been discussed.

The proposed architecture allows the service to be executed either in the MANET or in the 3G network. The SIP servlets engine can be provided by individuals, since any end-user in MANET with this functionality can register with the SGW. With the proposed architecture, it is also possible that many SSEs coexist in the MANET, managed by the SGW. Furthermore, the architecture allows either centralized or distributed SSE, since the SIP servlets provider can provide a centralized or a distributed engine. The SGW makes the service execution environment transparent to 3G and therefore to the AS. Furthermore, the extensions are minimal and not difficult to achieve.

CHAPTER 7 : Validation for the case of stand-alone MANETs

This chapter presents a validation stack for the architectures and solutions proposed earlier in this thesis for stand-alone MANETs. Both a proof of concept prototype and a formal validation are discussed. The prototype was implemented to demonstrate the feasibility of the business model solution. The overlay network protocol was formally verified. The organization of this chapter is as follows: first it describes and discusses the business model prototype, and then it analyses the overlay network protocol validation using SPIN.

7.1 Business model proof of concept

This section discusses the prototype implemented as a proof of concept for the business model and the related publication and discovery mechanism. The results these scenarios are then analysed. The main goal of this prototype is to show that the novel business model proposed in chapter 4 is practical and feasible in a stand-alone mobile ad-hoc network.

Four roles have been implemented, those of end-user, service provider, capabilities provider and service execution environment provider. Each role publishes and/or discovers the features provided by the other roles. Both push and pull scenarios were implemented. Table 7.1 show the roles and their possible interactions during the service publication and discovery process.

Roles	Publishes	Discovers	Pushes
End-user	-	services	-
Service provider	Services	Capabilities	services
Capabilities provider	Capabilities	Execution environment	Capabilities
Service execution environment provider	Execution environment	-	Execution environment

Table 7.1: Business model roles' interactions during the publication/discovery process

The following prototype subsection focuses on the publication and discovery process.

7.1.1 Prototype

The Pervasive Discovery Protocol (PDP) was chosen, as discussed earlier, as the service publication and discovery protocol. Some extensions were added to the PDP to allow the push scenario and to permit the publication and discovery of different features (i.e. services, capabilities, execution environments). The features' descriptions are stored in XML files. The information related to service invocation and execution is not entered into the description file. We focus on the required resources in term of capabilities and the execution environment. In fact, a service is published with its required service capabilities. Furthermore, the capabilities are published together with the required execution environment. The execution environment can be published with no extra features required. Figure 7.1 shows the XML description with the focus on the relevant data for the service publication and discovery processes. The figure presents a service description. The capabilities and execution environment description are similar to those for service execution.

```
<Service_feature Type='Service'  name='  ' version='  '  URI = '  ' >
   <Parameters name=' '  value = ' '>
      <Variable> </Variable>
   </Parameters>
   <Port name=''>
      <Operation name=' '>
           <Args type=' '>  </Args>
     </Operation>
      <In > OperatioInName </In>
     <Out> OperationOutName  </Out>
   </Port>
   <Binding PortRef=' '>
           <Bind >IP_Port </Bind>
           <BindProtocol> </BindProtocol>
   </Binding>
   <Sessions URI=' ' Members=' '  Max_Members='' />
   <LogicReq>
     <Resource>
           <OS>
                 <Name> </Name>
                 <Version> </Version>
           </OS>
           <Memory> </Memory>
           <Processing> </Processing>
           <GraphicReq> </GraphicReq>
           <Capabilities>
                       <Name> </Name>
                       <Exec.Env.> </Exec.Env>
            </Capabilitities>
     </Resource>
   </LogicReq>
</Service_feature>
```

Figure 7.1: XML service description

The *type* tag specifies the nature of the service feature being described (i.e. a service, a capability or an execution environment). The *Resource* tag may contain one or multiple *Capabilities* tags. However, when the describe feature is a capability, this tag becomes an *Exec.Env* tag which specifies the required execution environment for that capability.

7.1.1.1 PDP and extensions

The Pervasive Discovery Protocol (PDP) has been selected for the feature publication and discovery protocols. The PDP does away with the need for any central entity. Furthermore, one of the main objectives of the PDP is to reduce traffic in the network by minimizing transmissions. Consequently, the PDP saves network bandwidth and devices' resources, particularly for those with very limited resources. Indeed, the PDP prioritizes the most powerful devices to reply to the requests, allowing the others to abort their replies. These are central properties for a protocol's efficiency in ad-hoc networks.

As mentioned earlier in this thesis, the PDP has two mandatory messages *PDP_Service_Request* and *PDP_Service_Reply,* to request services and reply/publish services, respectively. An optional message, *PDP_Service_Deregister,* is sent to inform about a service withdrawal.

Each device is assigned an availability time, a local and a remote memory cache. The availability time represents the excepted time a device will remain in the network. The local memory cache stores the services the entity is willing to share. The remote memory cache stores the discovered services. The PDP makes use of two agents: to discover available services in the network a device uses the *PDP User Agent,* and to publish services a device uses the *PDP Service Agent*.

We have added some extensions to the protocol so that it can better fit our architecture. First, we add a new field in the PDP messages to distinguish between services, capabilities and execution environments. The field *Category* is inserted into request (i.e. reply and deregister messages) which gives PDP the possibility to publish and discover different features. Second, in order to enable the push scenario we modify the *PDP_Service_Reply* message header by adding a new field *flag*. When set this field informs the receiver that the protocol is operating in the push mode. Furthermore, a message notification for pushed services was implemented -- after notification, the services are added to the remote memory cache.

The PDP protocol was also augmented with a lightweight XML parser (e.g. NanoXML) to handle the features' descriptions. Some extra functions were implemented that can remove services from the local memory cache, deregister an individual service that was removed and automatically refresh the content of the remote memory cache.

7.1.1.2 Prototype architecture and environment

a. Prototype architecture

The prototype is made of four main modules. The same modules run in the end-user, the service provider, the capabilities provider and the execution environment provider devices. Figure 7.2 shows the prototype architecture.

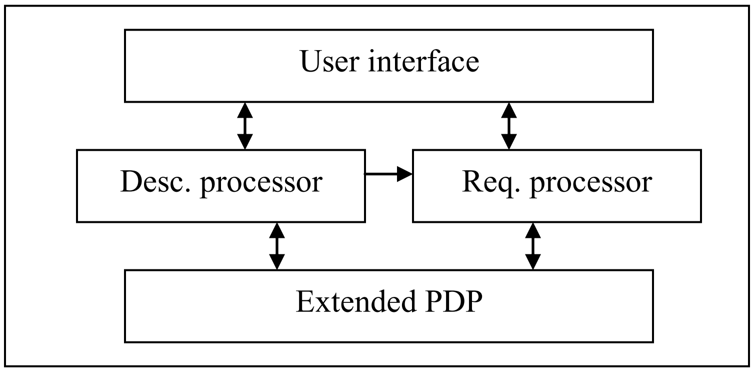


Figure 7.2: Business model prototype architecture

The User interface module offers users (i.e. end-users and providers) the possibility to select the features they wish to offer or to discover. The providers can choose, via the user interface, to push the selected features to the network using a multicast address. Furthermore, this module allows the lifetime for each feature to be set and to display the discovered features along with their lifetime (i.e. Time To Live).

The description processor module (Desc. Processor) manages the features' descriptions and processes the XML files. The description is transformed into a PDP message. The Extended PDP is the protocol for publication and discovery with the extensions mentioned in the previous sub-section. The Extended PDP is also responsible for network communication. Finally, the request processor (iReq. processor) module processes the incoming requests (i.e. features push or discovery requests). It checks the messages and decides to reply immediately, do nothing or initiate another request before replying. The module replies immediately if the requested feature is available and does not require additional features (i.e. capabilities or execution environments). It decides to do nothing if the requested feature is not available or if other nodes have already replied. Finally, the module may initiate a series of requests to discover the features that correspond to the specific requested feature. For example, if a service is requested the module will request its corresponding capabilities before replying with that service.

b. Prototype environment

Three laptops with IEEE 802.11g adaptive cards were used to create an ad-hoc network. These machines are Pentium 4s or mobile Pentium 4s models with 512 MB RAMs running Windows XP Professional. Java was chosen as the programming language. We also used NanoXML (version 2.2.3) as a lightweight XML parser. The parser is employed to map a service to its required capabilities and the capabilities to the required execution environment.

Since any functional entity can be supplied by any business role at any time, all of the machines can play any business model role.

However, we dedicate a laptop to the service provider agent role, since it has direct communication with all the other roles. The remaining two machines host a combination of the capabilities provider agent, the execution environment provider agent and the end-user agent, depending on the scenario. Several scenarios were implemented.

7.1.2 Results

In this sub-section the implemented scenarios are described and then results are analyzed.

7.1.2.1 Scenarios

Figure 7.3 presents the general pull scenario.

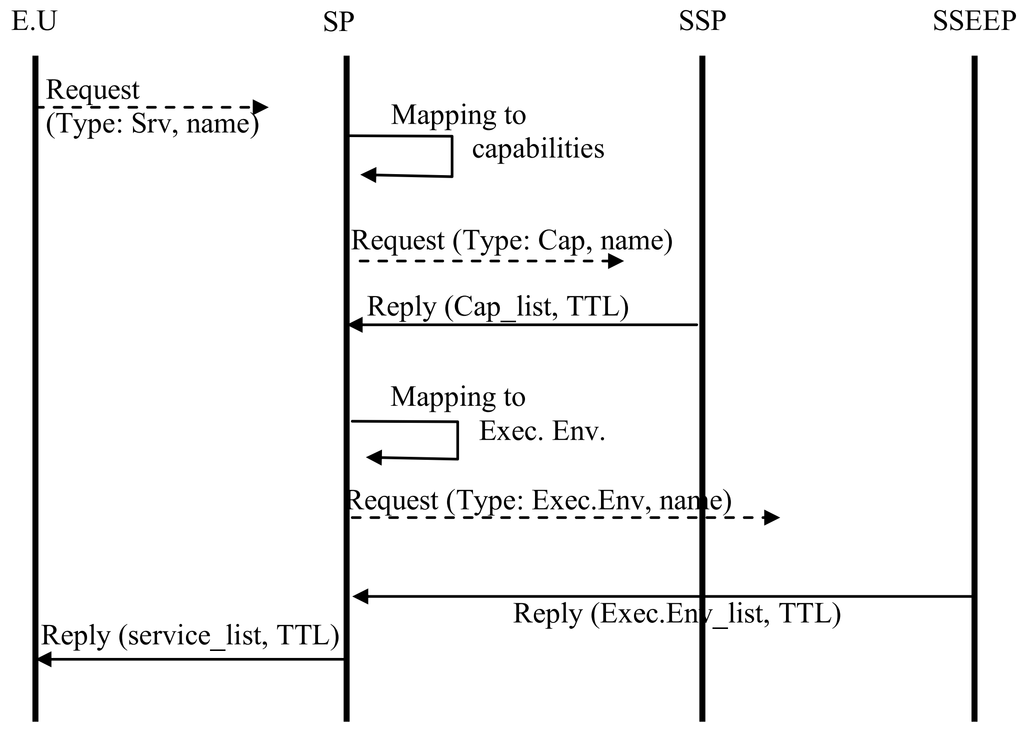

E.U
SP
SSP
SSEEP
Request (Type: Srv, name)
Mapping to capabilities
Request (Type: Cap, name)
Reply (Cap_list, TTL)
Mapping to Exec. Env.
Request (Type: Exec.Env, name)
Reply (Exec.Env_list, TTL)
Reply (service_list, TTL)

Figure 7.3: The general pull scenario using PDP

When the service provider (SP) receives a request for a service, it gets the list of services from its memory cache. For each service, it gets the corresponding capabilities and execution environment from a local XML file. Then, the service provider sends a request for each capability and its corresponding execution environment to the SIP servlets provider (SSP) and the SIP servlets execution environment provider (SSEEP), respectively. If for a given service, all the capabilities and execution environments are available, the service provider returns this service to the end user.

However, different scenarios can be derived from the one described above. Any combination of push and pull constitutes a hybrid scenario. We have implemented seven scenarios in a pull fashion and three in a hybrid fashion. All of the pull scenarios are based on Figure 7.3. The difference between the seven scenarios is mainly in the number of capabilities and the number of execution environments required for a service. Table 7.2 describes the service requirements used in our scenarios.

Services	**Required features**				
A	Required capabilities	*Cap1*			
	Required execution environment	*Exec1*			
B	Required capabilities	*Cap1*	*Cap2*		
	Required execution environment	*Exec1*	*Exec1*		
C	Required capabilities	*Cap1*	*Cap2*	*Cap3*	
	Required execution environment	*Exec1*	*Exec1*	*Exec1*	
D	Required capabilities	*Cap1*	*Cap2*	*Cap3*	*Cap4*
	Required execution environment	*Exec1*	*Exec1*	*Exec1*	*Exec1*
E	Required capabilities	*Cap1*	*Cap2*		
	Required execution environment	*Exec1*	***Exec2***		
F	Required capabilities	*Cap1*	*Cap2*	*Cap3*	
	Required execution environment	*Exec1*	***Exec2***	***Exec3***	

Table 7.2: Required features for the scenarios' services

We should mention that we use empty services, since we focus on the publication and discovery process. Six services are used. Each has different requirements in terms of capabilities and execution environment. The services range from simple ones that require one capability and one execution environment to services requiring several capabilities and multiple execution environments. However, the most common scenario requires multiple capabilities and the same execution environment.

The pull scenarios are described in table 7.3.

Scenarios	Description
Scenario 1	The service provider discovers capabilities and execution environment
Scenario 2	End-user discovers **service A**
Scenario 3	End-user discovers **service B**
Scenario 4	End-user discovers **service C**
Scenario 5	End-user discovers **service D**
Scenario 6	End-user discovers **service E**
Scenario 7	End-user discovers **service F**

Table 7.3: The pull scenarios description

The first scenario is set at the service provider level. It gives an idea about capabilities and the corresponding execution environment discovery process. The remaining scenarios are at the end-user level. In these scenarios, the services described in table 7.2 are discovered according to the pull mode.

We also define hybrid scenarios. We chose three of the pull scenarios and ran them in a hybrid manner. Indeed, instead of discovering all the capabilities and all the execution environments required, we push some of these features in the network. Table 7.4 describes the hybrid scenarios.

Scenarios	Description
Scenario 1	*Cap1* and *Cap2* are pushed, then end-user discovers **service D**
Scenario 2	*Exec1*, *Exec2* and *Exec3* are pushed, then end-user discovers **service F**
Scenario 3	Same as scenario 2 but *Cap2* is also pushed

Table 7.4: The hybrid scenarios description

Pull scenarios that require several capabilities and execution environments are chosen for the hybrid scenario. The goal is to measure the impact of the push mode on the discovery process. In hybrid scenario 1, two of four capabilities are pushed while the execution environment is discovered in a pull mode. In hybrid scenario 2, the execution environments are pushed while the capabilities are discovered. Finally, in hybrid scenario 3 both capability and execution environment are pushed.

The services **E** and **F** are a variant of services **B** and **C,** respectively. In the former, each capability needs a different execution environment. Hence, scenarios 6 and 7 are an extension of scenarios 3 and 4. Similarly, the hybrid scenarios 1 and 2 are related to the pull scenarios 5 and 7, respectively, where some features are pushed and others are discovered.

7.1.2.2 Results and analysis

Each scenario was executed several times and an average response time was calculated. We ran each scenario five times as a trade-off between achieving realistic results and the time constraints. A comparison between the pull and the hybrid scenarios are presented. Table 7.5 presents the average response time of the pull scenarios.

Pull scenarios	**Average response time (sec)**	**Standard deviation**
Scenario 1	0.178	0.07
Scenario 2	0.554	0.07
Scenario 3	0.732	0.08
Scenario 4	0.904	0.02
Scenario 5	1.044	0.04
Scenario 6	0.898	0.03
Scenario 7	1.258	0.04

Table 7.5: The average response time for the pull scenarios

It is clearly shown that the response time increases with the number of capabilities to be checked and discovered. The response time also increases when the capabilities require different execution environments. For example, the response times are higher in scenarios 6 and 7 than in their comparable scenarios, 3 and 4. In fact, scenarios 3 and 4 are the same as scenarios 6 and 7 but with different execution environments to discover. As we can see, the response time exceeds 1 second for services that require 4 different types of capabilities (e.g. service **D** in scenario 5) and for services that require 3 different types of execution environments (e.g. service **F** in scenario 7). We could consider these service features' requirements as a threshold for better service provisioning performance. However, meticulous performance evaluation should be elaborated before any conclusion.

For all the validation runs, the standard deviation remained relatively low, showing that the results are coherent. In fact, during several runs the response time remained stable.

Table 7.6 presents the results of the average response time of the hybrid scenarios.

Hybrid scenarios	**Average response time (sec)**	**Standard deviation**
Scenario 1	0.78	0.19
Scenario 2	0.77	0.06
Scenario 3	0.5	0.01

Table 7.6: The average response time for the hybrid scenarios

The hybrid scenarios showed better response times compared to the pull scenarios. Some features are pushed, and then the service provider does not need to discover them. In scenario 1, two of the four capabilities required by service D are pushed. In scenario 2 the three execution environments required for service F are pushed, while in scenario 3, in addition to the execution environments, one capability is also pushed, which explains the lower response time. The standard deviation shows that the intermediate results are not very different from the average response time.

To compare the hybrid scenario and the pull scenario, hybrid scenario 1 needs to be compared to pull scenario 5, and hybrid scenarios 2 and 3 compared with pull scenario 7.

Figure 7.4 illustrates the differences between the hybrid and the pull scenarios. The response times from pull scenario 5 and hybrid scenario 1 are plotted. We can see clearly that the hybrid scenario can improve the response time during service discovery in our model.

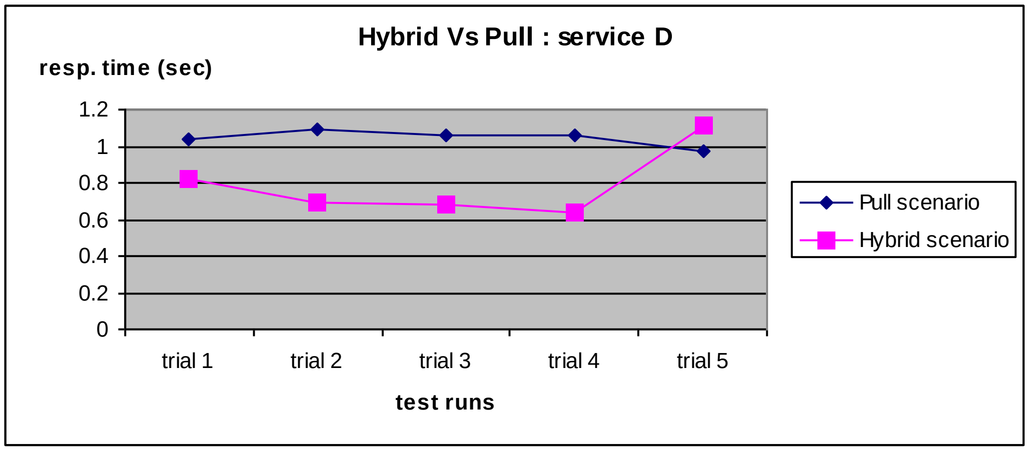


Figure 7.4: Comparison of the pull and hybrid scenario for the service D discovery

At trial 5 the hybrid scenario response time is higher than in the pull scenario. However, since the trials are very similar, this may due to implementation, network or device load issues.

Figure 7.5 shows the difference between the hybrid and pull scenarios for the F service discovery. The response times from pull scenario 7 and hybrid scenarios 2 and 3 are plotted. In hybrid scenario 3 more of the features required by service **F** are pushed than in the hybrid scenario 2.

The graph in Figure 7.5 illustrates again that with an adequate mechanism for pushing service features, the providers can significantly improve the response time perceived by the end-user.

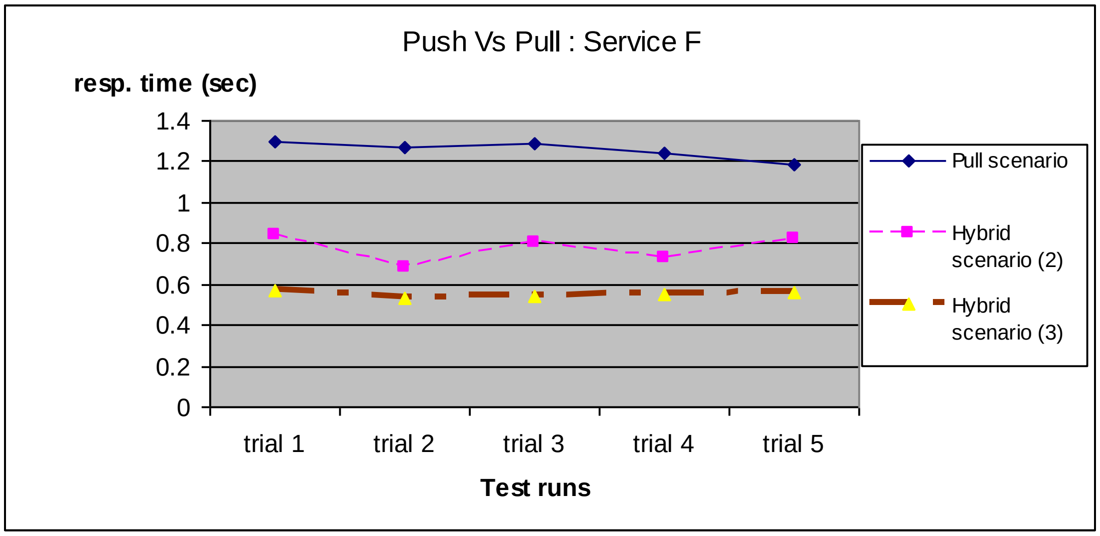


Figure 7.5: Comparison of the pull and hybrid scenario for discovery of service F

7.2 Overlay network validation

This section presents the formal validation of the overlay network protocol for self-organization and recovery. The validation tool is introduced first, followed by the modeling details. Next, the section analyzes the different validation parameters and then concludes.

7.2.1 The validation tool

The validation was performed using PROMELA and SPIN [111]. PROMELA (PROtocol/PROcess MEta LAnguage) is a high-level specification language that allows the dynamic creation of concurrent processes. These processes communicate via message channels. PROMELA is employed to model finite state-distributed systems. PROMELA programs are called validation models. They focus on process interaction and abstract unrelated protocol or distributed system details.

SPIN (Simple PROMELA INterpreter) takes PROMELA's validation models as input and simulates the interactions between the processes. SPIN also performs a formal validation by checking the correctness of the model. This is done using *assertions* within the PROMELA program, or by expressing correctness properties with the Linear Temporal Logic formula at runtime. The correctness criteria are checked in PROMELA by expressing them as invalid behaviours or properties in a particular state. Therefore, SPIN acts as a simulator and as a validator.

Furthermore, SPIN can perform this validation using the exhaustive search method, which is the best option since all the paths of the validation model of the system to be checked are explored. If the exhaustive search method does not report a given violation, then there cannot be an execution sequence with that violation. Systems of less than 100 000 states can use this method with limited impact on device performance.

SPIN can also perform the validation using a partial search method based on the *bitstate* algorithm. Indeed, for large to very large systems the exhaustive search method is not feasible due to memory and time constraints. The partial search or *bitstate* method solves this problem by using a hash function that stores a state using one bit. It is a partial search because the algorithm counts all the newly inserted states that have the same hash value as having been visited.

7.2.2 The modeling process

In PROMELA, both processes and communication channels are modeled. These are discussed in the following subsections.

7.2.2.1 Validation processes

All of the processes of the overlay network were modeled in PROMELA. We modeled the *controller*, the *wrapper*, the *connector* and the *session repository* entities. The different validation models were sketched according to the final state machine models discussed in chapter 5. Furthermore, a main process was created that initiates the overlay network protocol. The main process specifies, for each process type, the number of instances that the simulation/validation will contain.

A simulation is successful when the appropriate links are established between the processes. The processes' relationships should reflect the overlay network organization described in chapter 5.

Each node is defined by a unique *Id,* a *type* and a *status*. The *Id* is used to send/receive messages. The *type* defines the entities' type (i.e. controller, connector, session repository, or wrapper) and the *status* informs if a node is free or busy (i.e. already involved in service provisioning). Furthermore, each *controller* is assigned a data table that contains information about the nodes it manages. We also assign a data table to the *session repositories* in order to store the session information and the list of the existing *session repositories.*

In addition, we use certain probability functions to initiate the node departure. Nodes can leave by sending a BYE message to the appropriate node (e.g. the controller). However, this assumes that there is a *controller* in the network at that time. To fulfill this constraint, we add the following condition:

```
if
:: (defaultCtrId!=0) -> toCtr[defaultCtrId] ! Bye();
        goto wait_ok_to_Bye;
:: else ->  goto end;
fi;
```

This condition checks if the default *controller* of the leaving node is in the network. If yes, it sends a BYE message to that *controller;* if no, then it simply leaves.

7.2.2.2 Communication channels

The different processes require a mechanism for communication. PROMELA provides point-to-point channels. We defined several channels for message exchanges. We assume that the publication/discovery is done by using a global repository.

We defined four main channels: *ToCtr*, *ToSR*, *ToNodes* and *varExchange*. *ToCtr* is used by any node to send messages to the controllers. The *ToSR* channel is used by session repositories and controllers to send messages to session repositories. *ToNodes* is used by controllers to send messages to any node, and *varExchange* is used by the connector to contact the end-user. This is necessary because the end-user must be informed of an alternative access point prior to connector departure.

During the protocol processing, the controllers and the session repositories need to multicast their messages to the other controllers and session repositories, respectively. However, PROMELA does not allow point-to-multipoint communication. To solve this problem, the above channels are treated as variables declared as an array of channels. For instance, *ToCtr* is declared as follows:

```
chan ToCtr[max_Controllers] = [QSZ] of {byte,byte,byte}
```

[QSZ] refers to the maximum size of each channel. Each message has three parameters: the request type (e.g. Refers), the node's *Id* and the node's type. Furthermore, each channel is indexed by the *Id* of the concerned node. For example, the messages addressed to the controller with *Id* equal to 1 are written/read to/from ToCtr[1].

7.2.3 The correctness requirements

PROMELA only provides a global timeout, which fires when there is no executable process in the system (i.e. a deadlock). The PROMELA timeout is used to escape from deadlock states or to recover from message loss. SPIN allows checking for several properties: deadlocks (i.e. invalid end-states), livelocks (i.e. cyclic executions) and improper terminations (i.e. execution completion with a violation of the termination conditions).

Consequently, our first correctness requirement is to ensure that our overlay protocol is free of deadlocks and livelocks. To this end, we use the *end* and *accept* predefined labels of PROMELA. For each entity we identify the valid end-states and the operations that should not be repeated indefinitely, prefixing them with the labels *end* and *accept,* respectively.

A second requirement is to verify the different data tables' consistency. We defined global variables for the *controller* and the *session repositories* entities. SPIN allows the data to be traced and ensures that the values reflect the protocol's progress. Furthermore, assertions are employed to express conditions related to certain data values.

The third correctness requirement that we checked is the association between the overlay network organization and the protocol result. Indeed, each *controller* should manage the nodes it is supposed to manage, and there should not be any free node in the network. This is checked by looking at the *controller*'s and the node's tables.

The messages sequence was also verified. We guarantee that no unexpected messages are received by a given entity and that the correct reply is sent upon reception of a valid request. PROMELA's temporal claims, which are prefixed by the keyword *never*, were used to achieve this goal. Figure 7.6 shows an example of a temporal claim specifying that a session repository entity can never receive a *Request_node()* message. Furthermore, if the session repository entity receives that message, it should ignore it and not reply.

```
never {
do
        ::!ToSR[1]?[Request_Node]
        :: ToSR[1]?[Request_Node]-> goto accept

od;

accept:
do
        ::!ToCtr?[Node_Reply]
od;
}
```

Figure 7.6: An example of a temporal claim

In addition, we also verify via SPIN/PROMELA that the process ends correctly, meaning that nodes departures are well handled well. The data tables are checked to verify that the updates are made appropriately, and that the new links are created properly.

Furthermore, the SPIN simulator has a graphical output that shows the protocol progress. This helps to detect any error in the message order or in the protocol behaviour. Figure

7.7 illustrates SPIN's output during protocol simulation. The messages and the process are identified by numeric Ids.

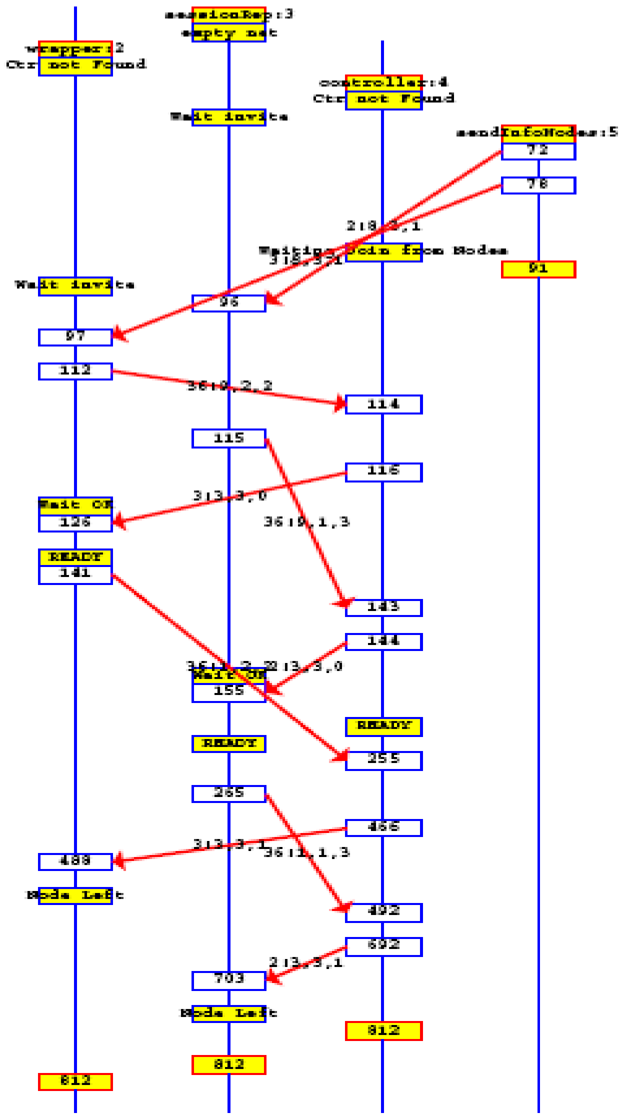


Figure 7.7: Example of SPIN's output during the overlay network simulation

7.2.4 Conclusion

We simulated and validated the overlay network protocol on a Pentium 4.3 GHz desktop with 512 MB of RAM and running Windows XP. The simulation was performed using SPIN 4.2.7 and XSPIN 4.2.7. The simulation environment was set up for a 1 000 000 depth search boundary and a memory limit of 512 MB. We used the partial (i.e. supertrace or bitstate) method to search, which offers good coverage in SPIN. Saving on the memory requirements motivated this choice of search method.

Several scenarios were simulated and the correctness requirements were checked for each scenario. The scenarios reflect the different situations that may occur. Three families of scenarios were defined: the first is the node of type^{+*} (i.e. a connector, a wrapper, or a session repository) joins. The second family is for a controller join situation, and the third family is for overlay nodes' leaving.

For each family we define two or more sub-scenarios. The first family contains the following two scenarios: nodes of type^{+*} join and find a controller in the network, and nodes of type^{+*} join and do not find a controller. The second family contains the following two scenarios: the first controller in the network joins, and a controller joins and finds existing controllers The third family includes three scenarios: departure of nodes of type^{+*}, a controller leaving, and the last controller in the network leaves. Furthermore, we use several instances of each entity for each scenario. The number of instances is from one to five for each entity.

For each simulation, the data tables and assertions, the temporal claims and the message flow are checked. The data generated after the simulations were coherent with the executed scenario. The validation process established that the protocol was free of deadlocks and that the assertions and temporal claims were not violated, thereby proving the validity of this protocol.

7.3 Summary

In this chapter we presented the proof of concepts related to our proposed architecture from the pure MANET point of view. The business model prototype and its results were discussed. We then elaborated on the overlay network protocol validation. The protocol was simulated using SPIN.

The business model prototype demonstrated two main ideas. First, that the proposed business model architecture is feasible and that the different roles can collaborate for service provisioning. Furthermore, using the appropriate mechanism for publication and discovery and inter-role communication makes the solution realistic for MANETs. Second, the prototype's results revealed that the performance is improved with a hybrid scenario of *push* and *pull* modes.

Regarding the overlay network protocol, the formal validation demonstrated that the protocol is free of deadlocks, livelocks and unreached states. It also showed, through the correctness criteria, that the protocol is correct and that it constructs the desired overlay network structure.

CHAPTER 8 : Validation for the case of integrated 3G/MANETs

This chapter presents a proof of concept for the integrated 3G/MANET architecture for service provisioning. It also discusses the performance evaluation of this architecture. The organization of this chapter is as follows: first it describes and discusses the implemented prototype as a proof of concept. Then, it presents and analyses the simulation results as a performance evaluation. The performance analysis is done using the OPtimized Network Engineering Tool (OPNET). Furthermore, it is for the precise case of the *Interest-based conferencing* service.

8.1 Integrated 3G/MANET prototype

8.1.1 Assumptions and mechanisms

Signalling issues in Multi-hop Cellular Networks (MCNs) have been solved [40] and are out of the scope of our work. Furthermore, the prototype is based on three assumptions: that one instance of the Service Gateway (SGW) is used in the prototype; that the S-CSCF is pre-configured to know that SGW; and that the SIP servlets are hosted in the Application Server (AS). In addition, the implementation makes use of different existing technologies. For gateway discovery -- in the MANET portion, the Pervasive Discovery Protocol (PDP) is employed. Furthermore, HTTP is used to download the SIP servlets from their location. Indeed, the SIP Servlets Engine (SSE) implements the HTTP client side while the AS implements the server side. Furthermore, a MANET end-user agent with SIP capabilities was implemented as a UE.

8.1.2 Prototype environment

Figure 8.1 shows the prototype settings. The S-CSCF is collocated with the AS.

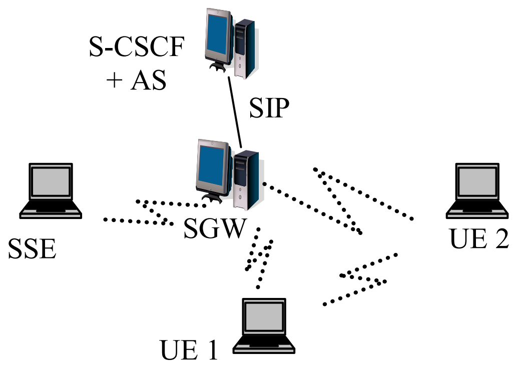


Figure 8.1: The integrated 3G/MANET prototype settings

JAVA was our programming language. Three laptops make up our MANET: two host the user equipment and the other hosts the SIP Servlets Engine Provider (SSEP). Two desktops form our 3G network. One machine hosts the SGW and the other one hosts the AS and the S-CSCF.

The laptops are Windows XP with 802.11g adapting cards configured in the ad-hoc mode and using the EODV routing protocol. The machines are mobile Pentium 4's with 512 MB of RAM. Furthermore, the desktops are Windows XP with a 3 GHz Pentium 4 and 1Gig of RAM. The Service Gateway (SGW) has a dual interface: an ad-hoc interface for MANET communication and an infrastructure link to the wired network.

Since we cannot have multiple end-users in our environment, the scenario chosen is the establishment of a conference between two end-users based on their interests. The minimum number of users required is obviously fixed to two end-users. The conference service is hosted in the 3G while the execution is done by the SSE hosted in the MANET.

8.1.3 The scenario description

The *interest-based conferencing* service introduced in chapter 2 is implemented for the proof of concept. Figure 8.2 shows the SIP implementation of the conference establishment. For clarity, only the main SIP messages are shown and the discovery process is skipped.

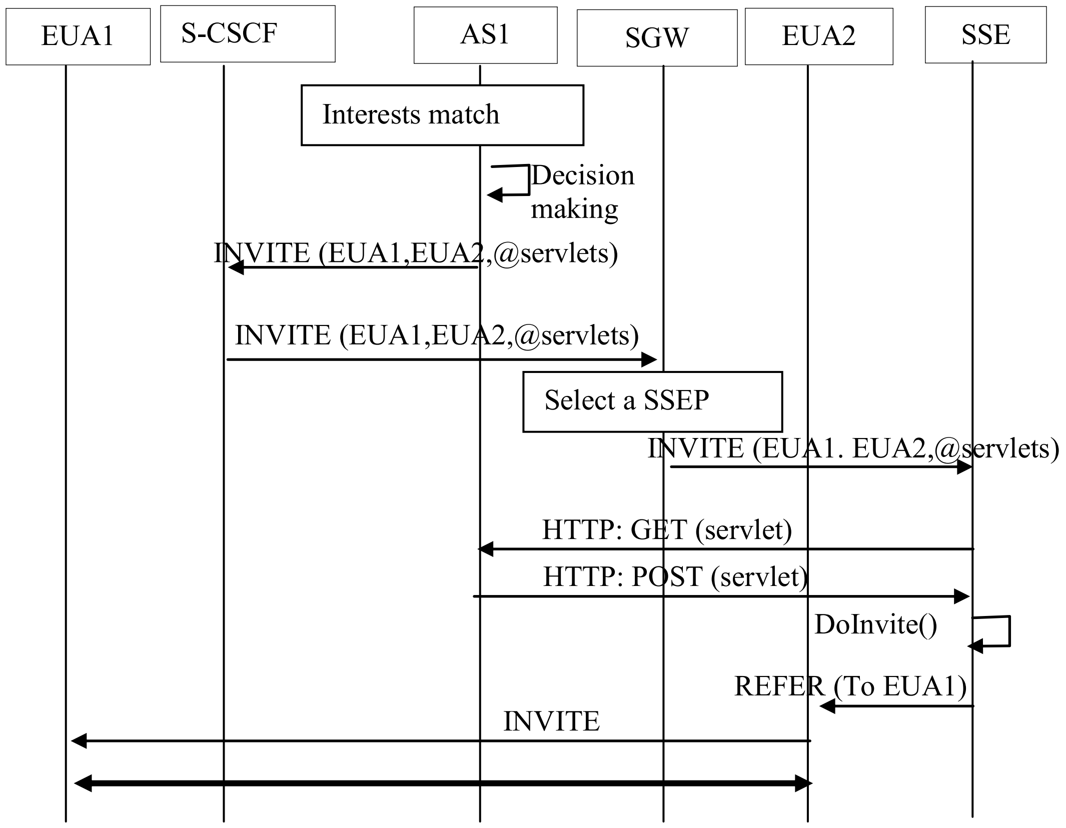


Figure 8.2: Interest-based conference establishment in MCNs

A simple prototype has been built as a proof-of-concept. We implemented a simplified S-CSCF and AS, a simple UE, the SGW and the extended SSE functional entities. In fact, it is impossible for us to create a complete IMS infrastructure in our lab. Therefore, we used a dummy S-CSCF that only forwards the service requests to the AS and exchanges SIP messages with a pre-configured SGW. The HSS was not implemented but the end-

user location was stored in a plain text file and checked by the AS. The interest-based service was implemented as per the AS. An HTTP server was also added to the AS. However, the Context Information Base (CIB) and context information parts were not considered. Instead, a local function randomly generates the current network situation.
Furthermore, the UE was configured to send service requests to the preconfigured SGW.
Another functional entity we extended is the SIP servlet engine. The JAIN SIP SE reference implementation was extended with the PDP publication and discovery protocol and a registration/deregistration module. An HTTP client was also added to the SSE.
Finally, our SGW was implemented with the following functions: SIP stack, PDP publication and discovery protocol and a registration/deregistration handler.
Overall, the prototype shows that it is possible to provide SIP servlets-based 3G services to MANET users and execute these services in the MANET, instead of running them in 3G, thus demonstrating the feasibility of our architecture.

8.2 Performance evaluation of the integrated 3G/MANET architecture

In this part we describe the simulation environment and then we present the system design. The simulation scenarios we used are then described followed by an analysis of the performance results.

8.2.1 Simulation environment and settings

The simulation was performed using OPNET V.11.5.A which is a high-quality commercial tool for the rigorous study of different types of networks. It provides an environment for the modeling and simulation of communications networks, distributed systems, devices and protocols [112]. The user development environment of OPNET is based on the finite state machine. Furthermore, the programming language is Proto-C - a combination of C, C++ and OPNET event simulation APIs.

OPNET provides several models and modules with which to build customized networks. For our integrated 3G/MANET simulation we used the standard MANET and UMTS modules to respectively model the MANET and 3G sub-networks.

We built our integrated 3G/MANET with a one-cell UMTS system and a group of mobile wireless nodes. A gateway between the two sub-networks was simulated. The UMTS part contains the following core nodes: a node-B (i.e. the WCDMA base transceiver station), a Radio Network Controller (RNC), a Serving GPRS Support Node (SGSN) and a Gateway GPRS Support Node (GGSN). The MANET nodes are based on the IEEE 802.11 standard and configured with the AODV routing protocol.

Furthermore, the gateway is connected to the node-B using a wireless connection. The Application Server (AS), the Context Information Base (CIB), the HSS and the S-CSCF are connected to the UMTS via a hub. Figure 8.3 gives an overview of the simulation setup.

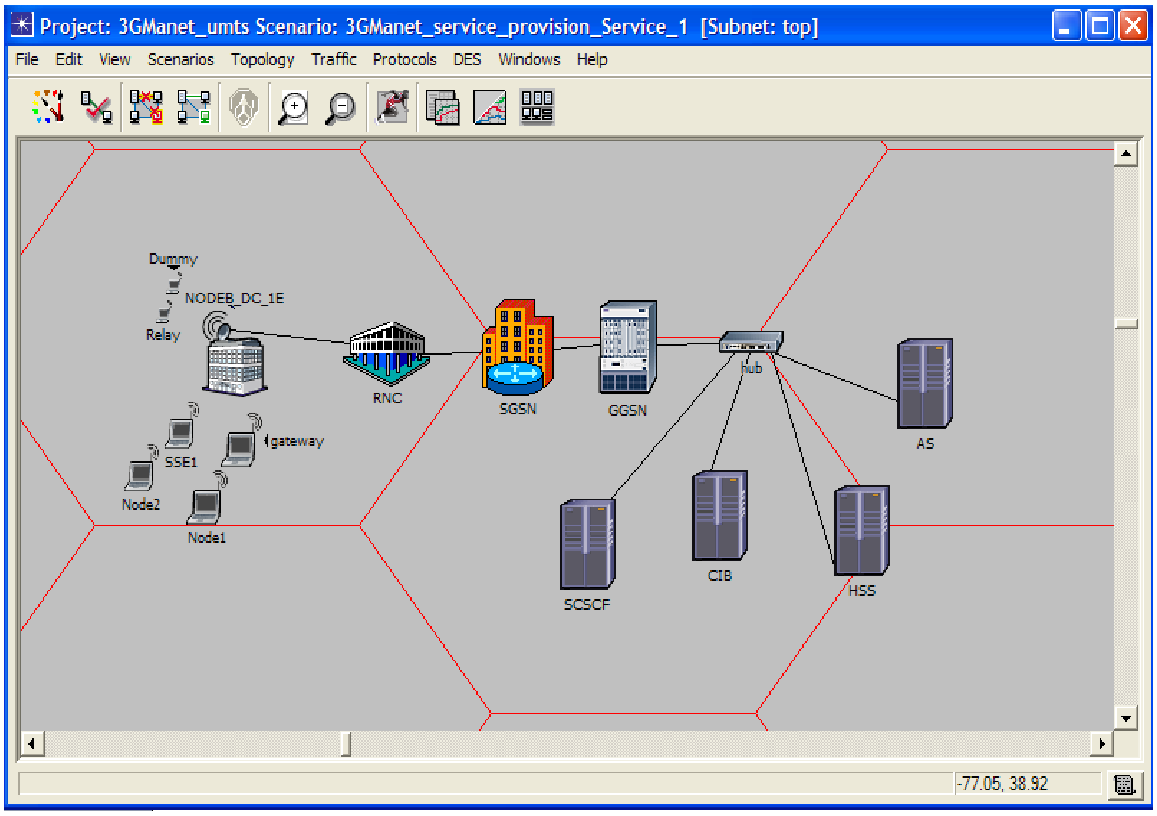

Project: 3GManet_umts Scenario: 3GManet_service_provision_Service_1 [Subnet: top]
File Edit View Scenarios Topology Traffic Protocols DES Windows Help
Dummy
NODEB_DC_1E
Relay
RNC
SGSN
GGSN
hub
AS
SSE1
gateway
Node2
Node1
SCSCF
CIB
HSS
-77.05, 38.92

Figure 8.3: Overview of the integrated 3G/MANET simulation setup

8.2.2 System design

In OPNET we first design the network, then for each node we specify a *node model* and one or more *process models*. The packets' format should also be declared so that it can be recognized by the tool's environment.

Node models:

The node model of the mobile wireless nodes is shown in figure 8.4, while figure 8.5 shows the node model of the 3G nodes (i.e. S-CSCF, AS, HSS and CIB). The difference between the two node models is at the application level. Indeed, we implement the application layer that corresponds to each type of entity, and so we developed different process models.

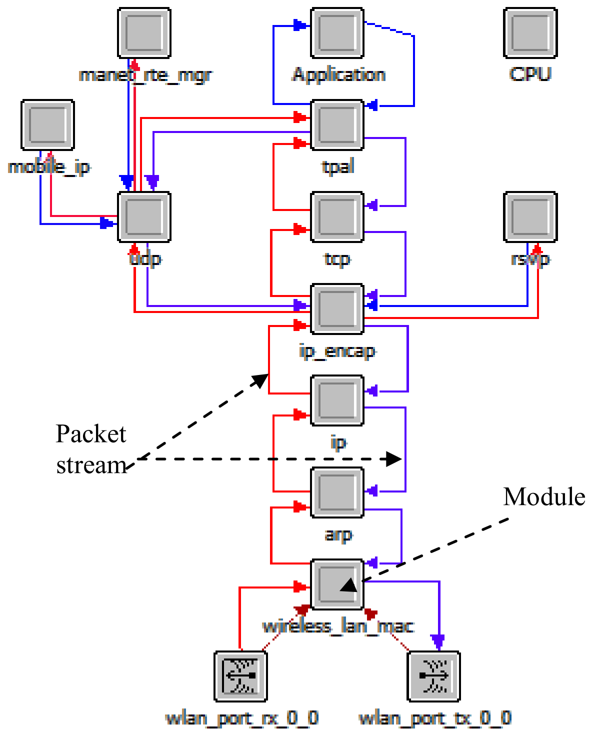


Figure 8.4: Node model of the wireless nodes

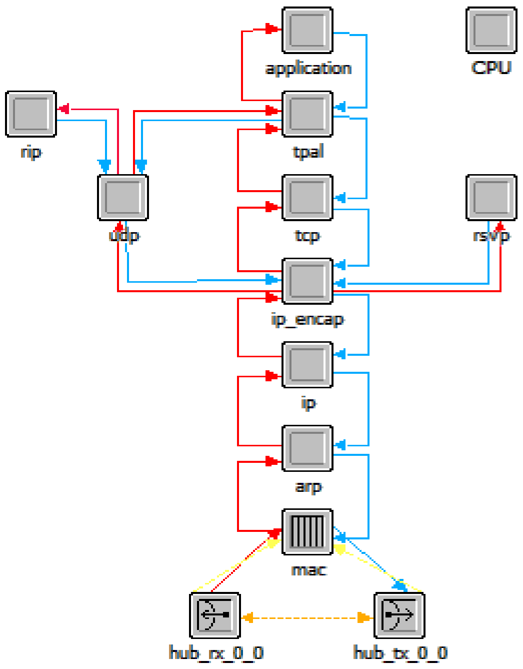


Figure 8.5: Node model of the 3G nodes

The application layer is on top of the Transport Adaptation Layer (TPAL). The TPAL module provides uniform access to the transport layer. Furthermore, nodes are identified using symbolic names. In our case we configure TPAL to use the TCP protocol.

Process models:

The functionalities of each entity are described in chapter 6. In the following section we illustrate the most relevant of the implemented processes. The service gateway (SGW) is essentially responsible for managing the SIP servlet engines' registration and for making the 3G structure transparent to the MANET users. The process model of the SGW is composed of an SGW manager and an SGW agent. The SGW agent process model is presented in figure 8.6.

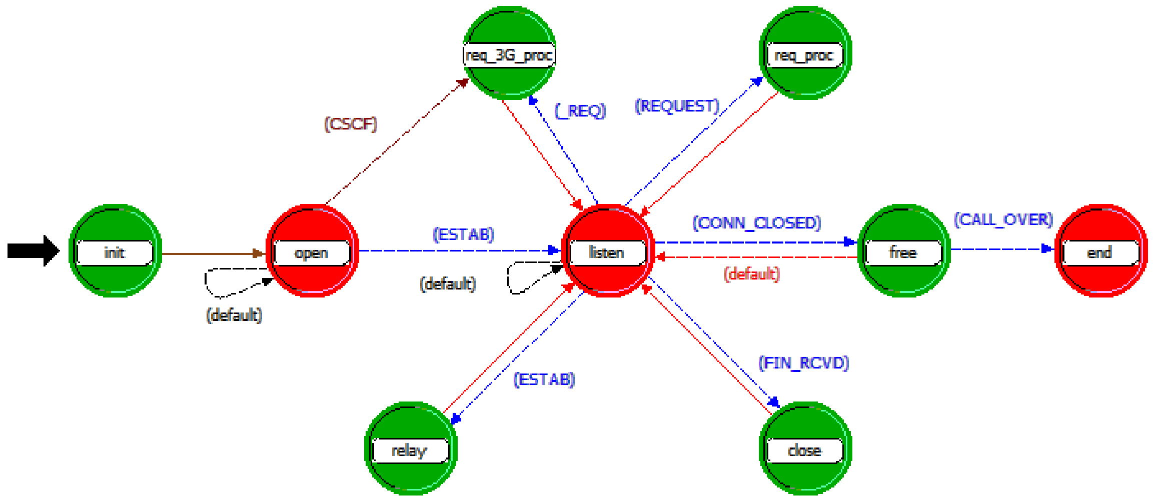


Figure 8.6: The SGW agent process model

Figure 8.7 presents the process model of the SIP Servlets Engine (SSE) agent that is responsible for service execution. It has been extended to register with a pre-configured or a discovered SGW and to download the SIP servlets from remote locations prior to service execution.

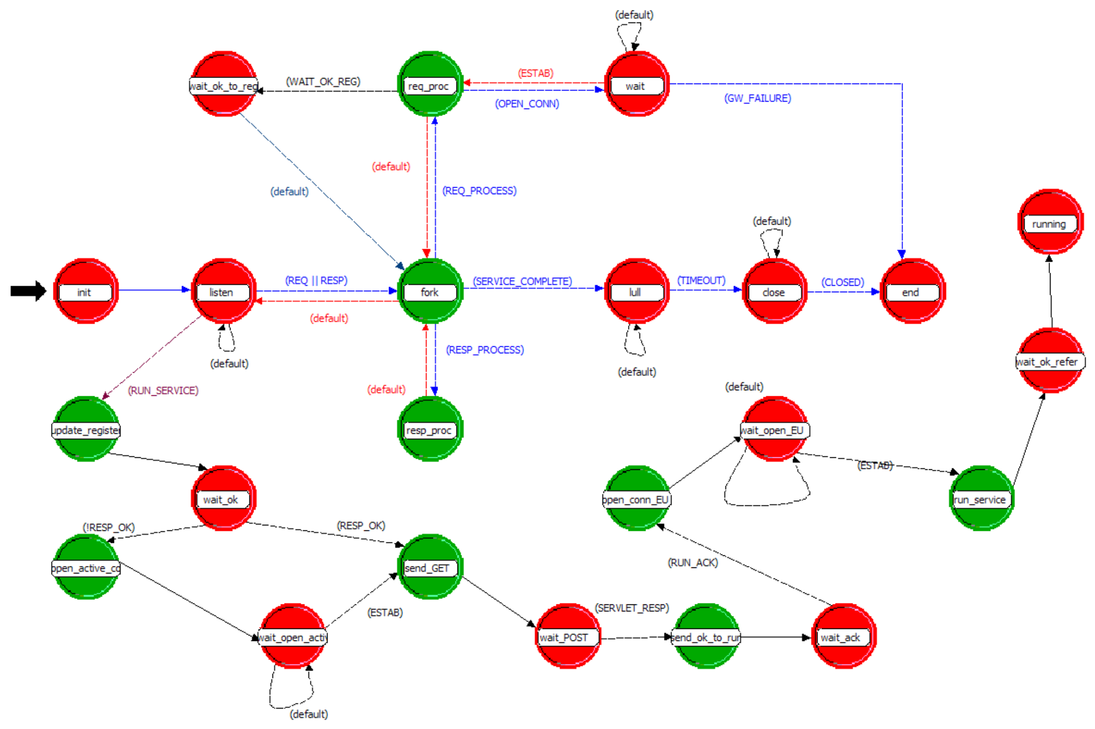


Figure 8.7: The SSE entity process model

Three process models define the AS process model: the AS Manager (ASM) process model, the AS Agent (ASA) process model and the Remote Information AS (RIAS) process model.

The ASM is the root process, responsible for listening to the incoming connections and managing the collected statistics. The ASA is the main application server process. It implements the behaviour of the AS entity as described in the chapter 7. Furthermore, it creates the RIAS process, on demand, in order to get the location information from the HSS or to obtain the context information from the CIB. Figure 8.8 illustrates the main process model of the application server entity.

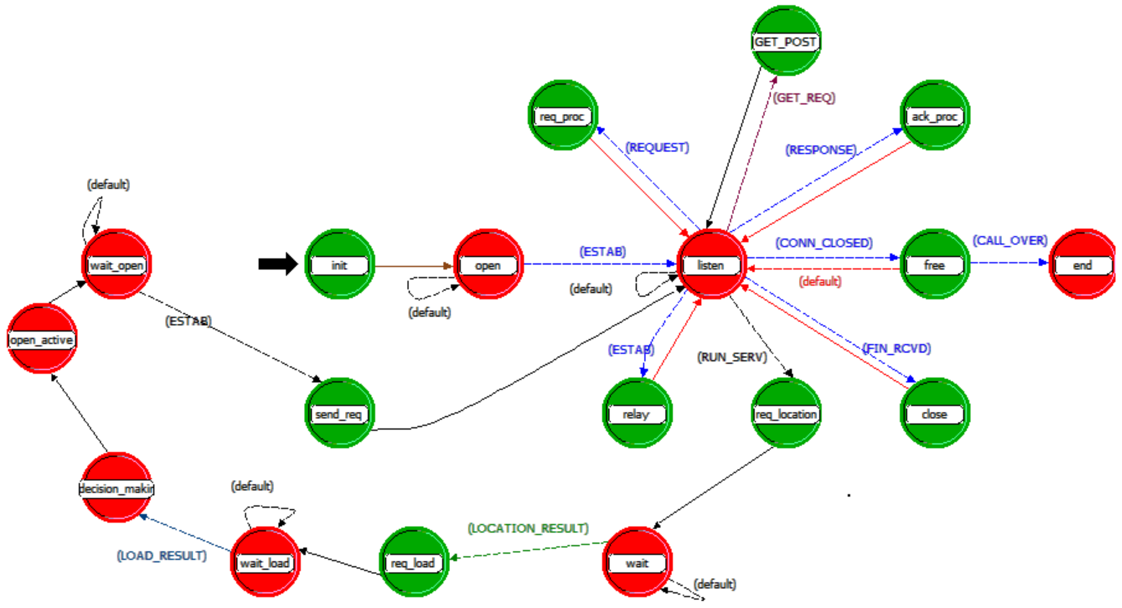


Figure 8.8: The main process model of the AS entity

Packet format:

We have defined several packet formats for the communication between the different entities. Register, service-invoke, service-run, location/load and http get/post are the main packets.

The register packet is used by the SSE to register, deregister or to update the registration information (i.e. capacity, TTL) with the service gateway. A service-invoke packet is used by MANET end-users to initiate the interest-based conference service. The service-run packet is sent from the AS to the SGW to start the service in the MANET network. The location/load packet is used to get the end-users' location from the HSS and the context information from the CIB. Finally, the http get/post is a simple implementation of the http protocol to download the servlets from their location. Figure 8.9 illustrates the service-invoke packet.

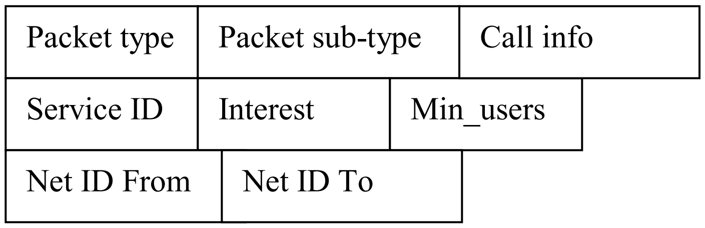


Figure 8.9: The service-invoke packet used for simulation

The packet type is *Serv_Inv_Pkt*. The possible values for the packet sub-type field are: serv_inv_pkt_req, serv_inv_pkt_resp, serv_join_pkt_req and serv_join_pkt_resp. These are request-response pairs for invoking and joining a service.

The service ID identifies the name of the service. The interest field specifies the interests that should be shared between the conference participants. Min_users is the minimum number of users required to start the conference. Call info contain the information related to the call (e.g. source address, destination address, connection status, client connection time). *Net ID From* and *Net ID To* are added to help the SGW to route the packet between the MANET and 3G systems.

8.2.3 Simulation scenarios

We have defined two major scenarios to evaluate the performance evolution of the integrated 3G/MANET architecture. We focus on conference establishment with new users and different services.

In the first scenario, we evaluate the impact of a growing number of users on a given service. The conference is started when the min_users is reached and new users subsequently join this conference. We stopped at 50 MANET end-users participating in the same conference, which we consider a very large conference.

The second scenario measures the impact of the number of concurrent services on network performance. Multiple services with different interests and min-user parameters were defined. The number of simultaneously-running services in the MANET range from 5 to 50. In that context, a small MANET service network is one running 5 services, while 50 services is considered to be a very large MANET service network.

The processes of service invocation and join are the same in both scenarios. Since OPENET does not allow user interaction during our simulation, we have defined two parameters for the MANET end-user entities: *Initiator* and *Joiner*. The *Initiator* will initiate the service (i.e. send a service invoke message) and the *Joiner* will join the service with the pre-configured interest.

The *Initiator* sends a service-invoke message, with the interest and minimum number of users required to start the conference, to the service gateway. The service gateway (SGW) forwards the packet to the S-CSCF which in turn forwards it to the application server (AS). The AS then starts a process to receive the request to join the conference and verifies the interests and the minimum number of users. When this objective is reached, the AS starts a process to check the users' location and the network load, and then decides to run the service in the MANET or not. If the service is to be run in the MANET, the AS sends a service-run message to the SGW via the S-CSCF. The SGW then selects, from the list of registered SSEs, the SIP servlet engine that has the highest capacity and TTL. The service-run request is then sent to the selected SSE, which in turn starts the conference.

From the description above, some stochastic phenomena have to be modeled. In fact, probability models are required to define the number of SSEs that should be present in the network, the capacity of each SSE, and the traffic load generated by the CIB.

Therefore, the first question to answer is: how many SSEs are present in the network while the MANET is growing, and in both scenarios? We generate this number using the binomial low, since the phenomena to model is about choosing *x* entities with the SSE characteristics among *n* entities. This statistic law has two parameters: *n* -- the size of the sample (i.e. the number of nodes), and *p* -- the frequency of the SSE in the network. In our stochastic model, the parameter *n* ranges from 10 to 50. In the MANET system we can have three different types of nodes: end-users, SSEs, and eventually, service gateways. It is obvious that end-users are more likely to join the MANET than the SSEs and service gateways. Therefore, the parameter *p* of the binomial law was fixed to *0.3*.

The second question is: what is the value of the capacity for each SSE? The value in both scenarios is generated following the uniform law since each SSE has equal probability to have a given value for its capacity. The outputs of the uniform law are chosen to belong to the interval [5, 20]. Thus, the minimum capacity allowed for an SSE is 5 and the maximum is 20.

Finally, the traffic load is simulated rather than calculated. In fact, four values are possible as a response from the Context Information Base (CIB): low load, regular load, heavy load and crisis situation. We use a uniform law to generate the responses, with a 35% chance to get a low or regular load, a 20% chance to get a heavy load, and a 10% chance to get a crisis situation.

Each scenario is executed five times and the averages are calculated when the executions end correctly.

8.2.4 Results and analysis

For the performance evaluation we focus on the scalability of the integrated architecture in terms of the number of users and the number of services. This sub-section describes the metrics we measured during the simulation. Then, it presents and discusses the results of the first scenario where the impact of the number of users is evaluated. Finally, it presents and analyses the results of the second scenario, which evaluates the impact of the number of concurrent services on the architecture's performance.

8.2.4.1 Metrics

We have considered the following metrics for the architecture evaluation:

- *Delay:* the delay is calculated in seconds. Two types of delays are calculated: end-to-end and packet delay. The end-to-end delay refers to the time elapsed between when the request is sent and its corresponding response reception. The following requests are considered: SSE registration, EU join, EU service invocation and AS service-run. The delay is calculated when the OK response is received by the sending entity. The AS service-run request contains two delays: one includes the load and location transactions, and the other is the delay after the load and location request finishes. Furthermore, the delays include the internal processing, such as the AS's decision making and the SGW's SSE selection. Figure 8.10 illustrates the different end-to-end delays. Only the major messages are shown in the figure.

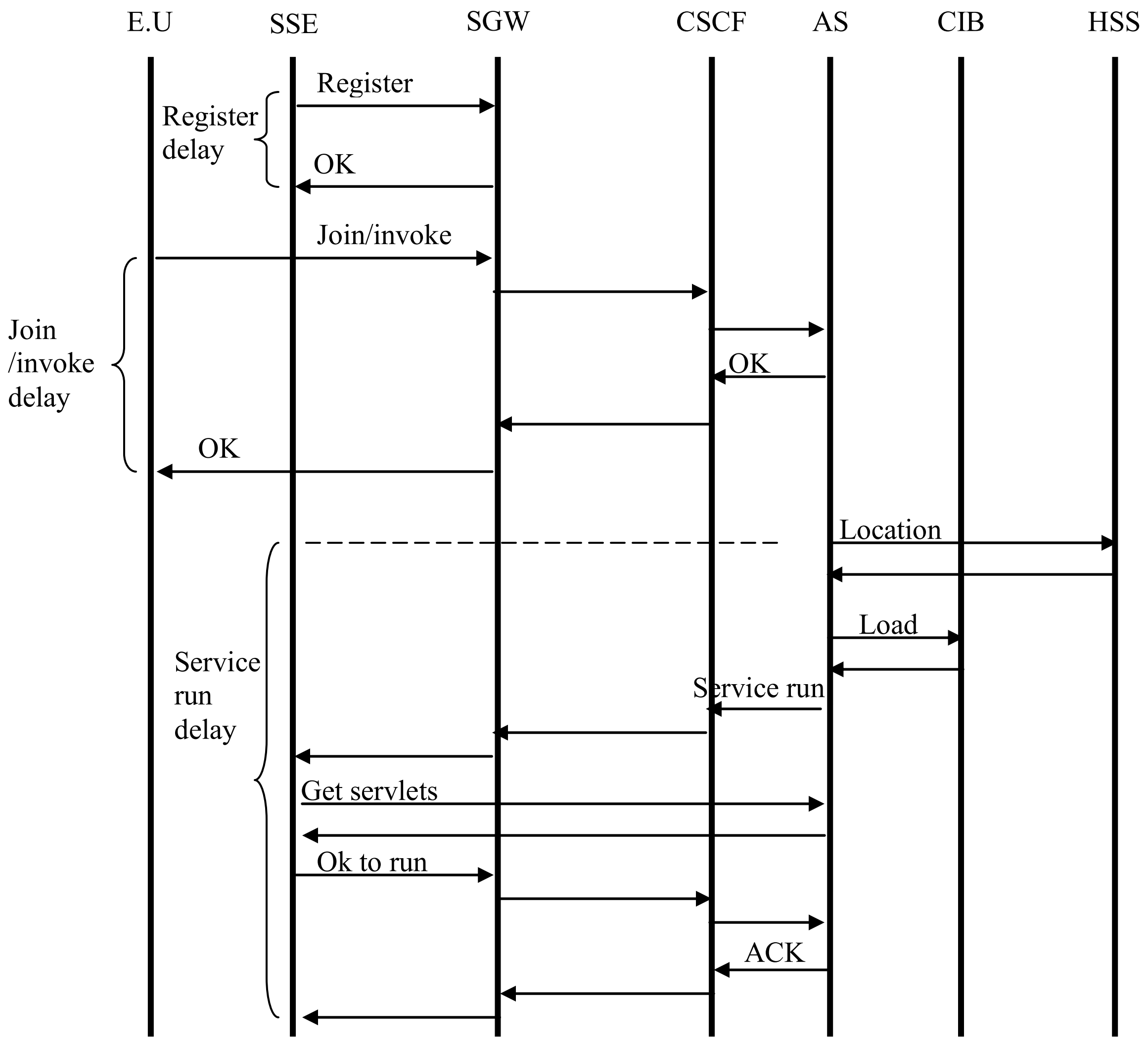


Figure 8.10: Illustration of the calculated delays

The second type of delay that we measure is the packet delay. This is the average propagation time of all of the packets received by the SIP Servlets Engine (SSE), the Service Gateway (SGW), the End-User (EU), the Context Information Base (CIB), the Application Server (AS) and the Home Subscriber Server (HSS). It is the difference between the time a packet is created and the time that packet is received by any of the previous entities.

- *Network load*: the load represents the number of messages received by all the network entities. The number of messages each node receives are calculated for each scenario. This load figure gives an idea of the network load. Furthermore, we calculated, in bytes, the load generated at all the entities (i.e. EU, SSE, SGW, AS, CIB and HSS). The load generated by all of the packets (received or sent) that involve a given entity is calculated, which provides a good estimation of the workload at a given network node.

8.2.4.2 The impact of the number of users

In this scenario we use one service, and all users have the same interest. The minimum number of users required to start the service is fixed at 2. Users may join at anytime. Next we present the packets' delay, the end-to-end delay and the load results.

a. Packets' delay

Figures 8.11, 8.12, 8.13, 8.14, 8.15 and 8.16 show the average delay of all of the packets sent by the AS, the CIB, the HSS, the EU, the SSE and the SGW, respectively.

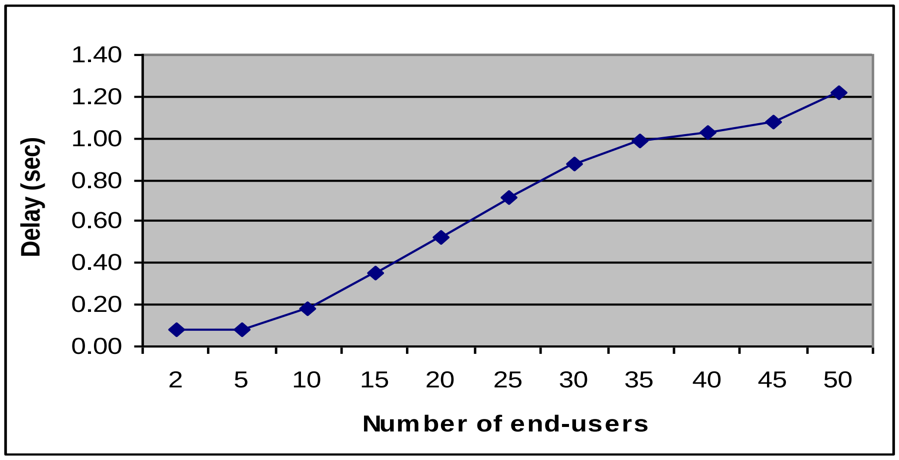


Figure 8.11: The average packet delay for the AS entity

The delay grows linearly with the number of users. It remains under 1 second from 2 to 35 end-users involved in the same service. It starts at 0.07 second and ends at 1.2 seconds. At 40 end-users, the delay starts to exceed 1 second. It is about 1.02 seconds and it appears that the curve is increasing at a greater rate after 45 end-users.

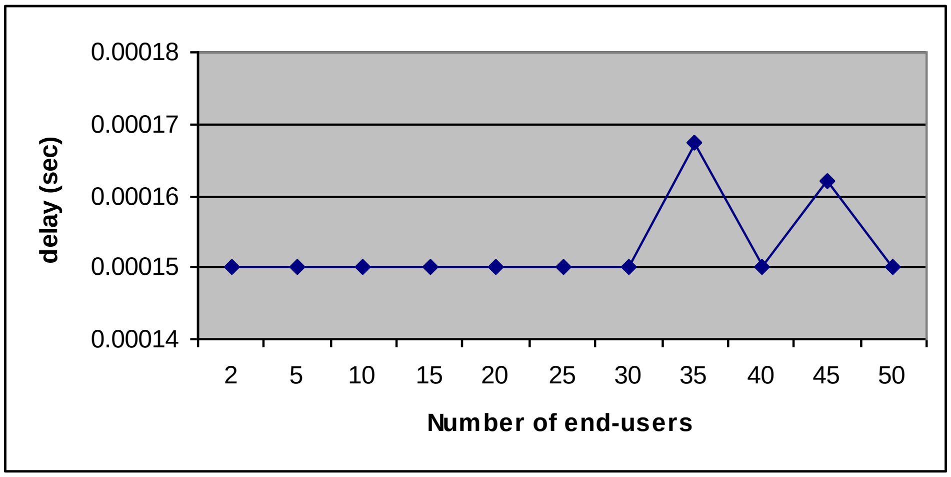


Figure 8.12: The average packet delay for the CIB entity

The CIB is requested for context information (i.e. network load) before service execution. The delay at the CIB is very low and virtually stable at 0.15 milliseconds. At 35 and at 45 end-users a variation is noticed. The delay jumps, but remains under 1.17 milliseconds. This may be due to local network rush or the machine being slow, or due to our implementation, since the CIB packet delay is most likely to be steady around 0.15 milliseconds.

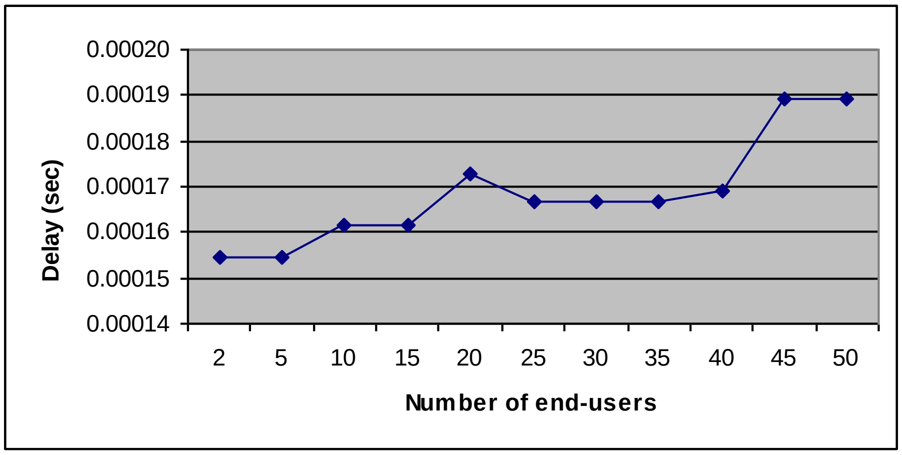


Figure 8.13: The average packets delay for the HSS entity

The HSS is requested to check the end-users location. The delay varies from 0.15 to 0.19 milliseconds, which is an insignificant delay. The curve grows slowly (the variation is about 0.01 millisecond) until 40 end-users, when the slope of the curve becomes greater.

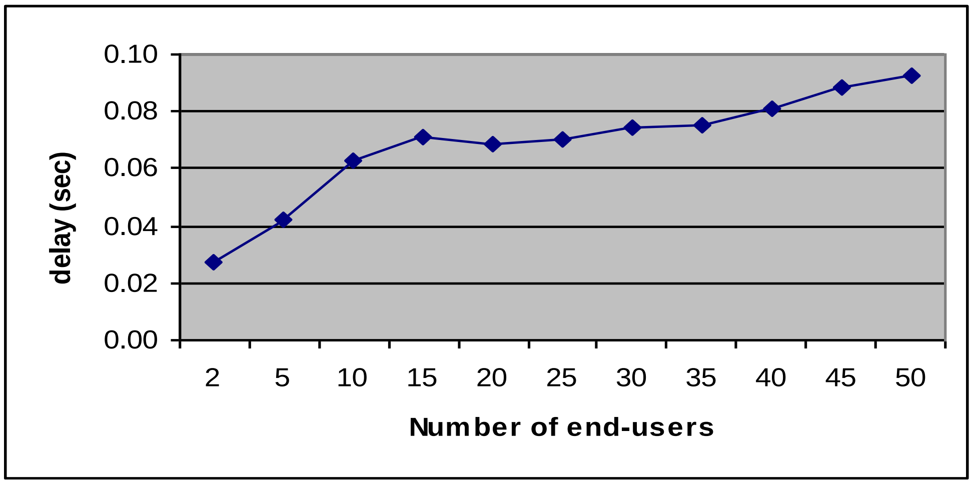


Figure 8.14: The average packet delay for the EU entity

At the end-user entity, the delay varies from 0.027 to 0.092 seconds. This is a very reasonable delay. Furthermore, the delay grows rapidly at the beginning (i.e. from 2 to 10 end-users), and then it stabilizes at 0.07 second with very low deviation. The variation increases after 40 end-users join the same service. However, the end-user perception is not affected, since the delays are still low and many other users can join the same service. Figure 8.15 shows the average packet delays sent by the SIP Servlets Engine (SSE). With from 2 to 40 end-users, the delays increase by less than 1 millisecond. This shows that the impact of the number of end-users on the SSE delay is limited. After 40 end-users the curves grow more quickly but the average delay remains low (13 milliseconds for 50 end-users).

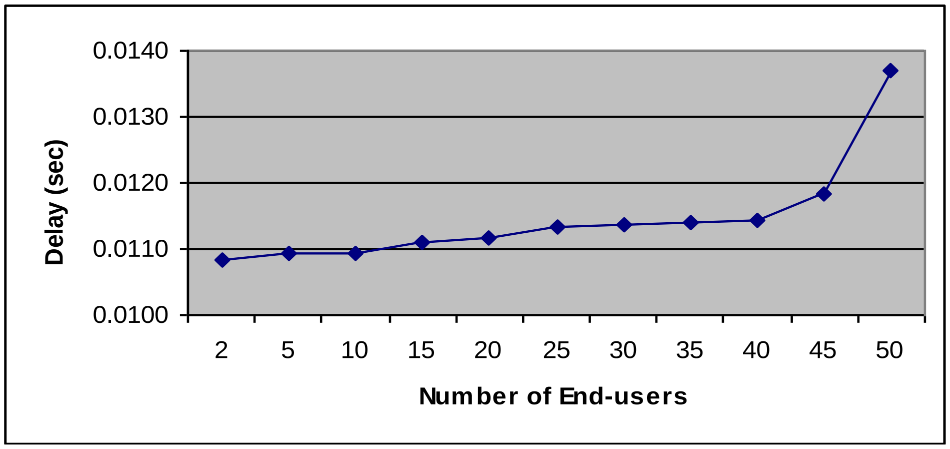


Figure 8.15: The average packet delay for the SSE entity

Figure 8.16 presents the results related to the average delay at the service gateway entity (SGW).

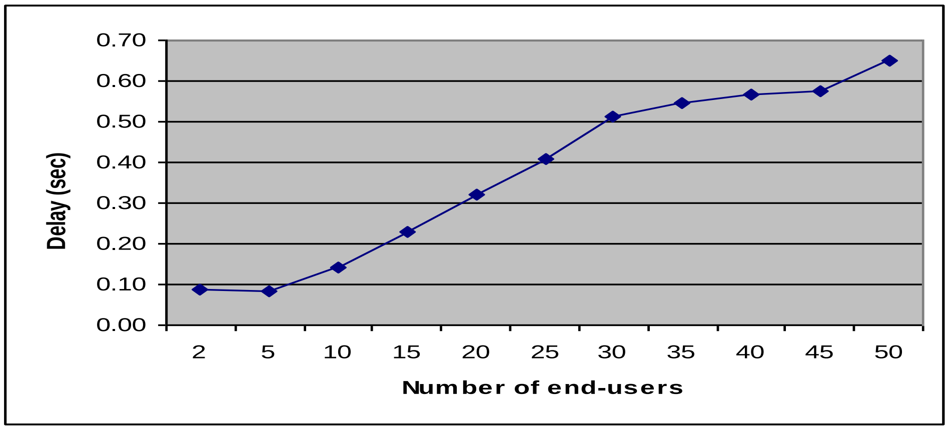


Figure 8.16: The average packet delay for the SGW entity

The delay at the SGW starts at 0.08 seconds and grows almost linearly with the number of end-users until this number reaches 30. This can be explained by the fact that the SGW is the bridge between the MANET network and the 3G network. Therefore, all the messages pass through the SGW. However, after 30 end-users, the delay varies less and seems to stabilize at around 0.5 seconds, thanks to the network and system stability. The curve's slope then increases at 50 end-users, but the delay remains under 0.07 seconds.

b. End-to-end delay

In this sub-section the end-to-end delays' results are presented. Five main requests are considered: the service-run request before and after the location and context information requests, the service-invoke request, the service-join request and the SIP servlets engine register request. The end-to-end delay is observable by the end-users and therefore, it is important to evaluate and minimize.

Figure 8.17 presents the evolution of the end-to-end delay of the service-run request, as shown in figure 8.10.

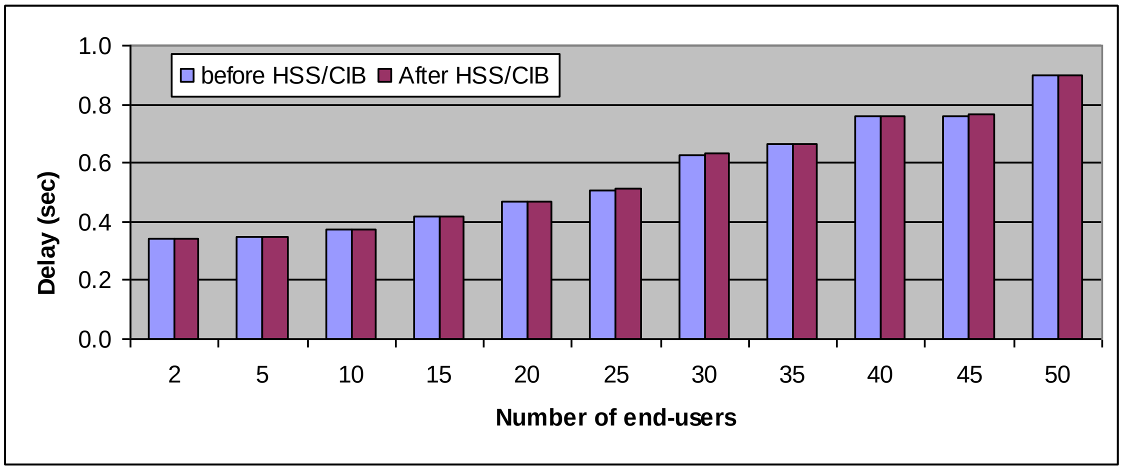


Figure 8.17: The average end-to-end delay for the *service run* request

The average end-to-end delay is calculated at two different moments: before the application server requests the users' location from the HSS and the context information from the CIB, and after requesting this location and context information. However, since the packet delays for users' location and context information (i.e. network load) are very low, as shown in figures 8.12 and 8.13, the average end-to-end delays for the service-run requests are similar. Indeed, there is no observable difference, from the end-user's point of view, between a service-run request delay that includes the users' location and load transactions' delay and the one that excludes these transactions.

Furthermore, the end-to-end delay is growing slightly with the increasing number of end-users. The delay remains under 1 second for 50 end-users, which we consider a large service network. The delay at 50 end-users is about 2.6 times the delay with 2 end-users.

Figure 8.18 presents the average end-to-end delay for the End-User (EU) service-invoke and service-join requests.

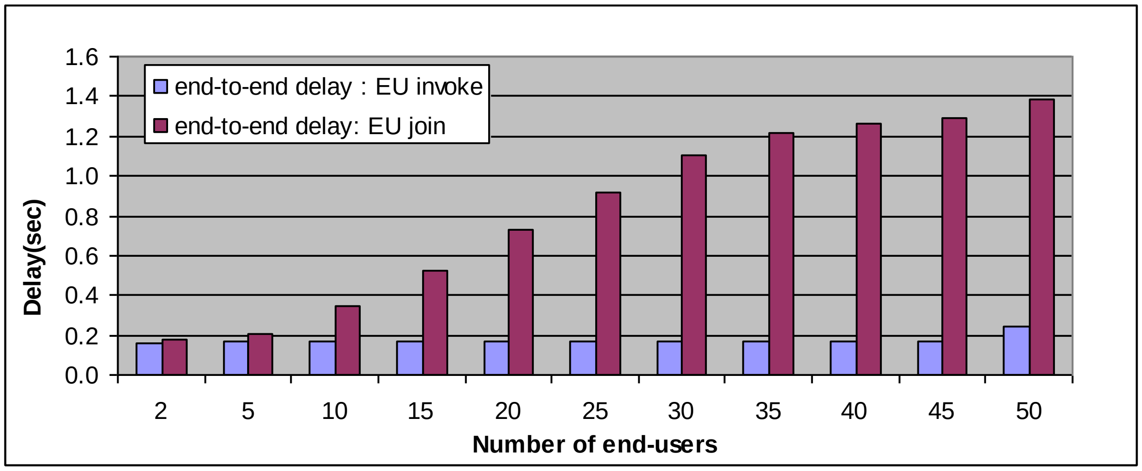


Figure 8.18: The average end-to-end delay for the *service-invoke* and *service-join* requests

We notice that the service-invoke request does not significantly change with the number of end-users. It remains almost stable at around 0.16 seconds. At 50 end-users, the delay jumps to 0.25 but remains low. However, for the service-join request, the end-to-end delay grows quickly as the number of end-users increases. The delay becomes more than

1 second at 30 end-users and is under 1.4 seconds at 50 end-users. The difference between the two requests is the processing time required by the application server and the SGW upon the reception of the join message (e.g. check the location, match the interest and reply to the SGW).

Figure 8.19 shows the end-to-end delay from the SSE provider's perspective. It presents the average end-to-end delay for the SSE register request.

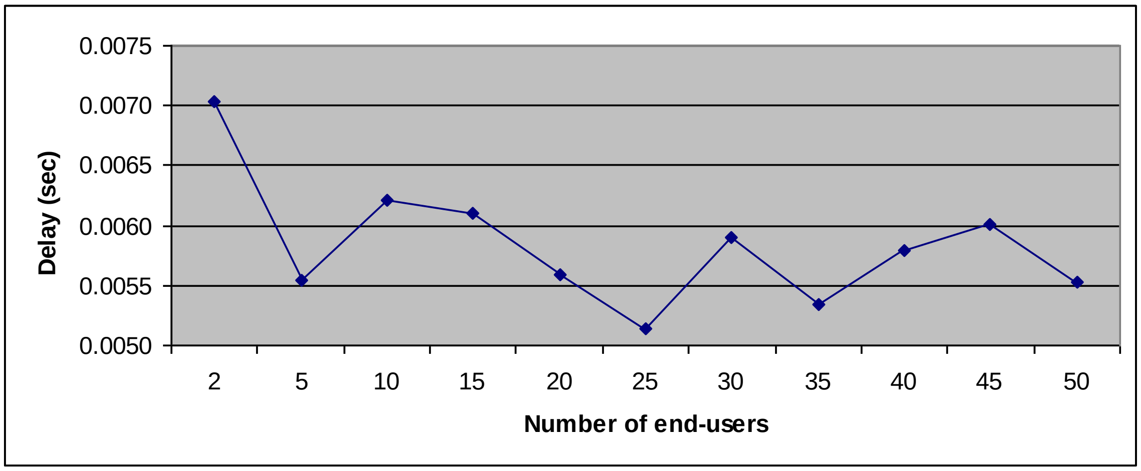


Figure 8.19: The average end-to-end delay for the *SSE register* request

The SSE register end-to-end delay varies from 0.05 to 0.07 seconds, but most of the time the delay is below 0.06 seconds. The variation is very low and we can consider the delay to be stable. The curve is irregular and could be because the number of SSEs in the network follows a stochastic low then the average end-to-end delay variation is affected. Furthermore, the highest value is recorded for 2 end-users, which is explained by the fact that the network is not yet stable at the beginning of the experiment.

c. Network load

This sub-section presents the results related to the network load in terms of the number of packets and bytes sent. The Application Server (AS), the Service GateWay (SGW) and the CSCF are considered. The total number of packets and bytes exchanged in the network are also presented. The load considered is introduced by the service provisioning process only.

Figure 8.20 illustrates the total packets sent by the main entities: the AS, the SGW and the CSCF, while figure 8.21 shows the overhead in term of the bytes introduced by these entities. The number of end-users grows from 2 to 50.

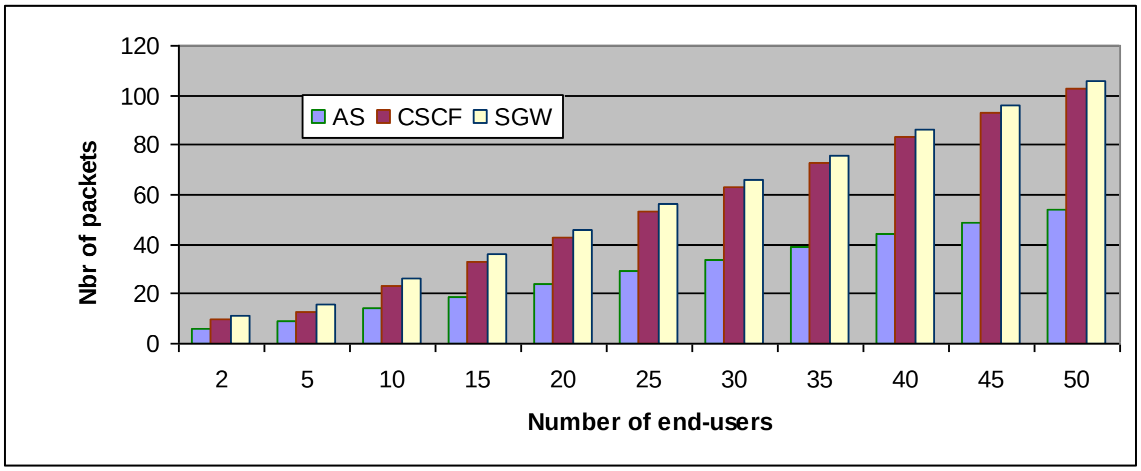


Figure 8.20: Number of packets sent by the main entities

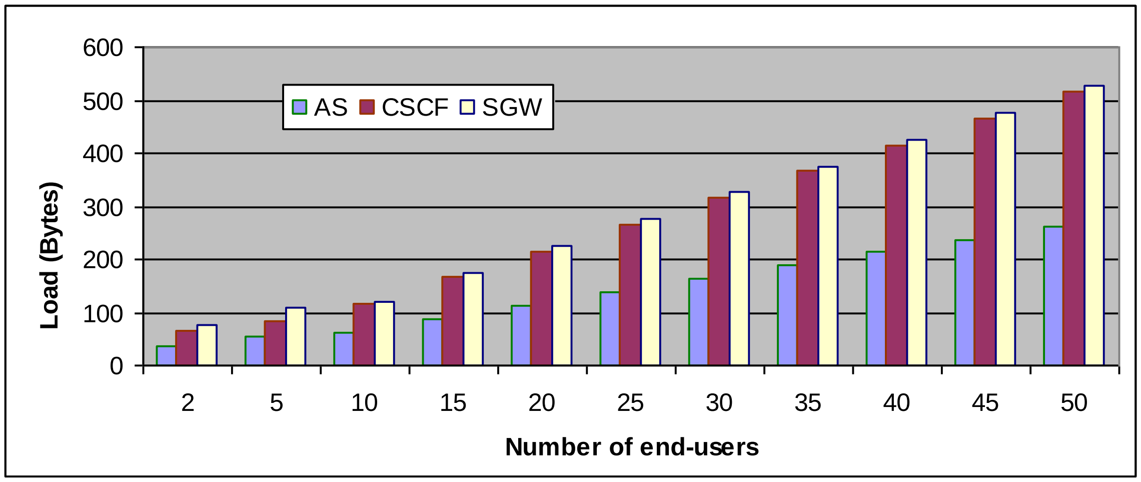


Figure 8.21: Overhead introduced by the main entities

The AS is the entity that introduced the least overhead and that sent fewer packets over the network. The SGW and the CSCF have similar results, with higher values for the

SGW entity. This is comprehensible since the CSCF only forwards the SGW messages to the 3G network.

The maximum number of packets sent by the AS when 50 end-users are involved is 54, while this maximum is 103 and 106 for the CSCF and the SGW, respectively. Furthermore, the overhead introduced by each entity is low. In fact, for 50 end-users the AS generates 263 bytes, the CSCF 516 bytes and the SGW 526 bytes.

Figures 8.22 and 8.23 show the total load injected into the network by the service provisioning process.

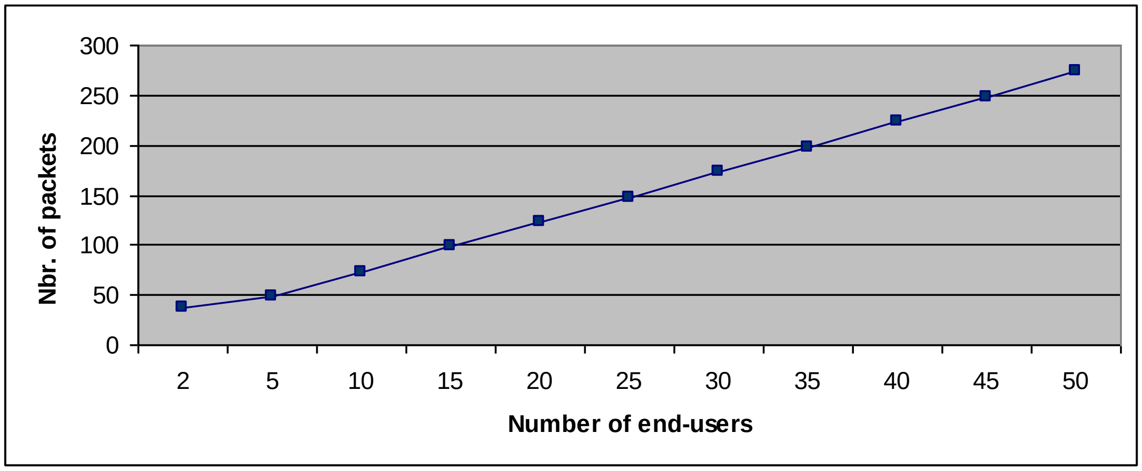


Figure 8.22: Total number of packets exchanged in the network

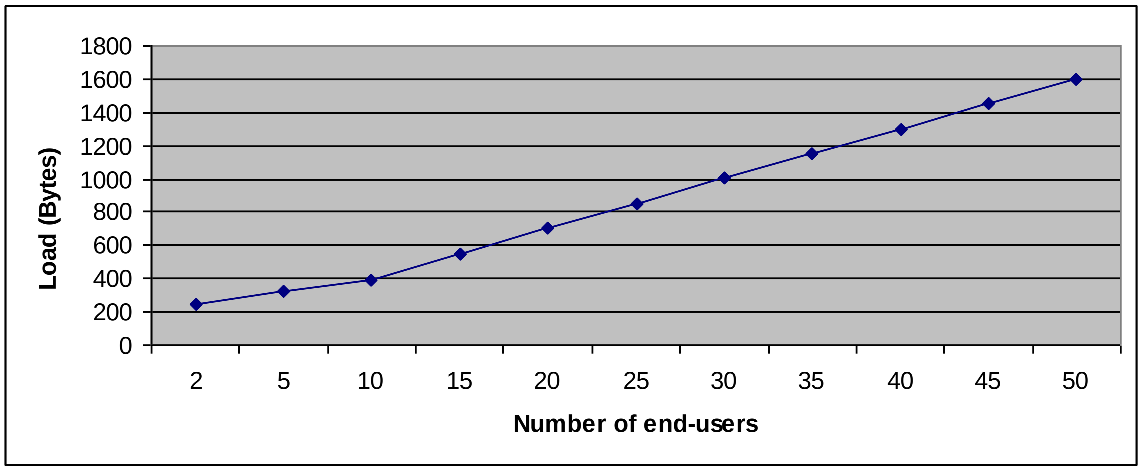


Figure 8.23: Total bytes exchanged in the network

Obviously the load (i.e. number of packets and bytes) is growing linearly with the number of end-users. However, it remains reasonable since the total number of packets introduced by the service provisioning process does not exceed 280 packets. On average, six packets are generated per end-user, which does not affect the global performance of the network. Furthermore, the corresponding overhead in bytes does not exceed 1600 bytes for 50 end-users since the packets are light-weight.

8.2.4.3 The impact of the number of concurrent services

In this scenario we vary the number of services from 2 to 50. Each service has different interest fields as its parameters. The minimum number of users required to start the service is fixed at 2. Basically, several concurrent services run simultaneously, where each service involved two different end-users. We present here the packet delay, the end-to-end delay and the load results.

c. Packet delay

Figures 8.24, 8.25, 8.26, 8.27, 8.28 and 8.29 illustrate the average delay of all the packets sent by the AS, the CIB, the HSS, the EU, the SSE and the SGW, respectively.

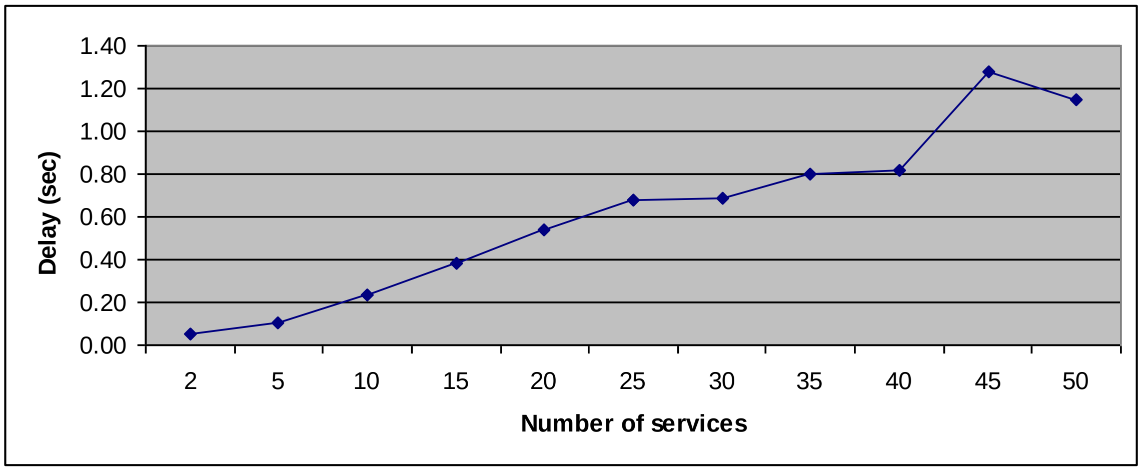


Figure 8.24: Impact of the number of services on the average packet delay for the AS entity

The packet delay at the application server starts at 0.05 seconds for 2 concurrent services and ends at around 1.1 seconds for 50 parallel services. This is a significant delay evolution, and is mainly because the AS is responsible for processing the service requests. Thus, the more services that are invoked, the more the delays are observed. However, the average delay stays under 0.9 seconds up to 40 concurrent services. It then jumps to 1.2 seconds for 45 services, which is considered to be a large service network.

Figure 8.25 shows the impact of the number of parallel services on the average packet delay. The packets considered are those traversing the CIB entity.

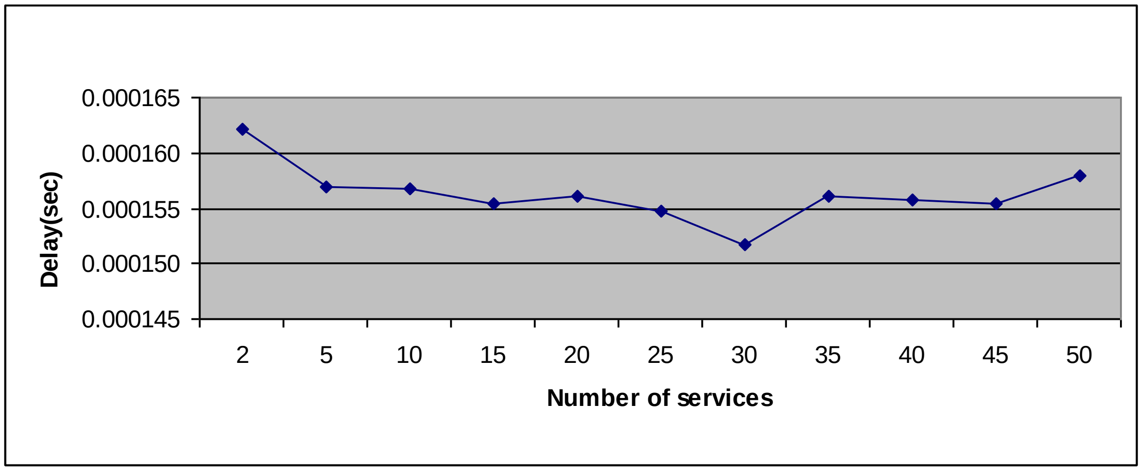


Figure 8.25: Impact of the number of services on the average packet delay for the CIB entity

The average delay at the CIB entity is very low and stabilizes at around 0.15 milliseconds. We can conclude that the number of services does not have a significant effect on the delay at the context information base.

In figure 8.26, the results related to the average delay at the HSS entity due to the number of services are presented. Although the average delay is very low and does not go beyond 0.18 milliseconds, we can clearly see that the number of services has an impact on the delay. Indeed, the curve’s slope is growing with the number of parallel services. We also notice that after 45 services, the slope of the curve become significantly high. However, the variation between the lowest and the highest value of the delay is 0.2 milliseconds. Therefore, the impact remains unimportant.

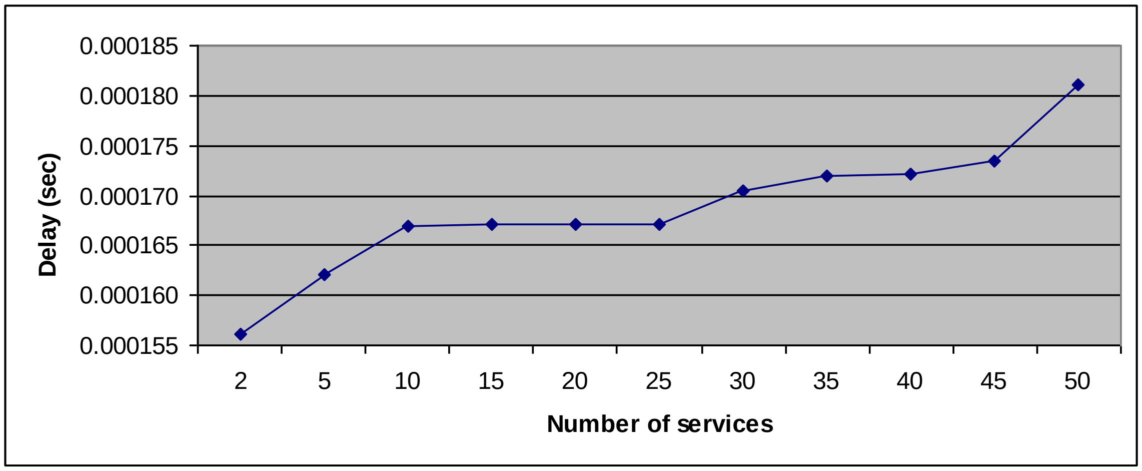


Figure 8.26: Impact of the number of services on the average packet delay for the HSS entity

Figure 8.27 illustrates the evolution of the average delay at the end-user (EU) entity while the number of services is increased.

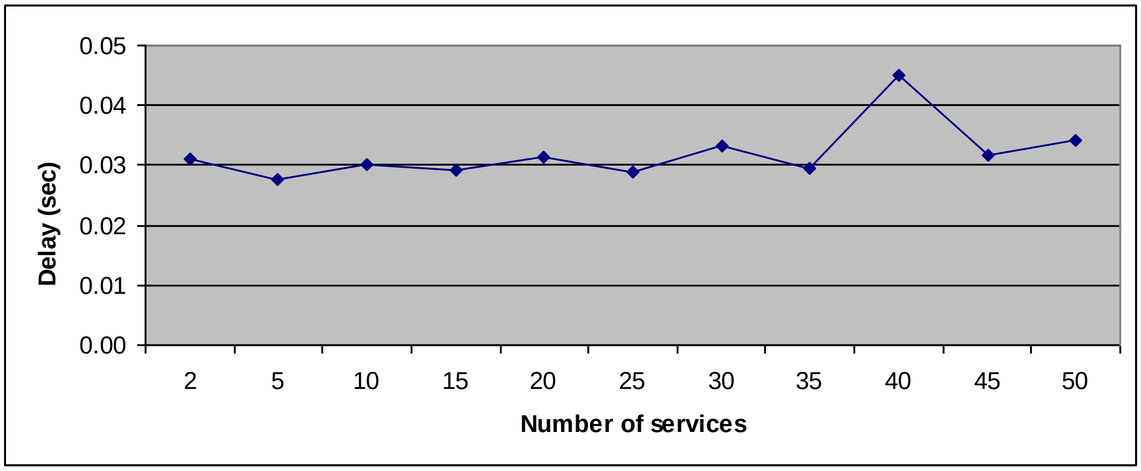


Figure 8.27: Impact of the number of services on the average packet delay for the EU entity

The average delay at the EU entity is stable at about 0.03 seconds. The delay jumps to 0.44 seconds 40 services, but it is an isolated point. The EU perception of the delay is not altered by the number of concurrent services in the network, as the EU receives packets with reasonable delays.

Figure 8.28 presents the impact of the number of services on the packet delay at the service gateway entity.

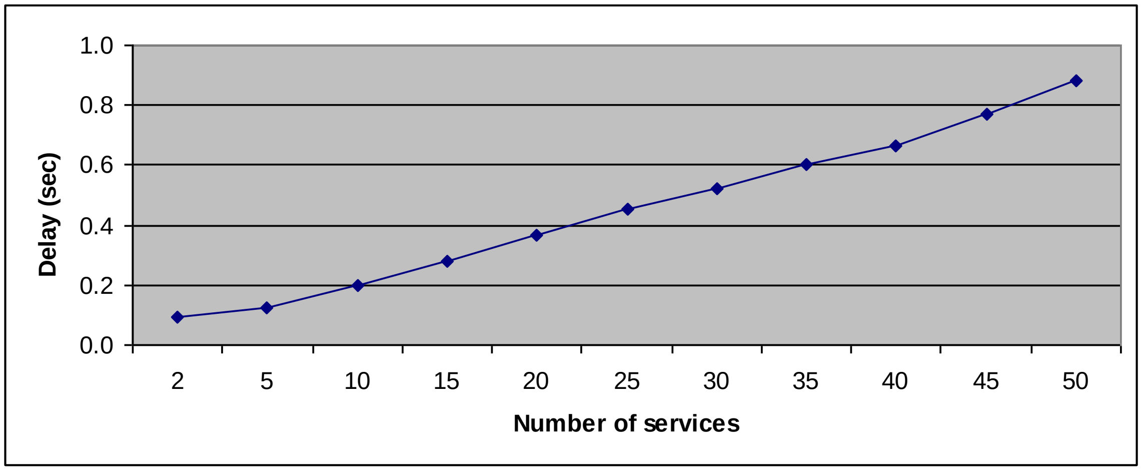


Figure 8.28: Impact of the number of services on the average packet delay for the SGW entity

The delay curve grows linearly. The SGW is the entity in the middle between the 3G and the MANET, so the number of services involved in the network has an impact on the registered delays at the SGW entity. However, the simulation shows that for the service provisioning process, the delays are less than 1 second, even for 50 concurrent services. We also want to mention that even though the delay at 50 services is 9 times greater than it is at 2 services, the average packet delay increase is remains brief.

Finally, figure 8.29 illustrates the results related to the packet delay for the SSE entity.

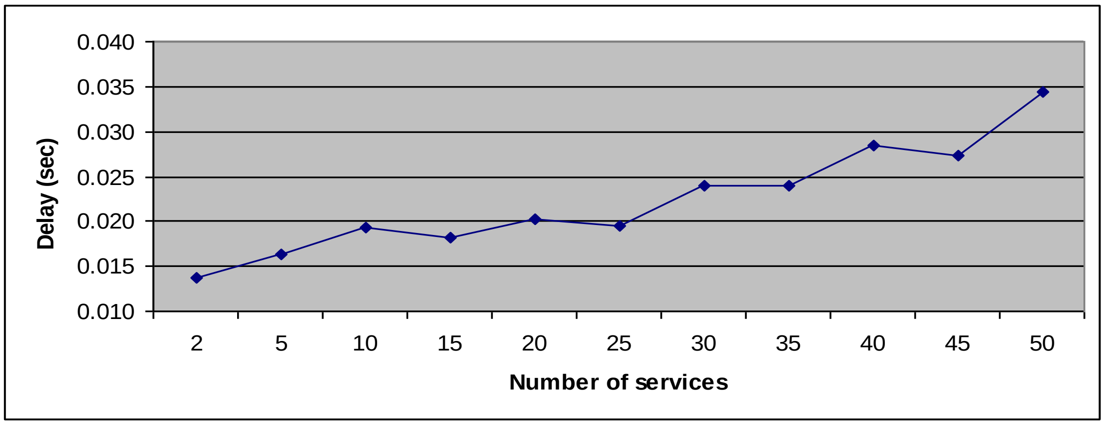


Figure 8.29: Impact of the number of services on the average packet delay for the SSE entity

The average packet delay at the SSE is relatively low. It varies from 13 to 34 milliseconds, which translates to a ratio of 2.5 between the delay at 2 services and the delay at 50 services. Therefore, the number of parallel services does make the average delay increase but the variation remains low, and the performance of the SSE is not greatly affected.

b. End-to-end delay

In this sub-section the end-to-end delays' results are presented. The main requests are considered: the service-run request before and after location and context information requests, the service-invoke request, the service-join request and the SIP servlets engine register request.

Figure 8.30 presents the end-to-end delay's evolution of the *service run* request, according to the number of concurrent services. The graph shows the request delay before and after the AS requests the location from the HSS and the context information from the CIB.

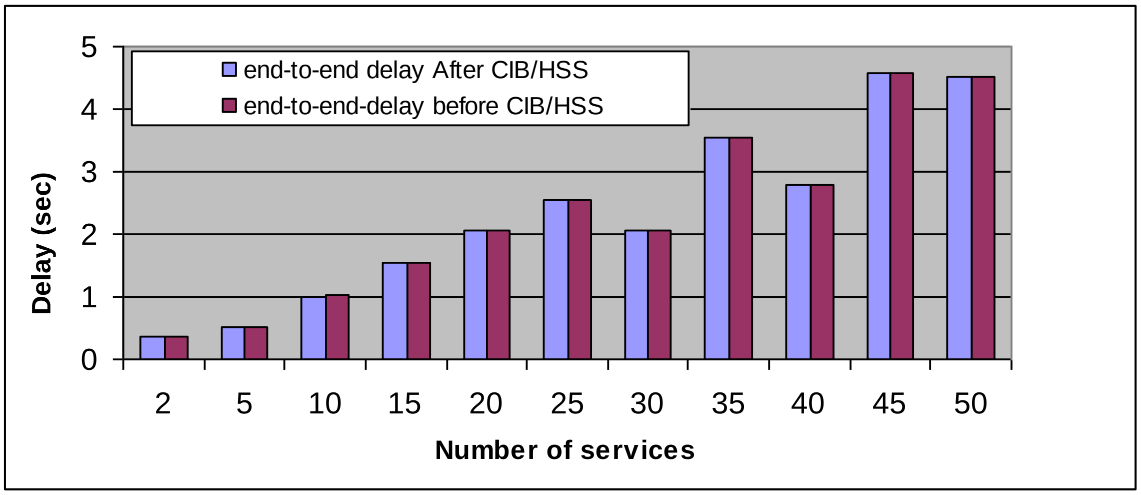


Figure 8.30: Impact of number of services on the average end-to-end delay for the *service run* request

The difference is marginal between the end-to-end delay before and after the location and context information requests. As the simulation showed (figures 8.25 and 8.26) these requests' delays are very low and are not influenced by the number of parallel services. However, the end-to-end delay of the service run request starts at 0.3 seconds for 2 services and ends at 4.5 seconds for 50 services. This variation is high and the delay increases fast. The delay is basically due to the processing time of the application server. In our configuration only one application server was considered, meaning one service provider serving hundred users with 50 different services. Therefore, the performance can be enhanced when different service providers are considered.

Figure 8.31 illustrates the impact of multiple services on the *service- join* and the *service-invoke* requests.

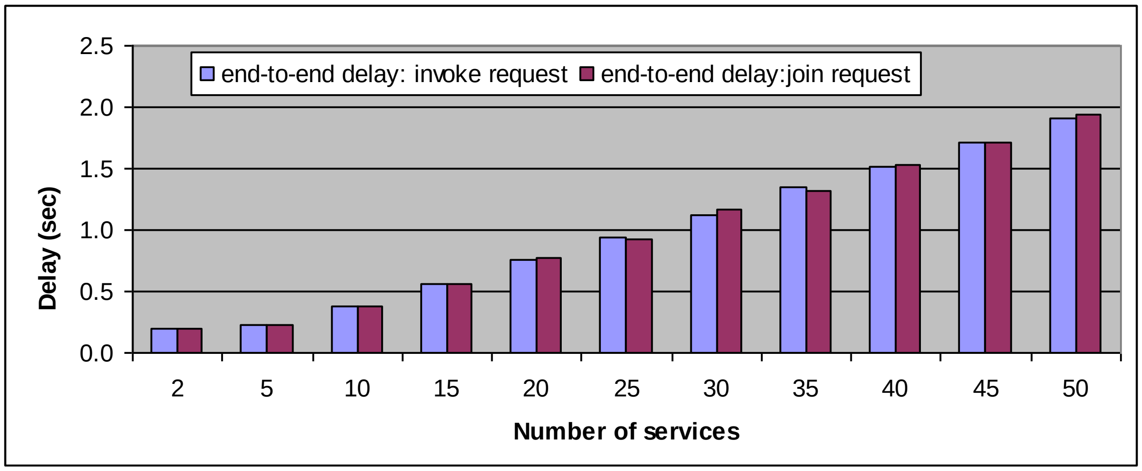


Figure 8.31: Impact of number of services on the average end-to-end delay for the *service-join* and *service-invoke* requests

In general, the average end-to-end delay for the *service-join* request is slightly higher than for the *service-invoke* request, except for the cases of 25 and 35 services, but we consider these values as exceptions. Thus, the number of concurrent services has an impact on both requests, unlike the number of end-users. Furthermore, the average delay is as low as 0.19 seconds for 2 services and reaches 1.9 seconds for 50 services. It is then about 10 times higher at the end of this simulation than at its beginning. However, the delay does remain under 1.5 seconds until 40 services is reached.

Figure 8.32 presents the results related to the end-to-end delay for the SSE register request.

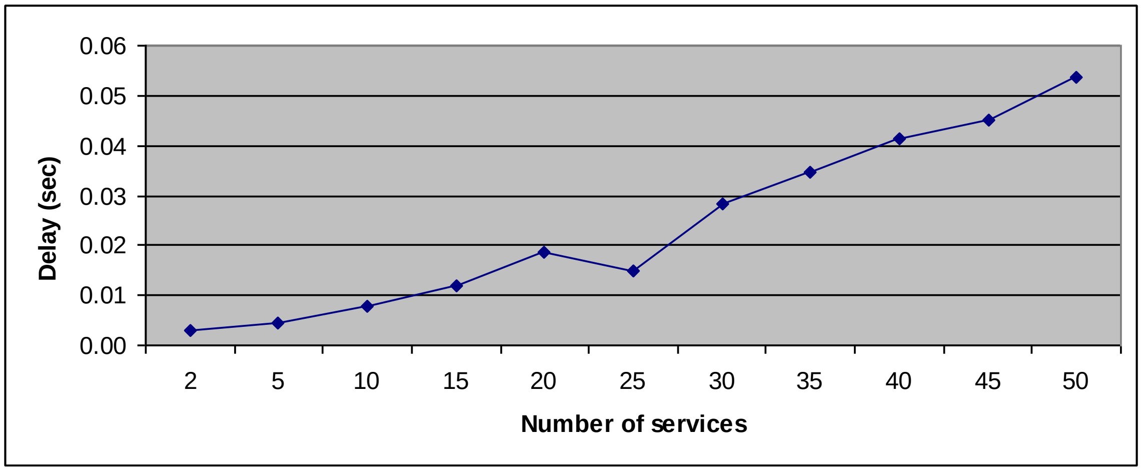


Figure 8.32: Impact of number of services on the average end-to-end delay for the *SSE register* request

The average end-to-end delay is lower for the SSE register than for the previous requests. However, it is higher compared to the impact of the number of end-users (see figure 8.19). Furthermore, the delay remains under 0.053 seconds for all of the simulation times. We notice an exception at 25 services but we do not consider it to be significant. At 2 services the delay is as low as 0.003 seconds. The SSE register end-to-end delay grows linearly with the number of services until 15 services, and then the curve's slope becomes greater. The end-to-end delay at 15 services is four times higher than it is at 2 services. However, it is 2 times higher at 50 services than at 20 services.

c. Network load

This sub-section presents the results related to the network load in term of the number of packets and bytes sent. The Application Server (AS), the Service GateWay (SGW) and the CSCF are considered, and the total number of packets and bytes exchanged in the network are presented. The load considered is only introduced by the service provisioning process.

Figure 8.33 illustrates the total packets sent by the entities AS, SGW, CSCF, SSE, CIB and HSS, while figure 8.34 shows the overhead in term of bytes introduced by these entities. The HSS and the CIB receive almost the same type of requests and they have the same number of packets and overhead.

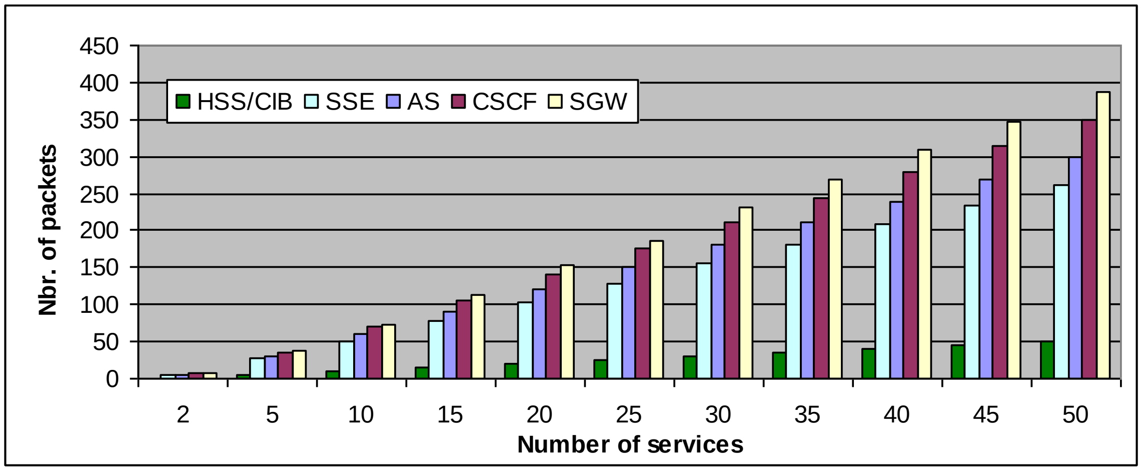


Figure 8.33: Number of packets sent by the different entities

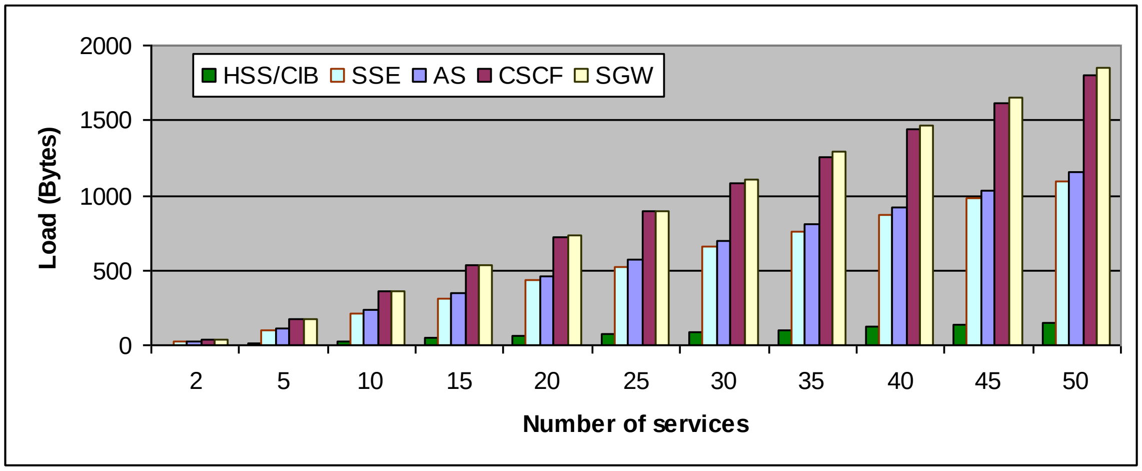


Figure 8.34: Overhead introduced by the different entities

The CIB and the HSS are the entities that introduce the least overhead and exchange the fewest packets. The SSE, the AS, the CSCF and the SGW, are then represented in the order of their overhead and number of packet exchanges. The SGW sends less than 400 messages during the service provisioning process when 50 parallel services are invoked. The CSCF sends around 350, the AS 300 packets and the SSE sends 261 messages. The corresponding load figure is similar. However, the SSE and the AS seem to have the same evolution and generate almost the same load in terms of bytes. This is also true for the SGW and the CSCF. There is a significant difference between the load generated by the SGW and the AS. Indeed, at 50 parallel services the SGW introduces 1848 bytes into the network while the AS introduces 1150 bytes. In terms of the total load in the network, figures 8.35 and 8.36 illustrate the curves that describe the network workload.

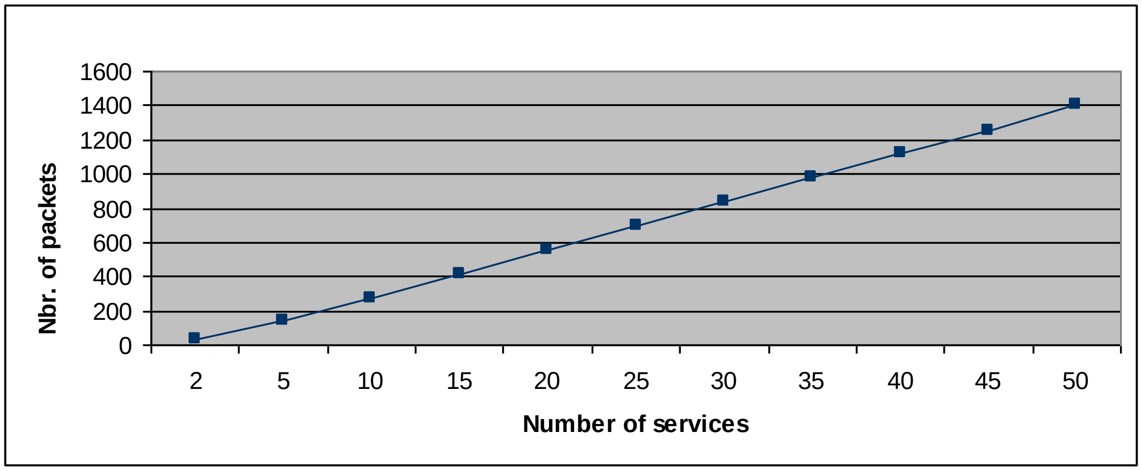


Figure 8.35: Total number of packets exchanged in the network - different services

The curves describing the total number of packet in the network and the total load in bytes introduced in the network are both linear. The values increase quickly. With 2 services the total number of packets is 31, while with 50 services the number of packets in the network jumps to 1400. Similarly, the load in bytes with 2 services is 145 while it becomes 7146 with 50 concurrent services. However, the load remains reasonable and may be reduced using different techniques (i.e. cross layer design, clustering).

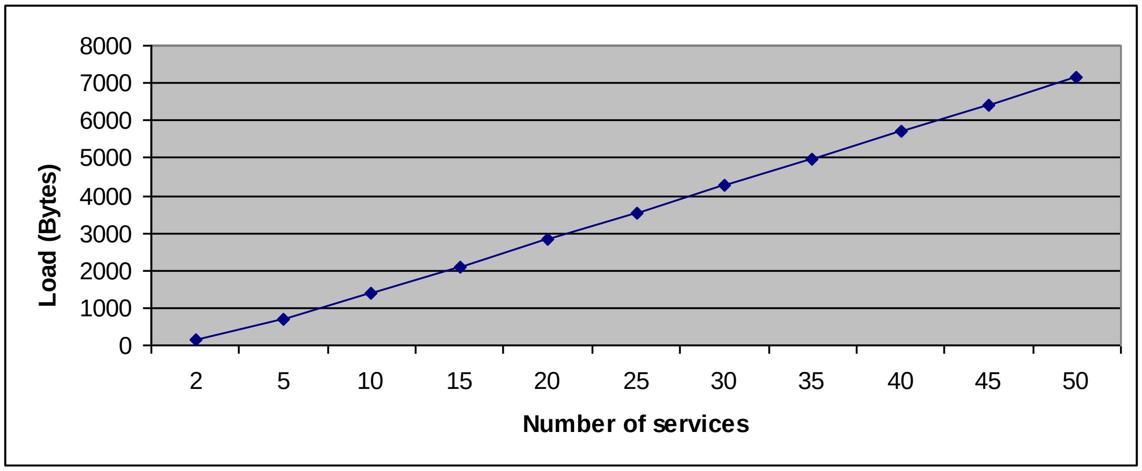


Figure 8.36: Total bytes exchanged in the network – different services

8.3 Summary

In this chapter we have presented a proof-of-concept prototype and a performance evaluation for the integrated 3G/MANET architecture. The prototype was described and the performance results based on using the OPNET simulation tool have been presented and analyzed.

The main goal of the prototype is to show the feasibility of the solution using the existing mechanisms (e.g. PDP, HTTP and SIP). Since it is impossible for us the build a realistic 3G network in our laboratory, the prototype results have no significant importance. Therefore, we have decided to make a simulation as a thorough validation.

OPNET was our simulation tool of choice. Different scenarios were considered and the impact of the number of end-users and the number of parallel services on the system performance was studied.

Regarding the impact of the number of end-users on the performance, the simulation results show that the individual packet delay is not affected by the number of users. In fact, for each entity the average delay of the received packets remains acceptable overall, even when the number of end-users reaches 50. We noticed that at the application server entity the average delay is over 1 second after 45 end-users.

Furthermore, the average end-to-end delay of the main packets (i.e. service-run, EU-join, EU-invoke, and SSE-register) is also acceptable, except for the EU join. After 40 end-users the delay becomes higher than 1.2 seconds for EU-join. Finally, the network load obviously increases linearly with the number of end-users. However, it does not get very high: less than 300 packets exchanged with 50 end-users.

As for the impact of the number of parallel services, the delays are a little bit higher than in the previous scenario. This is due to the fact that more packets are exchanged and more management is required. However, at each entity, the average packet delay remains reasonable. It only exceeds 1 second at the application server entity when there are more than 40 services in parallel. For the end-to-end delays, the performance was not as good as for the previous scenario. Indeed, the delays were usually more than 2 seconds. Furthermore, the load introduced in the network is higher when the number of parallel services increases: 1600 packets exchanged at a 50 service-level.

However, the performance results can be used as a guideline to enhance the architecture and improve the performance. For instance, clustering can easily improve the performance. Using different, SGWs will also have a positive impact on the delays. The presence of multiple ASs that can provide services will bring down the delays and provide better performance.

CHAPTER 9: Conclusions and future work

This chapter starts with a summary of the main contributions of our work. Then, it pinpoints and discusses the remaining issues and directions for future work.

9.1 Summary of contributions

Many research communities have made MANETs their central area. Fundamentally, the key element of a network is the services provided over it. Service provisioning in MANETs is very challenging because of these networks' characteristics. We have addressed the service provisioning aspects at the application level. The SIP servlets framework was our starting point. Furthermore, we have investigated service provisioning issues in two different MANET environments: standalone MANETs and Multihop Cellular Networks (MCNs). MCNs are an example of integrated 3G/MANET networks.

However, our solutions have their limitations. In fact, the business model has several roles therefore an instance of each role should be available in the network for service provisioning. Furthermore, the stand alone MANET overlay network solution has the limitation of relying on a given scheme of the distributed SSE and the performance evaluation does not consider different type of services and different instances of the IMS entities.

The main contributions of this thesis are that it has:

- **Identified issues related to service provisioning in MANETs:** We have compared the characteristics and constraints of MANET environments to the service provisioning process. Indeed, it is difficult to build a service provisioning architecture for networks where no pre-established infrastructure is permitted. Service provisioning becomes very challenging due to node mobility and topology changes. Furthermore, the service provisioning framework and related mechanisms and protocols should take into account the limited resources of MANET devices and the wireless links properties. Finally, the service provisioning solution for integrated 3G/MANET systems should be advantageous for network operators. Each of these aspects contributes to making our work a very stimulating and fruitful research area.
- **Derived requirements and contributed a critical review of related work:** We have derived the requirements for the different solution proposed in this thesis. We subsequently reviewed the related work accordingly. The requirements for the overall architecture for stand alone MANETs were elaborated first, followed by the requirements for each component of the architecture: the business model and the related publication and discovery mechanism and service execution architecture, and the corresponding communication mechanism. Furthermore, the general and specific requirements for service provisioning architecture for integrated 3G/MANET networks were derived.

  The related work was reviewed and we concluded that none of the existing solutions meets all our requirements. However, we opt for an evolutionary strategy and so we have considered the SIP servlets framework as a starting point to build an adequate service provisioning architecture for MANETs.

- **Proposed a business model for service provisioning in standalone MANETs:** We proposed a novel business model that we have refined to take into account all of the characteristics of MANETs. The proposed business model has been designed specifically for MANETs, with no central entity and with lightweight roles. The business model roles offer lightweight functions which can be provided by individuals using small devices, so that individuals as well as organizations can take part in the proposed business model. Furthermore, the roles and the functions they provide can be discovered dynamically as needed, which makes the business model flexible and well-adapted to MANETs.
- **Proposed an overlay network for service invocation and execution in standalone MANETs:** Based on the business model and a distributed scheme of the SIP servlets engine we have proposed an overlay network. The overlay nodes are the SIP servlets' engine components. The overlay network's main function is to address the highly dynamic aspect of MANETs. Several nodes coexist and cooperate to provide a service invocation and execution environment. The overlay network is autonomous and fault-tolerant. Indeed, it comes with a proposed protocol and procedures for self-organization and self-recovery.
- **Proposed service provisioning architecture for integrated 3G/MANETs:** We have proposed a novel architecture for service provisioning for Multihop Cellular Networks (MCNs). An exhaustive set of solutions were investigated, and high-level architectural alternatives for MCNs were discussed. Each alternative proposes a

solution for providing services by 3G and/or MANET service providers to 3G and/or MANET users. The advantages and drawbacks of each solution were analyzed. The most interesting alternative, from the network operator point of view, was then selected and a concrete architecture proposed. The architecture allows MANET users to access 3G services but the services are executed in the MANET network. A new functional entity was proposed, the Service GateWay (SGW), that operates in the middle. Furthermore, minimal enhancements are proposed for some existing 3G and MANET entities.

- **Implemented proof of concepts, formal validation and simulation:** We have implemented two prototypes as proof of concepts for the proposed business model and for the integrated 3G/MANET architecture. The prototypes demonstrate the feasibility of the solution. The results of the business model prototype show that the delays are acceptable and that they can be enhanced. We modeled the overlay network protocol and defined its correctness requirements using the PROMELA/SPIN tool. We were able to simulate the protocol behaviour using SPIN and checked its correctness via PROMELA. Finally, we used OPNET, a powerful simulation tool, to measure the performance of the integrated 3G/MANET architecture. Metrics were presented, different scenarios defined and the performance results have been discussed. Globally, the architecture performance is acceptable. Furthermore, the architecture scales well in terms of the number of end-users and the number of parallel provided services.

9.2 Future work

In this section we present some directions for future work. Indeed, several issues are still unsolved and constitute a good opportunity for future research. We have classified some of them into four work categories: future directions related to the overall architecture, future work related to the overlay network architecture, research directions related to the integrated 3G/MANET architecture, and potential work in the areas of implementation and performance.

9.2.1 Overall architecture

We have based our architecture on the SIP servlets framework. It will be interesting to investigate different service provisioning frameworks to evaluate how they can be used in developing new architectures. Parlay or web services are attractive candidates for building a service provisioning architecture. Indeed, both Parlay and web services are variants of the IMS application servers, and therefore could be used as design frameworks for an integrated 3G/MANET architecture. Stand-alone MANETs and MCNs could be targeted. A second consequent direction would be to identify, measure and compare the cost, performance and constraints of different architectures. Obviously, a basis for comparison would need to be established for the corresponding architectural alternatives.

Our work is based on a distributed SIP servlet engine. We have chosen a functional distribution of this engine. Other distribution schemes could be proposed, evaluated and compared. We believe that the overlay network can adapt to the new SIP servlets' engine distribution scheme. Two major criteria should be taken into consideration for the distribution. First, there must be loose coupling between the components to reduce the number of exchanged messages. Second, the engine should be broken into a few components to increase the probability of having all of them at the same moment in the MANET. Furthermore, since nodes move and leave frequently, the probability that one of the engine components will leave can be reduced. Therefore, the first goal is to design a new solution for the SIP servlet engine distribution. The second would then be to adjust the existing overlay network to the new distribution scheme. The new architecture could then be compared with our proposed one.

Finally, security is of the utmost importance for the overall architecture, as well as for each part of it, both for standalone and integrated architectures.

9.2.2 Overlay network architecture

In the proposed overlay network we have some nodes that are fully meshed (i.e. controllers and session repositories). This full-mesh connection has a cost in terms of resources and performance. Therefore, a potential research direction would be the use of clustering. For small networks, a flat cluster may be used while hierarchical clustering should be investigated for large networks.

Furthermore, in our solution, the same information is stored in several nodes. In fact, all the controllers store the information about the nodes in their areas of control and in the nodes of the other controllers. It is important that controllers get access to this information for recovery reasons. The same analysis is true for the session repositories that store sensitive data related to the ongoing sessions. One possible solution to release the storage resources would be to use some optimization techniques such as replication.

A survey of the replication techniques for MANETs is presented in [113]. A clustering replication [113] or a distributed shared memory [115] appear to be likely candidates.

In addition, the controllers' decision algorithm is based on certain characteristics: time to live and capacity. However, this can lead to a performance problem, since proximity is not considered. Indeed, a message may traverse several nodes before it reaches the selected controllers with the highest capacity and time to live, which is not the best use of resources. Therefore, another criterion should be added: proximity. A formula that makes a trade-off between these three criteria needs to be elaborated.

9.2.3 Integrated 3G/MANET architecture

One of the biggest challenges in an integrated 3G/MANET is the billing and charging system. Since in our proposal the service execution is done in the MANET portion, the question is, how is the charging performed? This is a central issue for the network operator. Offline charging systems could be used, for instance.

Another issue is related to service continuity. The proposed architecture brings a solution when all the users are in the MANET. However, what happens if the service has started and then a MANET user moves to the 3G portion? This user is not out of range and not disconnected but still cannot be reached by the MANET service execution environment. A possible solution is to have a process to frequently check end-users' locations. When the location changes from MANET to 3G, the service gateway will play the role of a relay between the end-user and the service execution environment.

Another potential research direction is to study and propose concrete architecture schemes for the other high-level architectural alternatives. A performance comparison could then be done. We strongly believe that each alternative should target a given network configuration and operator need. Therefore, a network operator may have several alternatives implemented in different sites. Switching from one alternative to another should also be investigated.

9.2.4 Implementation and performance

The first step, from the implementation point of view, is to integrate all the complete solution components and build the overall architecture. It will be interesting to see how the different solutions interact and what the performance of the global solution is. The integration of the MANET prototype with the overlay network and the integrated 3G/MANET may require some adjustments or enhancements. Therefore, code optimization is another issue to address. Indeed, implementing the solution using optimized code and calculations will improve the performance.

In addition, an important item is implementing and testing our architecture in larger networks. We used a one-cell 3G network, but it is essential to conduct an implementation with several cells and different configurations. Furthermore, the integrated 3G/MANET system we used is based on one service gateway, which would lead to problems with performance and scalability. Having multiple service gateways and multiple SIP servlet engines will be key to working with large-scale networks. The inter-communication in these contexts will need to be detailed.

References

[1]. Wakefield, Tony, Dave McNally, David Bowler, and Alan Mayne. *Introduction to Mobile Communications: Technology, Services, Markets*. Auerbach Publications. © 2007.

[2]. Karmakar, Gour, and Laurence S. Dooley (eds). *Mobile Multimedia Communications: Concepts, Applications, and Challenges*. IGI Publishing 2008.

[3]. WWRF, Technologies for the Wireless Future, R. Tafazolli (Ed.), Hoboken, NJ: Wiley, 2004.

[4]. George Aggelou, "*Mobile Ad Hoc Networks – From Wireless LANs to 4G Networks*", a book published by McGraw-Hill Companies, Inc., 2005.

[5]. Webb, William. *Wireless Communications: The Future*. John Wiley & Sons. © 2007.

[6]. Marco conti and silvia Giordano, "*Multihop ad-hoc networking: the theory*", IEEE Communications Magazine, April 2007, Volume 45, Issue 4, Page(s):78-86.

[7]. Ram Ramanathan, "*Antenna Beamforming and Power Control for Ad Hoc Networks*", Mobile Ad Hoc Networking, Wiley-IEEE Press, July 2004.

[8]. Sunil Kumar, Vineet S. Raghavan and Jing Deng, "*Medium Access Control protocols"* Issue 3, May 2006, Pages 326-358.

[9]. Elizabeth M. Belding-Royer, "*Routing Approaches in Mobile Ad Hoc Networks*", Mobile Ad Hoc Networking, Wiley-IEEE Press, July 2004.

[10]. S. Bah, R. Glitho and R. Dssouli, "A business model with LINDA- based broker for service provisioning in Mobile Ad Hoc Networks", 10th International Conference on Intelligence in Networks (ICIN 06), 29 May – 1st june 2006, Bordeaux, France.

[11]. S. Bah, A. Basel, R. Glitho, F. Khendek, R. Dssouli, "A SIP servlets framework for service provisioning in stand-alone Mobile Ad Hoc Networks", ICIN 2008, October 2008, Bordeaux, France.

[12]. S. Bah, R. Glitho and R. Dssouli, "An Overlay Network for a SIP Servlet-Based Service Execution Environment in Stand Alone MANETs", Second IFIP International Conference on New Technologies, Mobile and Security, NTMS'2008, November 2008, Tangier, Morocco.

[13]. S. Bah, R. Glitho and R. Dssouli, “SIP Servlets for Service Provisioning in Multihop Cellular Networks: High-Level Architectural Alternatives”, IEEE Consumer Communications & Networking Conference (CCNC 2008), Las Vegas. 10-12 January, 2008.

[14]. S. Bah, R. Glitho and R. Dssouli, “Provisioning Services in Multihop Cellular Networks When the End-Users are in the Mobile Ad-hoc Network portion”, IEEE International Workshop on Broadband Convergence Networks (BcN 2008) at IEEE Network Operations and Management Symposium (NOMS 2008), 7-11 April 2008, Salvador – Bahia, Brazil.

[15]. S. Bah, R. Glitho and R. Dssouli, “Service provisioning in Multihop Cellular Networks” submitted to IEEE Wireless communications magazine.

[16]. Xiang-Yang Li, Wireless Ad Hoc and Sensor networks: Theory and Applications, Cambridge University Press, 2008.

[17]. Mohapatra, P. and Krishnamurthy, S. AD HOC Networks: Technologies and Protocols. Springer-Verlag New York, Inc. 2005.

[18]. Prasad, Ramjee, and Luc Deneire, From WPANs to Personal Networks: Technologies and Applications, Artech House. © 2006.

[19]. Corson, M. S., and J. Macker, Mobile Ad hoc Networking (MANET), "Routing Protocol Performance Issues and Evaluation Considerations," Request For Comments 2501, Internet Engineering Task Force, January 1999.

[20]. J. Hubaux, J. Boudec, S. Giordano, and M. Hamdi, “The terminode project: Toward mobile ad-hoc WANs,” in Proc. MOMUC'99 San Diego, CA, 1999.

[21]. Jennifer J. –N. Liu and Imrich Chlamtac, “Mobile Ad hoc Networking with a view of 4G Wireless: Imperatives and Challenges”, Mobile Ad Hoc Networking, Wiley-IEEE press, July 2004.

[22]. J. Blau, “Wi-Fi Hotspot Networks Sprout Like Mushrooms,” IEEE Spectrum, pp.18-20. Sept. 2002.

[23]. R-S. Chang, W-Y. Chen, and Y-F. Wen. “Hybrid wireless network protocols”, IEEE Trans-actions on Vehicular Technology, 52(4):1099–1109, July 2003.

[24]. B. Liu, Z. Liu, and D. Towsley. “On the capacity of hybrid wireless networks”, In Proc. IEEE INFOCOM, volume 2, pages 1543–1552, San Francisco, CA, April, 2003.

[25]. P. T. Olivier Dousse and M. Hasler, “Connectivity in ad-hoc and hybrid networks,” in Proc. IEEE Infocom, 2002.

[26]. Bin Xie, Anup Kumar and D. P. Agrawal, “ Security Issues in an Integrated Cellular Network –WLAN and MANET,” Edited book entitled, Wireless Ad Hoc Networking: Personal-Area, Local-Area, and Sensory-Area Networks, CRC Press, Dec. 2006.

[27]. A. Bria, F. Gessler, O. Queseth, R. Stridh, M. Unbehaun, J. Wu, J. Zander, and M. Flament,“4th-Generation Wireless Infrastructures: Scenarios and Research Challenges,” IEEE Personal Communications, December 2001.

[28]. C. Polits, T. Oda, S. Dixit, A. Sieder, H. Lanch, M. Smirnov, S. Uskela, and R. Tafazolli, “Cooperative Networks for the Future Wireless World,” IEEE Communications Magazine, September 2004.

[29]. Bing (ed), Benny, Emerging Technologies in Wireless LANs: Theory, Design, and Deployment, Cambridge University Press, 2008.

[30]. 3GPP TS 23.002, “Network architecture” October 2005.

[31]. R. Pichna, “Interworking Architecture Between 3GPP and WLAN System,” IEEE Communication Magazine, Nov. 2003.

[32]. F. G. Marquez, M. G. Rodriguez, T. R. Valladates, T. De Miguel and L. A. Galindo, “Interworking of IP Multimedia Core Networks between 3GPP and WLAN,” IEEE Wireless Communications, June 2005.

[33]. 3GPP TS 23.234, “3GPP system to WLAN Interworking” V.6.5.0 Jun 2005.

[34]. M. Buddhikot, G. Chandranmenon, S. Han, Y. Lee, S. Miller, and L. Salgarelli, “Integration of 802.11 and Third-Generation Wireless Data Networks,” in Proc. of IEEE Conference on Computer Communications (INFOCOM’03), San Francisco, CA, April 2003.

[35]. M. Lott, M. Siebert, S. Bonjour, D. von Hugo, and M. Weckerle, “Interworking of WLAN and 3G systems,” IEE Proc. of Communications, vol. 151, no. 5, pp. 507-513, October 2004.

[36]. A. Salkintzis, “Interworking Techniques and Architectures for WLAN/3G Integration Toward 4G Mobile Data Networks,” IEEE Wireless Communications, vol. 11, no. 3, pp. 50-61, June 2004.

[37]. H. Wu, C. Qiao, S. De, and O. Tonguz, “Integrated Cellular and Ad Hoc Relaying Systems: iCAR”, IEEE JSAC, vol.19, 2001, pp. 2105-15.

[38]. H.Luo et al., “UCAN: A Unified Cellular and Ad-Hoc network Architecture”, proc. ACM Mobicom, Sept. 2003.

[39]. D. Cavalcanti, C. M. Cordeiro, D. P. Agrawal, B. Xie, and A. Kumar, “Issues in Integrating Cellular Networks, WLANs, and MANETs: A Futuristic Heterogeneous Wireless Network,” in IEEE Wireless Communications Magazine, Special Issue on Toward Seamless Internetworking of Wireless LAN and Cellular Networks, June 2005.

[40]. C. Fu, F.Khendek, R. Glitho, , “Signaling for multimedia conferencing in 4G: the case of integrated 3G/MANETs”, IEEE Communications Magazine, Aug. 2006.

[41]. D. Ben Khedher, R. Glitho and R. Dssouli , Media Handling for Multimedia Conferencing in Multihop Cellular Networks, IEEE Network Magazine, Forthcoming, 2009.

[42]. I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges," Elsevier Ad Hoc Networks Journal, vol. 1, no. 1, pp. 13-64, July 2003.

[43]. R. Shollmeier, I. Gruber and M. Finkenzeller, “Routing in Mobile Ad-Hoc networks and Peer-to-Peer Networks, a Comparison”, Inter. Workshop on Peer-to-Peer Computing, Pisa, Italy, May 2002.

[44]. Juneja, Girish, Blake Dournaee, Joe Natoli, and Steve Birkel, Service Oriented Architecture Demystified, Intel Press. © 2007.

[45]. Randall Perrey , Mark Lycett, Service-Oriented Architecture, Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), p.116, January 27-31, 2003.

[46]. C. Abarca et al., TINA Consortium, service architecture specifications, version 5.0, 1997, available from www.tinac.org

[47]. Berndt, H., Hamada, T., Graubmann, P., TINA: Its Achievements and its Future Directions, *IEEE Communications Surveys & Tutorials*, Vol. 1, 2000.

[48]. Roch H. Glitho, A Mobile Agent Based-Service Architecture for Internet Telephony, PhD dissertation, The Royal Institute of Technology, Stockholm, 2002.

[49]. Berndt, H., Graubmann, P., and Wakano, M., Service Specification Concepts in TINA-C. In Proceedings of the Second international Conference on intelligence in Broadband Services and Networks: Towards A Pan-European Telecommunication Service infrastructure, September 1994.

[50]. Rohan Sen, Radu Handorean, Gruia-Catalian Roman, and Christopher Gill, "Service Oriented Computing Imperatives in Ad Hoc Wireless Settings," Technical Report WU-CSE-2004-05, Washington University, Department of Computer Science, St. Louis, Missouri.

[51]. Erik Christensen et al., Web Services Description Language (WSDL) 1.1 specification, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl.

[52]. Shafer, S.M., Smith, H.J., Linder, J.C., The power of business models, Business Horizons 48, pp 199–207. 2005.

[53]. Magretta, J., *Why business models matter, Harvard business review*, Cambridge, MA: Harvard Business School Publishing Corporation, 2002.

[54]. TINA-C 4.0 business model Specification, "TINA business model and reference points", 1997. http://www.tinac.com

[55]. Gonzalo Camarillo, Miguel-Angel Garcia-Martin, the 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds, Second Edition, John Wiley & Sons, 2006.

[56]. J. Rosenberg et al., "SIP: Session Initiation Protocol," IETF RFC 3261, June 2002.

[57]. J. Rosenberg and H. Schulzrinne, Session Initiation Protocol (SIP): Locating SIP Servers. RFC 3263, Internet Engineering Task Force, June 2002.

[58]. R.Fielding et al. Hypertext Transfer Protocol HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[59]. WAP Forum. WAP architecture, Recommendation WAP architecture, Wireless Application Protocol Forum, July 2001.

[60]. 3rd Generation Partenership Project, Technical Specification Group Core Network and Terminals; "Conferencing Using the IP Multimedia (IM); Core Network (CN) Subsystem"; 3GPPP TS 24.147 v7.2.0; Sept. 2006.

[61]. Rajagopalan, Suresh, Java Servlet Programming Bible, John Wiley & Sons, 2002.

[62]. Dany Coward, Yutaka Yushida, “Java Servlet Specification, Version 2.4”; Release: November 24, 2003.

[63]. A. Kristensen, “The SIP Servlet API specifications, Version 1.0”, February. 2003.

[64]. José S. Lucas, SIP servlets, White paper, GEMINI Consortium, September 2003.

[65]. M. Munoz and C. Garcıa-Rubio, “A New Model for Service and Application Convergence in B3G/4G Networks”, IEEE Wireless Communications, 11(5):6–12, Oct. 2004.

[66]. S. Arbanowski et al. “I-centric Communications: Personalization, Ambient Awareness, and Adaptability for Future Mobile Services”, IEEE Communications Magazine, September 2004, pp. 63-69.

[67]. TINA Consortium web site: http://www.tinac.com/about/principles_of_tinac.htm, last accessed on April 2009.

[68]. Yu, C.-F., "Customer service provisioning in intelligent networks," *Network, IEEE*, vol.4, no.1, pp.25-28, Jan 1990.

[69]. Wireless Application Protocol Architecture Specification, version 12 July, 2001.

[70]. Sun MicroSystems, JAIN and Open Networks, white paper describing the positioning of the JAIN Application Programming Interfaces (APIs) within open network architectures, August 2003. http://java.sun.com/products/jain/JainAndOpenNetworks01.pdf. Last accessed on April 2009.

[71]. Parlay 6.0 Specification, Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview (Parlay 6), 2007. http://www.parlay.org

[72]. A.J. Moerdijk and L. Klostermann, “Opening the Networks with Parlay/OSA: Standards and Aspects Behind the APIs”, IEEE Network, pp. 58-64, May/June 2003.

[73]. H. Kreger, “Web Services Conceptual Architectures (WSCA 1.0)”, IBM Software Group, White Paper, May 2001.

[74]. F.Curbera et al., “Unraveling the Web services Web: An Introduction to SOAP, WSDL and UDDI”, IEEE Internet Computing, Vol. 6, No2, March-April 2002, pp. 86-93.

[75]. Adam Bosworth, "A Conversation with Adam Bosworth", ACM Queue vol. 1, no. 1 - March 2003.

[76]. SOAP specifications, http://www.w3.org/TR/soap last accessed on April 2009.

[77]. UDDI specifications, http://www.uddi.org last accessed on April 2009.

[78]. Hao-hua Chu, Chuang-wen You, Chao-ming Teng, Challenges: wireless Web service, Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on, 7-9 July 2004.

[79]. Satish Narayana S., Matthias Jarke , Wolfgang Prinz, Mobile Web Service Provisioning, Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services, February 19-25, 2006.

[80]. 3rd Generation Partnership Project, Technical Specifications Group; Services & Systems Aspects, IP Multimedia Subsystem (IMS), Stage 2, 3GPP TS 23.228 v8.7.0, December 2008.

[81]. Ericsson white paper, Introduction to IMS, 284 23-8123 Uen Rev A., March 2007. http://www.ericsson.com/technology/whitepapers/8123_Intro_to_ims_a.pdf last accessed on April 2009.

[82]. Y. Inoue, D. Guha, and H. Berndt, "The TINA consortium," IEEE Commun. Mag., vol. 36, Sept. 1998.

[83]. M. Mampaey and A. Couturier, "Using TINA Concepts for IN Evolution," IEEE Commun. Mag., June 2000.

[84]. Mang Li; Schieferdecker, I.; Rennoch, A., "Testing the TINA retailer reference point," *Autonomous Decentralized Systems, 1999. Integration of Heterogeneous Systems. Proceedings. The Fourth International Symposium on* , vol., no., pp.268-275, 1999.

[85]. W3C working group, "Web Services Architecture", W3C working group note, February 2004. http://www.w3.org/TR/ws-arch/#discovery_approaches. Last accessed on April 2009.

[86]. Li Li and Louise Lamont, "A Lightweight Service Discovery Mechanism for Mobile Ad Hoc Pervasive Environment Using Cross-layer Design", IEEE

Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops).

[87]. Christopher N. Ververidis and George C. Polyzos, "Routing Layer Support for Service Discovery in Mobile Ad Hoc Networks", IEEE Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops) 2005.

[88]. V. Park and J., "Macker. Anycast routing for mobile services", Proc. of Conference on Information Sciences and Systems (CISS), March 1999.

[89]. Tyan, J.; Mahmoud, Q.H., "A network layer based architecture for service discovery in mobile ad hoc networks", IEEE Electrical and Computer Engineering, 2004. Canadian Conference on volume 3, 2-5 May 2004 Page(s):1379 - 1384 Vol.3.

[90]. Erik Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71-80, July/Aug. 1999.

[91]. Salutation Consortium, Salutation Architecture Specification Version 2.0 Part 2", June 1999.

[92]. Sun Microsystems Jini Specification, " DJ- Jini discovery and join specification", version 3.0, 2005.

[93]. B. Miller, Home Networking with Universal Plug and Play, *lEEE Communication. Mag.*, vol. 39, no. 12, Dec. 2001.

[94]. F. Sailhan, V. Issarny: Scalable Service Discovery for MANET, Proceedings of the 3rd IEEE Int'l Conf.on Pervasive Computing and Communications (2005).

[95]. U. C. Kozat, L. Tassiulas: Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues, Ad Hoc Networks, 2, (2004).

[96]. Sumi Helal, Nitin Desai, Varum Verma and Choonhwa Lee, "Konark – A Service Discovery and Delivery Protocol for Ad-hoc Networks", Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, March 2003.

[97]. M. Nidd, "Service Discovery in DEAPspace," IEEE Personal Communications, August 2001.

[98]. C. Campo et al. "PDP and GSDL: A New Service Discovery Middleware to Support Spontaneous Interactions in Pervasive Systems", *In IEEE Middleware Support for Pervasive Computing (PerWare 2005) at the 3rd IEEE Conference on Pervasive Computing (PerCom 2005)* (March 8, 2005).

[99]. Radu Handorean and Gruia-Catalin Roman, "Secure service provision in ad-hoc networks," in Proceedings of The First International Conference on Service Oriented Computing (ICSOC 03), Springer Verlag, Ed., 2003.

[100]. David Gelernter, "Generative communication in Linda", ACM Transactions on Programming Languages and Systems, 7(1):80{112}, January 1985.

[101]. Sun Microsystems, JAIN SLEE (JSLEE) 1.1 Specification, Final Release. 2008.

[102]. Sun Microsystems, JXTA v2.0 protocols specifications, 2007. https://jxta-spec.dev.java.net/JXTAProtocols.pdf. Last accessed on April 2009.

[103]. Roch H. Glitho, "Engineering Value Added Services in Next Generation Networks: Issues, Concepts and Principles", Full day tutorial at IEEE Global Communications Conference, San Francisco, USA, 2003.

[104]. Amy L. Murphy and Gruia-Catalin Roman, Lime: A Middleware for Physical and Logical Mobility, Technical Report WUCS-00-05, Washington University in St Louis, MO, USA February 2000.

[105]. E. K. Lua et al, "A survey and comparison of peer-to-peer overlay network schemes", IEEE Communication Surveys & Tutorials, Second Quarter 2005, volume 7, N.2.

[106]. Dhafer Ben Kheder, Roch Glitho, and Rachida Dssouli, A Novel Overlay-Based Failure Detection Architecture for MANET Applications, In the Proceedings of the IEEE International Conference on Networks (ICON2007), Adelaide, South Australia, 19th - 21st November 2007.

[107]. Khlifi, H., Agarwal, A., Gregoire, J.-C., "A framework to use SIP in ad-hoc networks", Proceedings of IEEE CCECE 2003, 4-7 May, 2003.

[108]. N. Banerjee, A. Acharya, and S. K. Das. "Peer-to-peer sip-based services over wireless ad hoc networks", In BROADWIM: Broadband Wireless Multimedia Workshop, Oct. 2004.

[109]. S. Leggio, J. Manner, A. Hulkkonen, and K. Raatikainen. “Session initiation protocol deployment in ad-hoc networks: a decentralized approach”, In 2nd IWWAN, May, 2005.

[110]. M. El Barachi, R. Glitho, R. Dssouli, “Context-aware signaling for call differentiation in IMS-based 3G networks”, Proceedings of IEEE Symposium on Computers and Communications(ISCC’07), Aveiro Portugal 1st – 4th July 2007.

[111]. G. J. Holzmann, Design and Validation of Computer Protocols. New Jersey: Prentice-Hall, Inc. 1991.

[112]. OPNET Simulator. OPNET Technologies, Inc., http://www.opnet.com.

[113]. Padmanabhan, P., Gruenwald, L., Vallur, A., and Atiquzzaman, M., “A survey of data replication techniques for mobile ad hoc network databases”. The International Journal on Very Large Data Bases.Aug. 2008.

[114]. Takahiro Hara, Sanjay K. Madria, "Data Replication for Improving Data Accessibility in Ad Hoc Networks," IEEE Transactions on Mobile Computing, vol. 5, no. 11, Nov. 2006.

[115]. Duong, H. H. and Demeure, I. 2008, “Data sharing over mobile ad hoc networks”, In Proceedings of the 8th international Conference on New Technologies in Distributed Systems (Lyon, France, June 23 - 27, 2008). NOTERE '08. ACM, New York, NY.