ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI
INFORMATICA – SCIENZA E INGEGNERIA

TESI DI LAUREA

in
Sistemi Mobili

Connectivity management and dynamic context grouping in mobile heterogenous systems

CANDIDATO:
Denis Billi

RELATORE:
Chiar.mo Prof. Paolo Bellavista

CORRELATORI:
Prof. Theo Kanter
Prof. Ing. Antonio Corradi

Anno Accademico 2014/2015
Sessione II

Abstract

Negli ultimi 10 anni il numero di dispositivi mobili è cresciuto rapidamente. Ogni persona porta quotidianamente con se almeno due dispositivi ed i ricercatori sono convinti che in un futuro non troppo lontano questo numero crescerà almeno fino a dieci. Inoltre, tutti questi dispositivi stanno diventando via via sempre più integrati nella nostra vita rispetto al passato, pertanto la quantità di dati scambiati crescerà con il miglioramento dello stile di vita delle persone apportato da questi dispositivi. Questo è quanto i ricercatori chiamano l'*Internet delle cose*. In futuro vi saranno oltre 60 miliardi di nodi mobili e l'infrastruttura corrente non è pronta per ospitare un tale scambio di pacchetti. Sono stati proposti miglioramenti ai protocolli, come MobileIP e HIP, in modo da facilitare lo scambio dei pacchetti in mobilità, tuttavia nessuno di questi ha ancora raggiunto quel grado di ottimizzazione tale da permettere un cambio di rotta. Negli ultimi anni, i ricercatori della Mid Sweden University hanno creato a tal scopo il MediaSense Framework. Inizialmente, il framework si basava su Chord per il routing dei pacchetti e su DHT per la persistenza. Con l'introduzione di P-Grid, le performance di lookup nel trie sono migliorate fino ad arrivare a $0.5 * log(N)$, dove N è il numero di nodi nella rete. Tale risultato può essere ulteriormente migliorato con delle ottimizzazioni sulla gestione dei dati dei nodi, ad esempio tramite raggruppamenti logici. Inoltre, siccome lo scopo è l'utilizzo in mobilità, è necessario provvedere ad un substrato che permetta la gestione di più connessioni diverse. A tal scopo, è stato scelto SCTP in quanto si tratta di uno dei protocolli più promettenti per il futuro.

Abstract

In the last 10 years the number of mobile devices has grown rapidly. Each person usually brings at least two personal devices and researchers says that in a near future this number could raise up to ten devices per person. Moreover, all the devices are becoming more integrated to our life than in the past, therefore the amount of data exchanged increases accordingly to the improvement of people's lifestyle. This is what researchers call *Internet of Things*. Thus, in the future there will be more than 60 billions of nodes and the current infrastructure is not ready to keep track of all the exchanges of data between them. Therefore, infrastructure improvements have been proposed in the last years, like MobileIP and HIP in order to facilitate the exchange of packets in mobility, however none of them have been optimized for the purpose. In the last years, researchers from Mid Sweden University created The MediaSense Framework. Initially, this framework was based on the Chord protocol in order to route packets in a big network, but the most important change has been the introduction of PGrids in order to create the Overlay and the persistence. Thanks to this technology, a lookup in the trie takes up to $0.5 * log(N)$, where N is the total number of nodes in the network. This result could be improved by further optimizations on the management of the nodes, for example by the dynamic creation of groups of nodes. Moreover, since the nodes move, an underlaying support for connectivity management is needed. SCTP has been selected as one of the most promising upcoming standards for simultaneous multiple connection's management.

Contents

LIST OF FIGURES

ACRONYMS

AR	Access Router is the main router of the HMIP network.
ASCONF	Address Configuration Change Chunk is used in SCTP to communicate to the remote endpoint one of the configuration change requests.
BS	Base Station is the unit of the mobile communication system.
BU	Binding Update packet is a packet sent from the mobile node to his Home Agent (HA) with its mobility informations in MobileIP protocol.
CoA	Care-of Address is the current mobile address of a mobile node in a MobileIP network. It's usually associated with its HA address to correctly deliver the packets.
DAR	Dynamic Address Reconfiguration is an extension to SCTP that will allow an SCTP stack to dynamically add an IP address to an SCTP association, dynamically delete an IP address from an SCTP association, and to request to set the primary address the peer will use when sending to an endpoint.
DCXP	Distributed Context eXchange Protocol is an XML-based application level P2P protocol which offers reliable communication among nodes that have joined the P2P network.
DDCA	Distributed Dynamic Clustering Algorithm is the algorithm designed in this study and is capable of dynamically grouping mobile nodes inside an heterogeneous P2P network.
DHT	Distributed Hash Table is a class of a decentralized distributed system that provides a lookup service similar to a hash table.
FA	Foreign Agent is the last deliverer of packets for a mobile node in MobileIP networks. It also takes care of current mobile node location.
GSM	Global System for Mobile Communications is the most popular mobile phones' technology in the world.
HA	Home Agent is the main packets' router for a mobile node in MobileIP networks.

HI	Host Identifier is a name in the Host Identity namespace.
HIP	Host Identity Protocol is an evolution of the MobileIP protocol that simplify the network management thanks to the use of some hierarchical network layers of MobileIP routers.
HIT	Host Identifier Tag is a 128-bit static globally unique cryptographic SHA-1 hash over the HI.
HMIP	Hierarchical MobileIP is an evolution of the MobileIP protocol that simplify the network management thanks to the use of some hierarchical network layers of MobileIP routers.
HMIPv6	Hierarchical MobileIPv6 is an evolution of the Hierarchical MobileIP (HMIP) for the IPv6 protocol.
IETF	Internet Engineering Task Force develops and promotes Internet standards, cooperating closely with the W3C and ISO/IEC standards bodies and dealing in particular with standards of the TCP/IP and Internet protocol suite.
LSI	Local Scope Identifier is a 32-bit or a 128-bit local representation of HI.
MAP	Mobile Access Point has the same role of Mobile Switching Center (MSC) on GSM networks. Look at MSC for more.
MH	Mobile Host is the mobile device that moves with the user.
MNS	Mobile Naming System is the naming system that resolves a resource's identifier in its network addresses.
MS	Mobile Station is the same as Mobile Host (MH).
MSC	Mobile Switching Center is the center of management for devices' handoff.
SCTP	Stream Control Transmission Protocol is a Transport Layer protocol, serving in a similar role to the popular protocols TCP and UDP. It provides some of the same service features of both: it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP.
TDD	Test driven development is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards.
UCI	Universal Context Identifier is the identifier used in DCXP to identify resources like URI does in WWW.
URI	Uniform Resource Identifier is a string of characters used to identify a name or a resource on the Internet.

UMTS	Universal Mobile Telecommunications System is a third generation mobile cellular technology for networks based on the GSM standard.
WMN	Wireless Mesh Network is a communications network made up of radio nodes organized in a mesh topology.
WSN	Wireless Sensor Networks are the networks based on spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants and to cooperatively pass their data through the network to a main location.

INTRODUCTION

The first aim of this thesis work is related to the mobility problem. In fact, mobility is one of the most fascinating subjects that the modern research is trying to handle. Our life is increasingly becoming a life related to the mobility, where all our operations in the world are subordinated to the fact that we move. Let explain this assumption with some examples: firstly, in the last twenty years the portability (that means both size reduction and the battery life's growth) of devices has determined a massive diffusion of mobile systems like smartphones and tablets, nevertheless sensors of any type and size that can help us in a more efficient management of our life. The scenario in which we live is where we have the efficiency and the performance above all, in which a lesser use of bandwidth significantly improves battery use, and that means a numerically increase of the communications that we can have during a day. Secondly, it's now clear that the really next future networks will be populated by a large amount of devices: if we think that reasonably in the near future everyone will own ten devices (both wearables and simple devices) and that in the world there are 6 billions of people, then there will be 50 billions of total devices in the world, all of these potentially able to move and to change their current connectivity. Thirdly, another problem that is related to the mobility, is due to the fact that a person continually moves during a day, and that means that he cannot use always the same connection to access his personal resources. Fourthly, these devices and sensors are related in some way, and the relations between them are what the researchers call *context.* Everything that can be potentially helpful to me in this particular moment is part of my current context. Fifthly, it's now clear that the current infrastructure is unable to sustain a such growing network of relationships, so the researchers have studied some solutions that can both improve the existing networks.

The aim of the thesis is to study the P2P networks in order to let the devices to organize themselves only using their relations' information. Following this purpose, the evidence that a node (that can be a device, a sensor or whatelse) can belong to one or more persons, moves the focus of the work to a new level of abstraction, in which the nodes are clustered in several groups depending only on their context information rather than their phisical location.

In the first chapter is explained the beginning point of our study, and especially what are the major problems in the mobile connectivity research, what researchers are trying to focus and to solve. The second chapter explains what is the current state of art about network protocols that researchers have developed in the last 20 years in order to solve the problem of mobility and how is organized the MediaSense Framework. The third chapter gives a deep explanation of the Distributed Dynamic Clustering Algorithm (DDCA) algorithm, the methodology we followed and the design's guidelines we used in order to develop our solution. All the possible viable ways that we found and which one of them we will pursue in the next sections will be discussed in this chapter. In the fourth chapter, is finally explained the approach we pursued about the project's design, the technical analysis and requirements in order to obtain a good final working solution. Here we focus on the project's implementation, especially in the technical parts that needs further explanations. Lastly, in this chapter will ber reported the tests made and the results obtained.

CHAPTER

ONE

PROBLEM STATEMENT AND SYSTEM MODEL

People's life is increasingly becoming related to mobility, where all the operations in the world are subordinated to the fact that people move. The first aim of this thesis work is to find a way to manage a large amount of mobile devices basing on the available context information. Through the grouping of the entities, it is possible to manage them in an easier way. However, current infrastructure is not ready to manage such situations of high mobility, both in number and in connection changes.

This can be explained by the following assumptions: firstly, in the last twenty years, the portability (that means both size reduction and the battery life's growth) of devices has determined a massive diffusion of mobile systems. Smartphones, tablets, nevertheless sensors of any type and size can help people in a more efficient management of their life.

Moreover, efficiency and performance of all the internal components (both hardware and software) can highly determine a lesser use of bandwidth and a significantly improved battery life. This means a numerically increase of the communications that we can have during a day.

Secondly, future Internet will be populated by a large amounts of devices: reasonably, in the future everyone will own approximately ten devices and knowing that in the world there are 6 billions people, then there will be more than 60 billions of total devices in the world, all of these potentially able to move and to change their current connectivity.

Thirdly, another problem that is related to mobility, is due to the fact that a person continually moves during a day, and that means that he cannot always use the same connection to access his personal resources.

Fourthly, these devices and sensors are related in some way, and the relations between them are what the researchers call *context*. Everything that can be potentially helpful to me in this particular time is part of my current context.

Finally, current infrastructure is unable to sustain such a growing network of relationships, so the researchers have studied some solutions that can both improve the existing networks as well as create a totally new meaning of network that is based on the relations rather than on location.

Letting devices to self-organize in one network based on the object relations is the main idea followed by the research team in the ITM Department at the Mid Sweden University. Whether a node of the network (that can be a device, a sensor or whatever else) can belong to one or more people, no one in the literature already found a solution in order to permit the subdivision in categories of the devices currently present in the network. Specifically, there are not current protocols able to organize the nodes of a network in order to simplify the work of the intermediate nodes in packaging, routing and delivery of the information.

1.1 Mobile computing problems

Very soon, people will need to access their own information in a easier manner from everywhere. However, this is not possible with the current infrastructure.

Related to the movement's issue, there are several questions without an answer. Moving from one location to another, actually needs a manual change of the connectivity made by the user, without any kind of policy or intelligent system that helps him. Maybe, it could be possible that in one location there are two types of connections and the user would like to use both of them simultaneously. Researchers call this kind of action with the name of *multi-homing*.

Moreover, actual technologies permit people to access their data always and anywhere: such as cloud-storage solutions like Google Docs, Dropbox and iCloud. This kind of actions could be improved with the introduction of real-time interaction and context-sensitive technologies.

This can be exemplified by the following situation: a man is returning home after a long trip and when he takes the taxi from the airport, his home's heating reacts by turning on the radiators if the house is too cold. Another example could be the direct interaction with a car navigator that chooses the best way to the destination using the information given by the other nearby cars in transit. All these kind of interactions are being part of the so called *context-aware mobile-computing.* Researchers have been really interested in these studies since the late 80's, but today there are no protocols that works well.

1.2 High-level problem statement

Being able to access data from the move is an interesting research field. However, without any kind of organization of the resources, it could be a really difficult and time consuming task. Moreover, this can be impossible when the infrastructure doesn't give a proper help.

This can be exemplified with the following scenario: Jason wants to access his own home computer and wants to stream a video to his mobile device. Currently he is connected to his home's *access point* and when he will leave, he will disconnect from the access point and afterward reconnect to the 3G connection that is available outside to continue the conversation.

This operation will create an interruption in the stream service for Jason, and this happens because there is no concept of maintaining the *session* on connectivity change. The chronological history of the conversation between two entities (like two network's endpoints), known as the connection's state, is called *session*. As well known, the most used transport protocols are TCP and UDP. The former takes packet loss as network congestion and transmission gracefully degrades, so if there is a disconnection in a wireless mobile environment all the connection information is lost [9]. The latter instead doesn't provide a packet retransmission mechanism, so it's impossible to understand if a packet correctly arrived to destination [7]. Thus, unless there isn't something that takes care of keeping on the session's information, it is impossible to grant a high-standard connection's quality. When we face this problem in wireless environments, it is clear that all depends on our current location, because in every place there can be more or less types of connectivities. In the next chapter we will focus our attention on the current solutions for this issue.

To allow a finer explanation of the problem, researchers has defined two kind of *location management*: in case of *location registration* is up to the mobile node to inform the infrastructure that it is moving to another carrier and only then the infrastructure acts accordingly. On the contrary, in case if *location search* is up to the whole infrastructure to locate, instant by instant, the mobile node's movements [33].

Clearly, in our scenario the first solution is the less expensive of the two, because without any kind of infrastructure improvements it is unthinkable to keep track of 50 billions of devices' movements at the same time. So, the first but summarily intuitive solution for the mobility problem is based on the location registration approach, but we will discuss this later in section 1.2.2 on page 22.

The second aim of this work is to help people on finding their personal things in the big pot that the *Internet of Things* is. When we say *Internet of Things* we are talking about of the near future of the Internet, where all our personal devices and also all our home's appliances, will be connected to the Internet and we can access and interact with them.

Current infrastructure is not suitable for this kind of operations, because every single network have its own addressing routes and parameters, with all problems derived by NAT networks that hides underlaying resources from the outside. What we need is something that allows us to make an ontological research in the *Internet of Things* within a determined time bound, maybe in which we can subdivide resources and services in groups like "Personal" or "Friends" and so on, and in which we can share personal resources with friends or colleagues.

This can be exemplified by the following scenario: Jason is at home and he is enjoying his ultrafast Internet connection and his friend wants to connect to the Internet too. Today we should give him the network credentials, but in the *Internet of Things* with the addition of context's information, these credentials can be automatically sent to the Jason's friend's device because the system knows that he is a friend and he can access his friend's Internet connection.

1.2.1 Scope

Today's connectivity has become really easy, we can go everywhere and with our smartphones we are able to access our personal files. However, current infrastructure was thought and created more than forty years ago, so it is not reliable for these kind of operations. What most IT researchers studied in the last twenty years is the possibility to enhance network performance with external information that comes from the user's locations and habits. This information is called *context*. With such this information, we can simplify how the resources are organized in a really big network, because it is up to the resources themselves to understand which context they belong to, that means less work for the infrastructure. Thus, the more context information we have, the better it is for the final computation in terms of speed and quality.

Dynamic grouping Regarding the problem of the relationships between resources, we will focus this problem better in the next chapter, but it is important to understand which is its role in this study. In the last section there is an a example in which Jason and his friend are at Jason's home and the second decides to connect to the Internet. As stated before, having some context information about Jason friends and with the possibility to have some policy system, it would be really simple to handle the network credential so that every Jason's friend can access to his home's connection. Moreover, context information are useful also in the case we need to change our underlaying connectivity.

Security In the last example, Jason wants to share his network credentials with his friend. This is important to fulfill some security considerations:

1 Jason doesn't want that his friends do something illegal with his Internet connection;

2 Jason should tune up the system in a way that only when he is at home a friend could access the Internet;

3 A Jason's friend could be able to access the connection during the night because Jason he is actually at home, sleeping;

4 Jason would need a system to keep all the information about his friends.

Mobile connectivity MobileIP (section 2.2 on page 28) is one of the solution that researchers have found to manage the connectivity handover between various location. It relies only on the decisions taken by the device itself, and it takes too much time to do that. In the *Internet of Things*, this is not robust. With some context information (like the user's location or the signal strength, for example), infrastructure would take a great benefit, because it could use these information to figure out the exact moment when to switch the connections.

Lastly, to make it possible to access resources from everywhere and in a robust way, devices should continuously benefit of the connection. Obviously, this is possible unless there are no connection available in the place the user is, but that is only an extremely case that lies outside our studies. A typical scenario for this problem, is the case in which we have both wireless and 3G connections available. Jason is moving from his home (that has got a WiFi connection) to outside (where is available a 3G connection). He wants to continuously watch a movie, or have a nice conversation with a friend on Skype. There are four possible scenarios:

1 He manually switches off the wireless network and switches on the 3G antenna;

2 He uses an automatic switcher like the Locale application for Android;

3 The infrastructure helps him with a low level transport protocol, like MobileIP;

4 He uses a *middleware* that in an automatically and graceful way switches between one connection to the other, like in figure 3.2 on page 50.

What we can say, is that all these things are completely right, but all of them lies outside our competences and therefor they will not be mentioned in this study.

1.2.2 Verifiable goals

The main goal of this work is to fulfill the requirements about the management of a large amount of mobile nodes. The algorithm should be scalable for groups of moderate sizes. Moreover, the handover between connections should be faster (or within the same time range) than current solutions, and for the purposes of this thesis, all the work will be focused on the last two points of the previous section. This is really important, because in the future *Internet of Things*, connection maintenance should be done without user interaction.

Summarizing, these are the main purposes of the study:

1 Having a grouping algorithm that scales linearly or logarithmically for multiple participants;

2 Fulfill connectivity handover requirements for the resources;

3 Understand if it is possible to handle the switching faster or at least in the same times of current solutions, that take in the worst case 400 milliseconds;

1.2.3 Contributions

The *MediaSense Framework* with *PGrid*s and its source code was contributed by and is property of *Mid Sweden University.* This project has contributed by adding functionality to the existing framework in order to create groups dynamically on a real network basing on context information. The grouping layer is independent from the framework itself, therefore is possible to use it with any kind of implementation of the latter. Functionalities and calculations for the simulations have been provided by the *Context Cluster Simulator*, created by Victor Kardeby.

In the next sections, we provided a short introduction about the mobility issue and his specific definitions like *terminal mobility*, *user mobility* and *logical mobility.* This is important in order to understand what is the *context-awareness*, of which we'll talk about in the next chapter. Lastly, we will try to give a fast explanation of *multi-homing* and how to handle it and why it could be useful for managing the handover.

1.3 Context-awareness

> Context is the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user.

This explanation of *context awareness* written by researchers of the Department of Computer Science of the Dartmouth College (reference [6]) clarifies that in the future networks will be crucial the adaptability based on such information. In the future *Internet of Things*, everything will be a potential important information for devices. As well-explained in [27], context can be subdivided into three categories: *computing context*, *user context* and *physical context.* Also, using each of them across a time span, it's possible to obtain a *context history*, that could be used in various situations like, for example, the ones mentioned in the first chapter. Thus, [6] divides context into two different parts: the first is the *active context*, the one that "influences the behaviors of an application"; the second is the *passive context*, the one that is "relevant but not critical to an application". In this particular scenario, in the first caseis the application itself that adapts to the situation discovering new information and acting as consequence. In the second case it makes persistent the new context to an interested entity.
Moreover, more specifically in mobile networks, will be fundamental the location where the information are stored: both locally and remotely are

practicable ways; the substantial difference is who will manage such information in order to decide what actions to take in order to give a service to the user. These strategies are typically related to complex systems like power grids, in which the complexities and the difficulties in controlling the systems forced the researchers to formulate new solutions. We will examine further this topic in the next chapter.

1.4 Context-based dynamic grouping

Managing a large number of mobile devices could be really hard. The aim of this thesis work is to understand whether exists a way to programmatically differentiate different nodes basing on their context information in a distributed environment. Such operation can be also viewed as a distributed clustering option.

In the last years, several solution have been proposed in the literature in order to solve both the problem of the grouping and the handover between different networks. In 2011, some researchers from several universities have suggested to combine the concept of network virtualization with Wireless Mesh Network (WMN) technology [18]. This can be done using context information to autonomously build logical, virtual networks overlays on top of physical networks. In this scheme, is possible to model networking relevant user contextlike QoS, security etc., and let the users to connect to the *Virtual Network* that best fits their context requirements. Moreover, they decided to select mobility out of the set of available characteristics. They also concluded that a better predictive algorithm for the handover between different networks could enhance significantly their work.

1.5 The problem of mobility

Mobility related issues can be divided into three different groups: *terminal mobility* is when wireless devices are able to physically move along more connections and this requires both infrastructure and terminal enhancements. Another form of mobility is called *user mobility*, and it happens when a user relocates and switches devices. In this case, the user should be able to use the same services independently of the device itself. Moreover, the notion of mobility can be seen at an higher abstraction's level, where all the computation's information like user interests, context and other attributes does not corresponds exactly to the physical world. This kind of mobility is known as *logical mobility* [33].

Terminal mobility Wireless networking permits mobile users to benefit of the Internet connectivity and all the services related. Obviously, user moves must be managed from one connectivity covered area to another one, and the user should be able to switch at the same (for example from WiFi to 3G or from WiFi to Bluetooth). Moreover, active connections should be left active independently of the user's moves: it's what in GSM/UMTS connections is called *handoff*. On IP-based networks would be fine if it was possible to maintain the same IP address over more different networks, but in this section we'll understand that it's not possible because an IP address is dependent on the network itself, and so it tends to change on different networks and in case of handoff, new packets should be delivered to the new address.

User mobility With the massive beginning of devices's spreading, could be useful that all the user's information are being maintained by the infrastructure itself rather than by devices. In fact, many users would manage their data (and their work) independently by the device that they are using, both on smartphones, tablet, PC's and so on [35]. There are a lot of openstudies about this kind of mobility that are based both on individual model and group model, let's think about all the fashionable applications like Dropbox, iCloud and so on that try to solve this problem in an easiest manner.

Logical mobility When we talk about logical mobility (also know as session/resource mobility) we should think about what are the relevant information that don't directly relies on the physical world. In fact, current mobile applications are developed as monolithic architectures that are not suitable for the context in which they work and is up to developers to decide at design time the possible uses of their applications. It would be useful for these applications to perform intelligent decisions based on the current context information, personal preferences or local sensors' values (that would necessarily being constantly updated). In thisscenario, a mobile application should be a self-organising system adaptable on the requirements' changes [40].

In this work, we'll try to take the best of all these aspects of mobility to make the system suitable for all scenarios that we explained in the introduction.

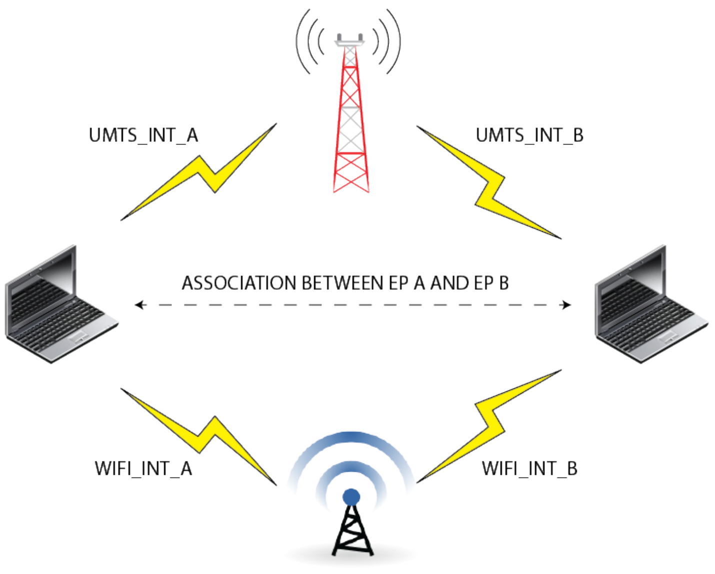


Figure 1.1: A typical association with multi-homed endpoints

1.6 Multi-homing

Today, this action is possible only with the use of *middlewares*, that creates a layer between the network and the application, and then is the middleware itself that manages all the underlaying networks. However, this kind of operations, done at the ISO application's layer is very expensive in terms of network performance because the throughput between the physical layer and the application layer is to high, would be necessary a low level management of this kind of operations.

In fact, the current studies that will bring all the world to the adoption of the new IPv6 standard had being conducted with the intent of permit multi-homing operations. The reasons that stands behind the multi-homing are reasonably two:

1. To contemporary handle more connections at the same time
2. To help with operations like connectivity handover

The second reason is why we would need this technology available for our purposes. Without multihoming is it too expensive to handle connectivity handovers. However, until it won't be possible to have this technology at lower layers, we must use high level solutionsat the best of their potentials.

CHAPTER

TWO

RELATED TECHNOLOGIES

A first important step is to know the state of art of the current research literature. The following sections will try to explain howthe current mobile infrastructure works.

Global System for Mobile Communications (GSM) technology is one of the most reliable technologies that makes use of the handover. In order to obtain some additional information that can be useful for the development of our studies, in the fourth section we will try to understand what there is behind the GSM handover algorithm. Even though GSM has been developed thirty years ago, inside it there are some interesting notions about handover's management, even if in this case we talk about homogeneous networks that are different to heterogeneous ones of which we are dealing. After that, we'll give an overview of some technologies that tried to manage the handoff on heterogeneous networks like MobileIP, HMIP and Host Identity Protocol (HIP), which respectively give the complete management of the handover to the device itself and then is the infrastructure that implements some features in order to provide the correct deliveries of the packets directed to the device from the correspondent hosts. Next to these, section 2.4 will talk about the Stream Control Transmission Protocol (SCTP) protocol, one of the first protocols proposed by Internet Engineering Task Force (IETF) capable of managing multiple streams at once. Finally, we'll have a look on the MediaSense Framework, developed by the IT department of the Mid Sweden University.

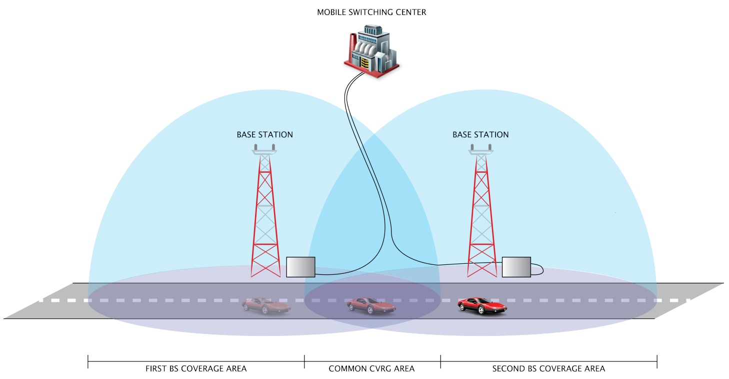


Figure 2.1: Typical real-working scenario of GSM handover

2.1 Handoff management on GSM networks

GSM communications's spread has become much popular in the last twenty years, during which they reached a level of maturity and versatility that permit us to benefit a lot of different services like mobile telephony, SMS, Internet and so on. What often stays behind the scenes in this technology is the whole infrastructure, but it's what allows to the communication to work and that is represented on figure 2.1. The base idea of the GSM infrastructure is related to the geographical subdivisionin hundreds of subareas, all adjacent each others and called *cells*. All these cells are served by a radio base, called Base Station (BS), each one with his own radio frequency for not interfere with the other near cells. In the GSM system, the infrastructure continuously checks some variables like received field strength, the signal quality and the distance between the mobile station (MS, that is the GSM device) and the base station (BS, that is the signal repeater). All this measurements serve as a basis for correct handover decisions. The basic algorithm processes the measured samples in the base station system, wherethe recent 32 measured values are stored and with the results obtained is the infrastructure itself (in this case, the base station) that decides if a handover is required.

In figure 2.2 on the facing page is possible to see the typical signal trends during a GSM handover and we can understand how the infrastructure can decide when the handover is required [19].

2.2 MobileIP

In section 1.2 on page 19 we talked about how to manage the locationof the nodes belonging to an heterogeneous connectivity network. We summarily concluded that the only manner to know the location of one node of the

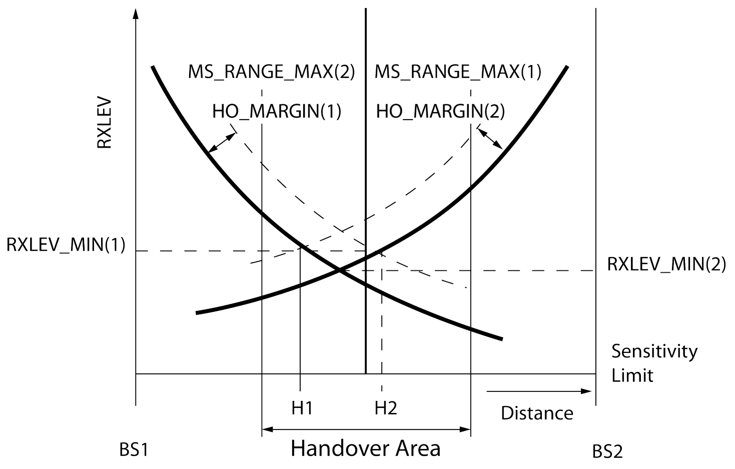


Figure 2.2: This graph shows the typical trends of the measured values in GSM networks during handover

network should rely on a direct feedback by the node itself. Just to understand how it works, let's think about our home's address. If we move to another address, we can tell our old doorkeeper to forward all our new mail to the new doorkeeper who will delivers it directly to us. This is exactly what MobileIP does.

> MobileIP is a standard proposed by a working group within the IETF in 2002 and was designed to solve this problem allowing the mobile node to use two IP addresses: a fixed *home address* and a *care-of address* that changes at each new point of attachment. The first is static and is used to identify TCP connections. The second changes at each new point of attachment and can be thought of as the mobile node's topologically significant address; it indicates the network number and thus identifies themobile node's point of attachment with respect to the network topology. The HA makes it appear that the mobile node is continually able to receive data on its home network, where MobileIP requires the existence of a network node known as the HA. Whenever the mobile node is not attached to its home network (and is therefore attached to what is termed a *foreign agent*), the home agent gets all the packets destined for the mobile node and arranges to deliver them to the mobile node's current point of attachment. Whenever the mobile node moves, it registers its new care-of address with its home agent [24].

In this way, the home agent delivers the packet from the home network to the care-of address using the mechanism known as *redirection.* Once arrived at care-of address, the reverse transformation is applied so that the packet appears to have the mobile node's home address as the destination IP address, which will be processed properly by TCP layer (or whatever higher level that logically receives it from the mobile node's IP).

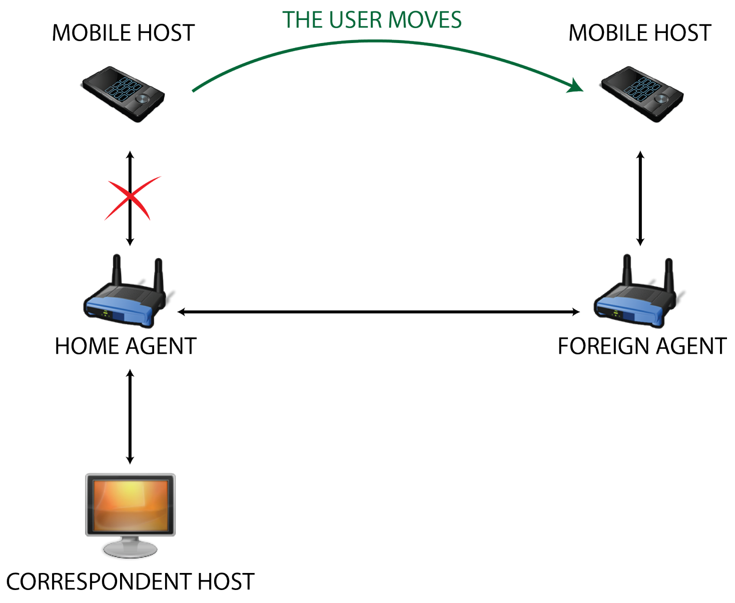


Figure 2.3: Working scheme of MobileIP's first draft

It has been proved that one of the biggest problem related to MobileIP is the potential retransmission timeout. For example, the MobileIP handoff latency can be up to to 3 seconds, which can easily force the TCP to timeout [29]. Even though this latency can be reduced using link-layer enhancements, some experiments show that TCP can still timeout during a link-layer handoff [41]. This handover latency is caused by standard MobileIPv6 procedures, such as movement detection, new Care-of Address (CoA) configuration and Binding Update packet (BU). It's usually unacceptable to real-time traffic, such as VoIP, and it may cause critical packet loss. Over the years, in order to reduce the handoff latency of a Mobile Host, have been proposed numerous solutions that can be classified intotwo groups: *the first* aims to reduce the network registration time by using a hierarchical network management structure, while*the second* attempts to reduce the address resolution time through address pre-configuration [28].

Secondly, MobileIP is architecturally based on re-using a single name space, that is the IP address space, for both the HA address and the CoA. This creates two potentially drawbacks: the first is an unwanted and undesirable dependency on the constantreachability of the home address. In other words, the mobile host reachability depends on the home agent. The second is the aliasing of multiple IP addresses caused by the weak dependency between the home address and the CoA. It's confusing to understand if oneis merely an alias for the other or a completely different host's identifier [21].

2.2.1 Hierarchical MobileIP

The HMIP concept is introduced mainly in order to minimize the signaling latency for a BU that's sent from the MH to the Home Agent. Just to explain quickly the mechanism, in HMIP packets destined to the MH are routed to the MH via the HA and a MAP. The MH has only to inform the Mobile Access Point (MAP) of the new CoA after handoffs within a MAP domain. This kind of architecture can reduce BU signaling latency since it will take less time to update the MAP than it does the distant HA. Moreover, to prevent packet loss during the disconnection, some extensions of HMIP like HMIP-B [23], FHMIPv6 [28], [15], [26] have been proposed. However, these solutions takes in the worst case up to 400 milliseconds delay in the disconnection. Just to understand what's behind this protocol, it's necessary to talk about the infrastructure behind MobileIP that allows its operation and why researchers are trying to focus on this extension rather than optimize MobileIP directly. In fact, MobileIP exhibits several problems regarding the duration of handoff and the scalability of the registration procedure. As said in the first chapter, our scenario foresee more than fifty billions of devices that change networks quite frequently, and it would mean an high amount of signaling overhead on the HA as well as on the networks. So, there is no efficient support to *micro-mobility*. The principle of Hierarchical MobileIPv6 (HMIPv6) is the MH that connects for the first time to the domain registers only one time to its HA with the address to the MAP as CoA. This is called Home Registration. Any further movement of the MH inside the domain will be transparent to the HA. Each time the MH changes the Foreign Agent (FA), it will perform a *Regional Registration* containing the new local CoA. Moreover, HMIP supports amultilevels hierarchy of MAPs where each MAP must maintain an entry in its *visitor's list* for each MH connectedto an Access Router (AR) of the hierarchy [26].

2.3 HIP

Today's Internet is based almost on two main namespaces: DNS names and IP addresses. The first has the responsibility of converting a logical name given by users to a physical IP address, helping all the infrastructure to

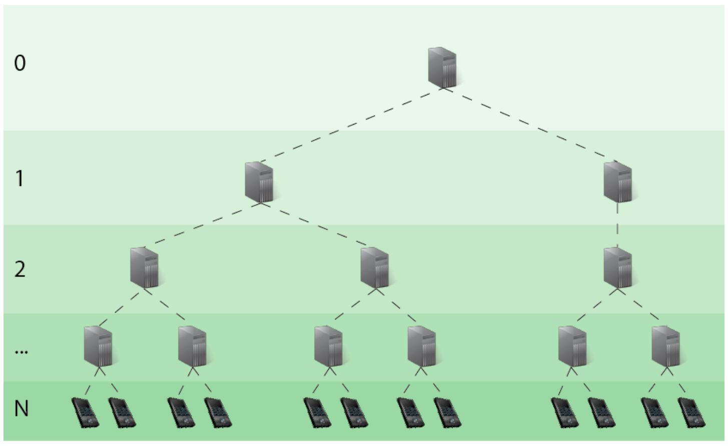


Figure 2.4: Architecture of Multi-level HMIP

correctly deliver all packages. However, IP address namespace describes both the host topological location in the network, and the host identity. As said before, this cause problems when the host has to change its IP address due to e.g. mobility. The location information changes, but it should not affect the identity information of the host. HIP introduces a separation between the *host identity* and *location identity*, where the IP address remains the locator, while a new namespace is introduced for host identifiers. This can be possible introducing a new layer between the *Transport Layer* and the *Networking Layer* of the TCP/IP stack, as shown in figure 2.5 on the facing page. Applications may use both Host Identifier Tag (HIT) addresses and Local Scope Identifier (LSI) addresses, that are respectively an *SHA-1* hash of Host Identifier (HI) and a 32-bit or 128-bit local representation of HI meant for IPv4 or IPv6 basedapplications.

The newly introduced layer hides IP addresses from the layers above, that means that applications are bound to sockets that consists of the HIT/LSI and port pair regardless of any IP address the host is using in the Networking layer.

HIP Multi-homing The indirection layer implemented in HIP provides the functionality of multi-homing due to the factthat this layer adds a new namespace (HI) to which the upper layer protocols (TCP, UDP, ...) are bounded. The IP still have the functionality of being the host's locator, and in order to be reached again by its peers, they have to be notified of the address' change [2].

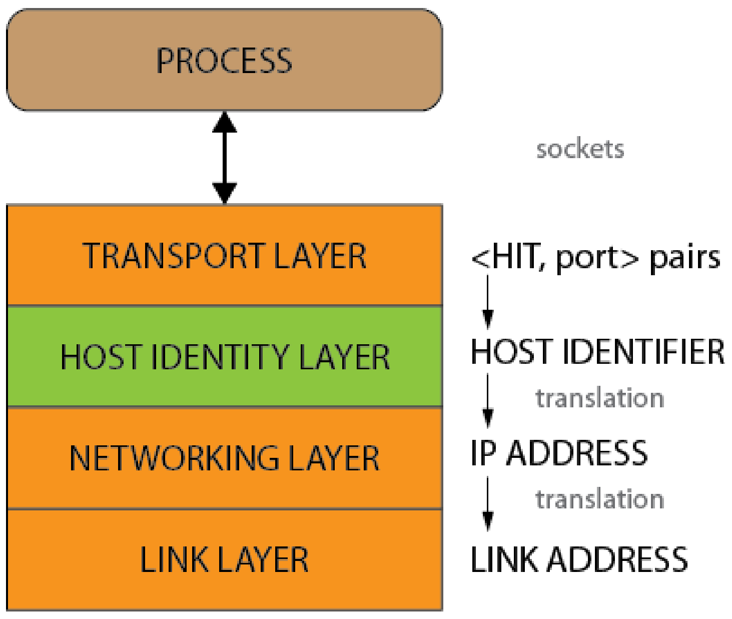


Figure 2.5: HIP stack

2.4 SCTP

SCTP is a general purpose transport layer protocol and had been released by IETF for standard in 2000. There are many features which make SCTP powerful. As said before, nowadays there are several wireless technologies like 802.11, Bluetooth, 3G and so on. Every single technology has got its own specifications and its working environmentbut no one supports the possibility of passing the state of a communication to the others. If we think to mobility, we know that WLANoffers high data rate but low mobility support, while Universal Mobile Telecommunications System (UMTS) offers higher mobility but with lower data rate. It's the same with Bluetooth, today at version 4 of the standard, that offers high data rate but in very limited space and it's perfect for small spacesand Wireless Sensor Networks (WSN), and the same is with Zigbee. In the future, a combination of these technologies will permit the user not to concern about which connection use in one specific location and how to connect to another user or resource, because the typical use of a device is about what task to do and not the way to communicate to the network. However, concurrent transmission using multiple-paths between multi-homed hosts has started to be studied almost recently in the literature, especially issues related to congestion control mechanism when striping data at the transport layer. It's important to know that a transport layer protocol that supports multi-homing usually consists of two components: a *data-striping* algorithm and a *data delivery* mechanism. The former isresponsible for distributing data to different paths according to certain *QoS* requirements, and the latter is responsible for data transmission and reception plus congestion control mechanism [39]. In such this way, SCTP can really help in the creation of this future. In fact, it inherits the congestion control and flow control schemes from TCP, and moreover it covers and gives a real and reliable solution to some specific problems like *multihoming*, *multistreaming*, partial reliability extension, selective acknowledgments and heartbeat messages. Finally, it can monitor the network situations and maintain reliable data transmission without causing any interferences to TCP. Thus, it's clear that SCTP offers many attractive new characteristics.

Multistreaming The purpose of SCTP is to connect two end points (that can be two users, or a user and a remote resource), with up to 216 different streams. Each stream is completely independent, and that means that it's not up to the developersto manage each connection. As explained in [8], this is really important because many TCP connections can lead to the problem of *slow start*, which can be interpreted as a congestion problem.

Multihoming One of the widely discussed feature of SCTP is multihoming. It enables an end user to utilize multiple IP addresses of different network interface cards simultaneously as shown in figure 1.1 on page 26. The general case in usingSCTP with multi-homing is the possibility to offer a reliable connection between two hosts. In fact, generally SCTP multi-homing uses one stream-connection as data transmission's path, and the others as backup paths, retransmissions and heartbeat messages. In such this way, the scheme is robust to faults of the streams.

Soft handover SCTP was developed to take full advantage of multihomed hosts to provide a fast failover and association survivability in the face of hardware failures. Moreover, many modern computers allow hot-pluggable network cards, and that means that in order to take advantage of this new configuration, the transport association must be restarted. For many fault-tolerant applications this restart is considered an outage and is undesirable. That's why researchers from *Network Working Group* proposed in 2007 an extension for the SCTP protocol called Dynamic Address Reconfiguration (DAR). This extension is nowadays a proposed standard with the name of *RFC 5061*. Thus, it would mean a great innovation to SCTP to contain an extension like DAR,that can be useful to adapt its applications to soft handover. In fact, the DAR enables the SCTP end users to add or deletebackup IP addresses within an existing association through a new data-chunk called Address Configuration Change Chunk (ASCONF). These chunks are useful to communicate to the remote endpoint one of the change requests that *must* be acknowledged.

Assuming that *gE*s are the end-points and that *Addr(gE)* represents the list of all network interface addresses of *gE*, below are reported the rules for address manipulation as described in *RFC 5061*.

```
1. An address can be added to a generalized endpoint
   gE only if this address is not an addressof a
   different generalized endpoint with the same port
   number.

2. An address can be added to an association A with
   generalized endpoint gE if it has been added
   to the generalized endpoint gE first.  This
   means that the address must be an element of
   Addr(gE) first and then it can become an element
   of Addr(A, gE). But this isnot necessary.  If the
   association does not allow there configuration of
   the addresses only Addr(gE) can be modified.

3. An address can be deleted from an association A
   with generalized endpoint gE as long as Addr(A,
   gE) stays non-empty.

4. An address can be deleted from an generalized
   endpoint gE only if it has been removed from all
   associations having gE as a generalized endpoint.
```

These rules simply make sure that the rules for the endpoints and associations given above are always fulfilled [30].

2.5 The MediaSense Framework

In the future *Internet of Things*, applications and services will needto make decisions about service deliveries based on the *context information* that will be collected from the environment like sensor values (that means physical data that needs to be interpreted in some ways), user preferences and habits and much more. All nodes of the future Internet infrastructure can freely participate in the decisions' process. This is possible only thanks to the useof multiple layers of abstraction that enforce the logic independence between them [34].

Specifically, the MediaSense Framework creates an abstraction layer upon each peer currently active in the network that hides the underlaying network

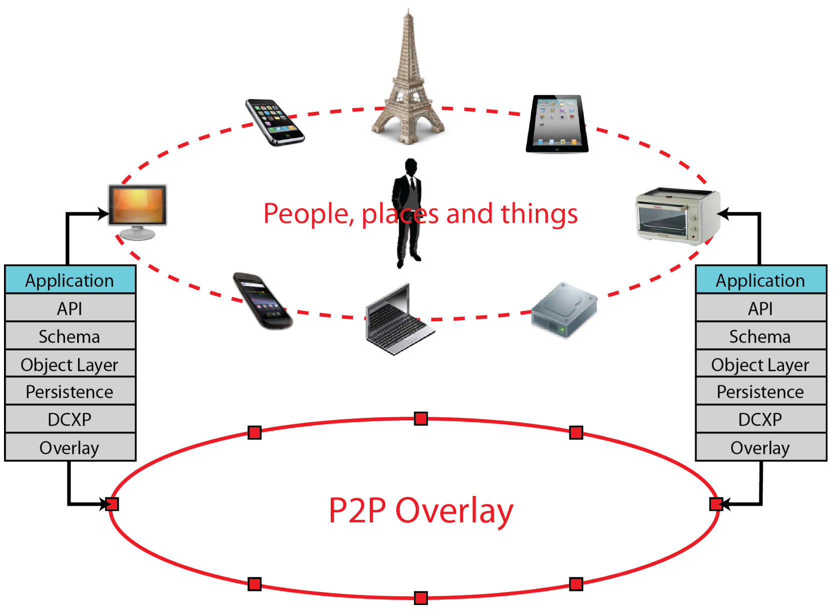


Figure 2.6: TheMediaSense Framework

structure and protocols in order to give to the device the idea of being part of a huge structured and distributed network, based on the services offered by each peer. This can be achieved thanks to the modularization of the framework, that is subdivided into seven different layers. Since the MediaSense Framework was born as the creation of a simple abstraction layer, initially it was based on a unique and monolitic solution, and the current research is trying to enhance its functionalities subdividing it into different layers completely independent each others. In Figure **??** on the facing page it is possible to see the current architecture of the framework itself. It is important to notice that currently it is divided into four different layers: the **Sensor Layer**, the **Application Layer**, the **Add-in Layer** and finally the **DCXP Layer**.

Sensor Layer Basically, the Sensor Layer is used by each type of devices and sensors that exposes a service in the structured Overlay. Specifically, every kind of information that can be given through the network, such as sensor values, IP address,ontological information (like the context aware services) or others, is kept and served consequently to the incoming requests, by this layer.

Application Layer In order to give a common interface to all the devices and sensors, the Application Layer exposes a list of public APIs.

Platform Figure.png

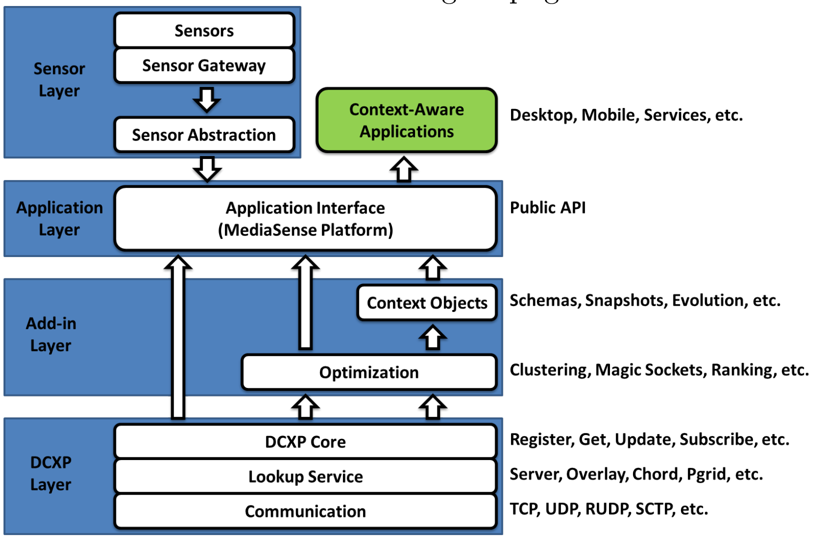

Sensor Layer
Sensors
Sensor Gateway
Sensor Abstraction
Context-Aware Applications
Desktop, Mobile, Services, etc.
Application Layer
Application Interface (MediaSense Platform)
Public API
Add-in Layer
Context Objects
Schemas, Snapshots, Evolution, etc.
Optimization
Clustering, Magic Sockets, Ranking, etc.
DCXP Layer
DCXP Core
Register, Get, Update, Subscribe, etc.
Lookup Service
Server, Overlay, Chord, Pgrid, etc.
Communication
TCP, UDP, RUDP, SCTP, etc.

Figure 2.7: Current MediaSense Framework Architecture

Add-in Layer The Add-in Layer contains all the appropriate routines for the optimization and the handling of specific parts of the framework. Currently, researchers are working in order to move this layer in the position above, to let this layer to bemore generic than now.

2.5.1 DCXP Layer

Context information in the MediaSense Framework are exchanged thanks to the use of theDistributed Context eXchange Protocol (DCXP), that provides the real-time exchange of information between hosts that are connected each others in a virtual P2P distributed space [12]. The DCXP has been created to permit to every device on the Internet to share context information joining as a node of the Overlay P2P network. DCXP is based on the exchange of context information using XML files. The DCXP protocol offers publish/subscribe abilities which are used to enable context exchange in real-time, but it presents also another interesting feature. Every device should register itself in the *Overlay* and is identified by a unique *Universal Context ID*, also known as Universal Context Identifier (UCI), that has the same meaning of the Uniform Resource Identifier (URI) in the World Wide Web . Fully qualified UCI can be seen as the following one:

```
dcxp://user[:password]@domain[/path[?options]]
```

Mapping between UCIs and source addresses can be done utilizing both Distributed Hash Table (DHT) and *P-Grids*. The advantage of the former isthat entries can be found in $O(log(n))$, meanwhile it takes only $O(0.5 * log(n))$ for the latter [12] [13] [34].

One of the most interesting extensions of DCXP, is related to the *Sensor Socket Module*. These sockets are treated exactlyas normal TCP-sockets and so are bounded to an IP address of the devices. When a node needs to create a connection with another one, it requests the *Overlay* to resolve the key's IP address (as a normal DNS system works) and then it can open an underlying TCP connection to the resolved IP/port pair [13].

It is really important to notice that actually the DCXP layer in the form of the MediaSense Framework itself is a completely passive system. Former assertion logically infers that a peer is not able to directly communicate with another one, but is the Overlay that takes care of the communication and the only operation that is possible is based on the *querying* system. Currently, DCXP provides two different types of queries: *direct queries* and *range queries*.

Direct queries Direct queries are probably the most used ones. They are presented in the following form:

```
uci://user[:password]@domain/resourcekey
```

In fact, they directly return the value referred to a particular key in the Overlay. If for example the peer is interested in the current temperature of the room number 123 of the building 4, it could perform the following query:

```
uci://room123@building4/temperature
```

The returning value will be directly delivered to the interested peer from the Overlay itself.

Range queries Instead, range queries are the most interesting feature of the MediaSense Framework. They are presented in the same form of *direct queries* but with the surplus of two more details: a *lower bound value* and an *upper bound value*.

When the Overlay is queried by the peers, it performs a lookup in the entire tree in order to find out all the values of the correspondent key that are between the lower and the upper bound of the query. This is particularly important in case of *context searches*, because thanks to P-Grid's structure they are pretty fast and that's why they are interesting in terms of research on this field.

2.5.2 P-Grids

Current applications resources on the World Wide Web are organized in groups, and in each group are concentrateda relatively small amount of nodes. Each node must apply sophisticated load-balancing and fault-tolerance algorithms to provide continuous and reliable access [1]. The bandwidth must be increased in case of a huge services, and in order to avoid node's failures it is important to introduce caching and replication solutions. The Client/Server paradigm has always worked since now because it is specifically optimized for such networks. Many problems were exposed since the advent of the necessity to create structured networks in which different nodes were able to see and operate on different groups and sub-groups. Peer-to-peer systems offer an alternative to such traditional paradigms, because every node of the system acts as both client and server and provides partof the overall resources/information available from the system. This is possible due to the absence of a central coordination and no peer has a global view of the system. The P2P approach tries to achieve the goal of a global search functionality without using a central directory. This can be obtained in two ways:

Unstructured network The data is distributed randomlyover the peers and the broadcasting mechanism is unconstrained (no limits on the dimension of the network and the time of the packets);

Structured network The network is organized as a distributed and scalable index structure, where the search requests are supposed to be routed directly to the owners of the respective information.

In the first case, peers are completely independent and they can manage their own data since this approach is fully decentralized. Moreover, search predicatesare not limited in this case. However, there are relatively high costs in search and message exchanges that consequently brings additional transmission delays. Instead, the second approach is more efficient, but a distributed index requires some form of centralcoordination. In order to maintain the beneficial characteristics of both the structured approach and the unstructured networks, *P-Grid* were introduced in order to provide a solution that permits a complete decentralization and support for sub-networks. This is possible thanks to the replacement of any form of global coordination with randomization.

P-Grid is a peer-to-peer lookup system based on a virtual distributed search tree. Each node of the tree can handle only part of the overall tree, that in MediaSense creates the big system called *Overlay*. Searching in *P-Grid* is efficient and fast. As reported in the previous section, given n as the number of nodes (leaves of the tree), even for unbalanced trees the lookup time is $O(0.5 * log(n))$ (2.10 on the next page). It is based on randomized algorithms and local interactions, and peers are assumed to fail frequently and to be online with a very low probability. In Figure 2.8 is shown a simple P-Grid.

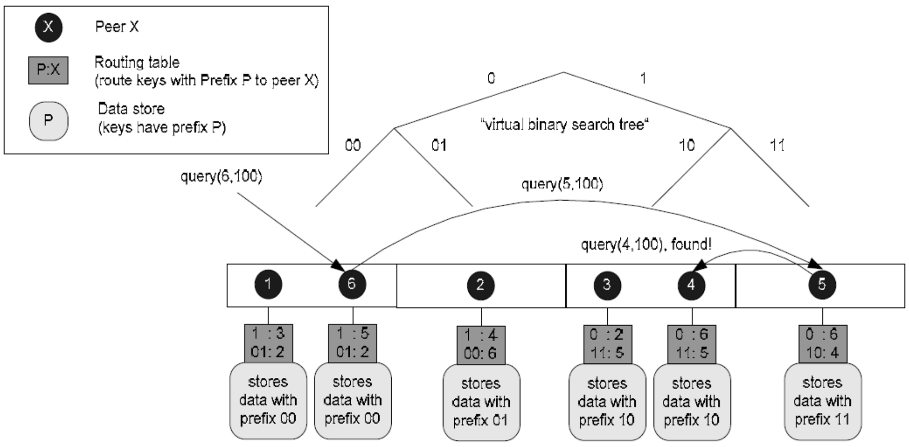

X
Peer X
P:X
Routing table
(route keys with Prefix P to peer X)
P
Data store
(keys have prefix P)
0
1
"virtual binary search tree"
00
01
10
11
query(6,100)
query(5,100)
query(4,100), found!
1
6
2
3
4
5
1 : 3
01: 2
1 : 5
01: 2
1 : 4
00: 6
0 : 2
11: 5
0 : 6
11: 5
0 : 6
10: 4
stores data with prefix 00
stores data with prefix 00
stores data with prefix 01
stores data with prefix 10
stores data with prefix 10
stores data with prefix 11

Figure 2.8: Example P-Grid

Every node knows its own position inside the tree, that is determined by its path, that is a binary bit string representing the subset of the tree's resource that the peer is responsible for. For each bit in its path, a peer stores a reference to at least one other peer that is responsible for the other side of the binary tree at that level. The lookup system is also based on the path. Thus, when a peer receives a binary query string it cannot satisfy, it forwards the query to the node that is "closer" to the result.The construction of the tree is based on a purely randomized algorithm that guarantees that each peer can always provide at least one path to one of the peers holding the queried information. For example, if Peer 1 forwards queries starting with 1 to Peer 3 (which is in Peer 1's routing table and whose path starts with 1), and Peer 3 can satisfy the query or forward it to another peer, depending on the next bits of the query. Unlike other DHT-based P2P systems, another salient feature of P-Grids is the separation of peer's identifier and peer's path: in fact, peer's path are determined by the tree itself, and it can change dynamically through negotiation with other peers as part of the network maintenance protocol. Thus, the construction of the tree is purely demanded to the nodes themselves, and it is totally decentralized and self-organizing, as it adapts automatically to a given distribution of data keys stored by the peers. Basically, the self-managing process of the tree begins from the peers. A node can locally decide whether to modify the routing infrastructure, if the present data justifies the modification, and this action changes the routing tables of the structure (virtually, the construction of the tree) in order to adapt to the data key distribution.

tree.png

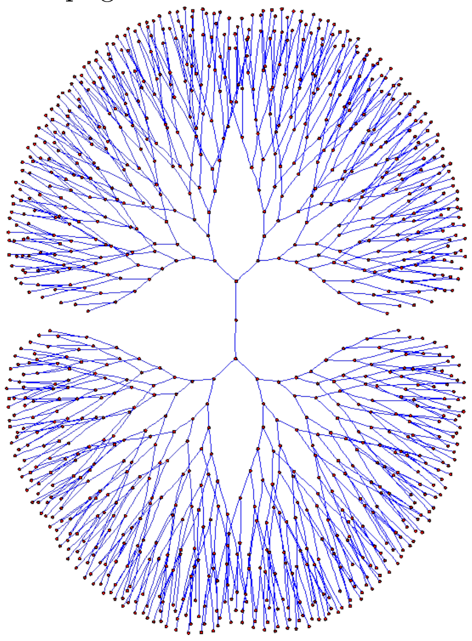


Figure 2.9: Example of a 1024 nodes binary tree

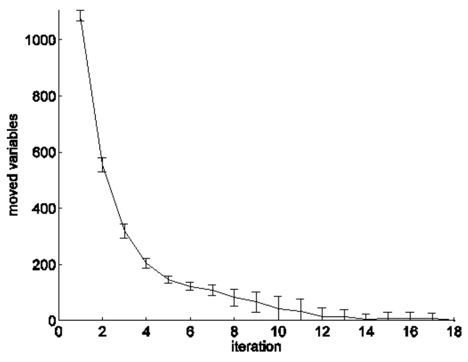


Figure 2.10: Reduction efforts of a binary tree

The P-Grid construction algorithm is based on purely randomized construction and guarantees that peer routing tables always provide at least one path from any peer receiving a request to one of the peers holding a replica so that any query can be satisfied regardless of the peer queried. Additionally, it guarantees that a sufficiently high number of replicas exists for any path and that the peers representing a certain path also know some of their replicas. Similarly, the routing tables will hold also multiple references for each level which the routing algorithm selects randomly.

In Figure 2.11 on the next page is represented the current platform's architecture of MediaSense for each node. What we call *overlay*, is the abstraction of the union of all the nodes that belongs to the network.

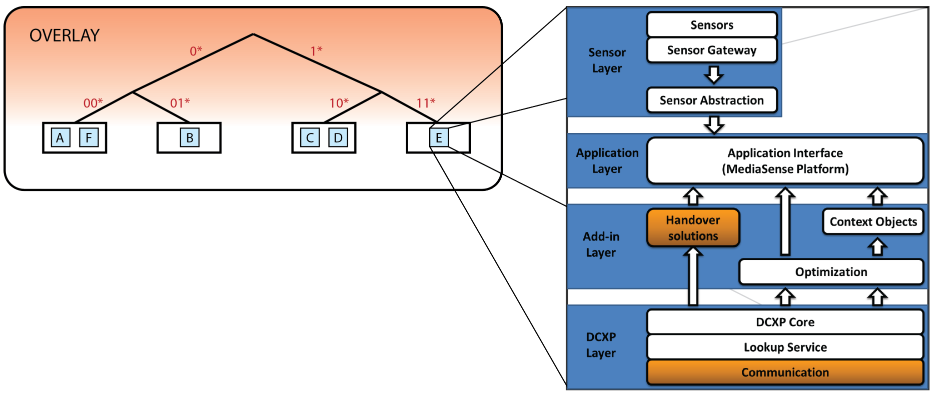


Figure 2.11: The MediaSense single node platform

2.5.3 Mobility models

In the literature exist several mobility models suitable for our purposes. Particularly we found the following ones interesting for the testing phase:

1. Random mobility models (Random waypoint, Random direction, Gauss-Markov)
2. Graph model
3. Obstacle Mobility Model
4. Group mobility models

Excluding the models studied specifically for vehicular networks such as the Graph model, it's interesting to analyze the pros and the cons of the other three. Researchers studied that people tends to group following specific dynamic patterns, thus is quite difficult to study their movement with a random model. Anyway, analyzing the strength points of an advanced version of the Group mobility model, combined with the possibility of defining not-relevant areas on the map, [25] created a model called *Coalition Mobility Model* that provides a middle ground between Graph, Group mobility models, and the disposition of the infantry forces (Figure 2.12 on the facing page). Paths and movements are predefined, and the "leader node" simply follow them.

Clearly, this model is particularly suitable for the study and the simulation of military tactics; in facts, since their concept of groups is actually based on several nodes that may change their disposition as a result of certain events (eg. an order of their leader), having the possibility of realizing different sub-networks of nodes, using this approach we can verify the algorithm in an ad-oriented environment.

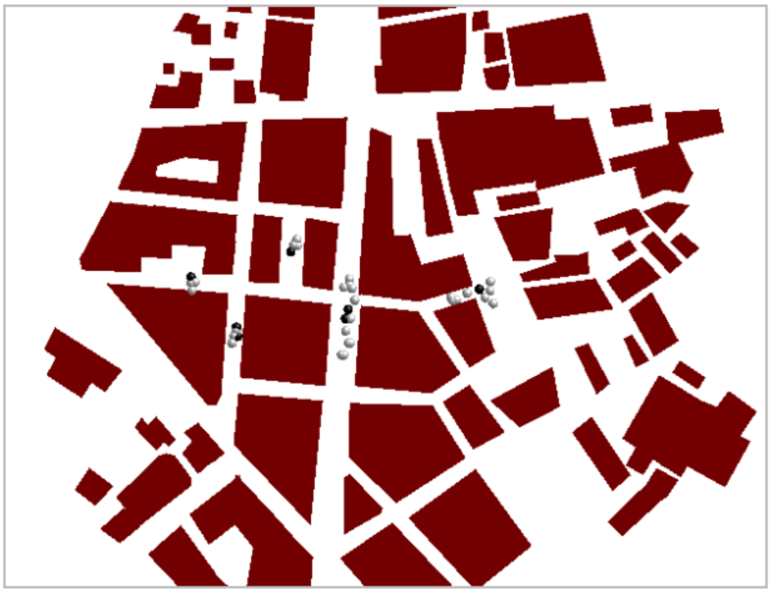


Figure 2.12: Static view of the infantry on a map

2.6 Related works

In the last 10 years, the advent of wireless sensors has necessitated the design of asynchronous, distributed and fault-tolerant information exchange algorithms. Most importantly, since the network's topologies may vary quickly in terms of number of nodes and contexts relatives to each of them, the dynamic and distributed creation of groups in a network of nodes is a relatively old problem that still has not found a proper solution that covers all the possible scenarios.

2.6.1 Gossip Algorithm

One of the first has been the Gossip Algorithm [5], a peer-to-peer, distributed, and asynchronous algorithm for the computation and the information exchange in an arbitrarily connected network of nodes. Since the nodes in the network operate at low energy rates, the "gossip" algorithm's scheme distributes the computational burden communicating with a randomly chosen neighbor, and afterwards the nodes computate a common task that was, in this specific case, the computation of the average temperature in a room. In 2013, a new version of the "gossip" algorithm have been proposed [38]. This solution, maintaining the same number of nodes, reduces the convergence time and also saves the energy of the sensors, thus increases the scalability of the algorithm.

2.6.2 Flow-sensor Mobility

In 2012, researchers from the KTH of Stockholm proposed a study on communication between sensors, access points and a packet transmission algorithm in order to divide into several multicast domains a list of several nodes, both stationary and mobile. The network architecture was based on OpenFlow, that can break the traffic path into data packets (controlled by the underlying router) and control packets (controlled by the controller) [17].

2.6.3 MANET

This section relies predominantly on [31]. Since the future networks will be mainly based on wireless technology, one possible evolution is strictly related to mobile ad-hoc networks (MANET). In this scenario, all the limitations derived by the infrastructure (base stations, routers etc.) are deleted. In fact, the IETF is working on a standardization of routing in ad-hoc networks , basing their goal on the scalability of the entire routers' network. As already said, their work is based on Mobile IP and standard IP addressing, but still their work is not reliable for network with more than a few hundred of nodes.

The main challenge in this situation is the scalability for very large networks, and the keys for the whole work are still a bootstrapping protocol and an infrastructure onto which rely.

Lastly, to supply a temporary service the MANETs are created on-the-fly and have to be highly volatile (they must be robust to failovers and sudden topology changes).

Even if the managing of IP addresses in a MANET is not straightforward, it presents some typical charateristics:

- Dynamic topologies
- Bandwidth-constrained and variable capacity links
- Energy-constrained operation
- Limited physical security

These routing protocols are considered as interior gateway protocols (a set of interconnected networks under the same administration authority), and there are two main fields of study:

- Proactive protocols
- Reactive protocols

The formers attempt to maintain routes continously exchanging the routing tables among the neighbors: in this way, a rout is always available when asked to a node to forward the packet/s. The latters instead use a discovery protocol that tries to find a rout only when needed.

There is also a third field that combines both the beneficies of the first method (always up-to-date routing tables) with the beneficies of the second (less overhead on the infrastructure), like the ones that proactively handle all the routes known to be more frequently used. These are called *hibrid protocols.*

Such an approach could be linked to geographical information of the nodes, reducing in this way the propagation of control messages, reducing the intermediate system functions. In fact, geographical routing allows nodes in the network to maintain only their one-hop neighbors' information. Another approach (always related to the maintainance of the scalability in small networks of 10-100 nodes) is the use of clustering. Within a cluster, a traditional MANET algorithm can be permitted throught the use of clusterheads and border nodes, where the clusterheads form a set that works as backbone for the network. This last is usually called *geographical routing protocol.*

2.6.4 MANET Clustering

In order to mitigate the topology changes in mobile ad-hoc networks, researchers proposed several algorithms suitable for the dynamic clustering. The main assumption made in a vast number of solutions, is that the MANET is homogeneus, that make them to suffer of very bad scalability.

Weight Based Adaptive Clustering

[36] proposes a solution in which the nodes are semantically subdivided in two groups: the former is for normal hosts that act as cluster members, the latter instead is for the cluster heads, the nodes responsible for the maintenance of the topology. They found three main issues in the realization of the protocol: the first is the election of the cluster head, the second is formation of the cluster and finally the cluster maintanance.

Cluster head election procedure is based on an heuristic that weights the suitability for the node to be a cluster-head. This heuristic is based on the ideal number of nodes in a cluster, the battery power, the average link stability and the average dependency probability.

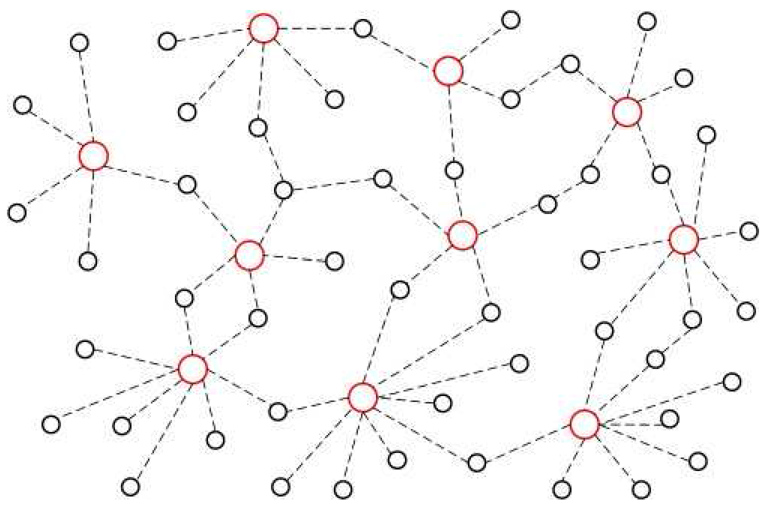


Figure 2.13: General model of hierarchy MANET [36]

Cluster formation is the phase in which every cluster member decides its cluster head.

Cluster maintenance is the last phase, in which a node may join the cluster or leave it, and in this paper they proposes several solutions in order to avoid the possible problems and to achieve a good scalability.

Location-Aware Routing Protocol

Another approach to achieve good results with MANETs is to reduce the overhead of the employed routing algorithms, that must be independent from the total number of nodes. [37] proposes a solution really similar to the previous one, but in which they try to form a stable clustering of a large scale MANET, and then to perform route discovery by using the geographical location of mobile nodes.

Route discovery might be done using the information obtained by the node itself: the timestamp of when the destination's location was collected) and the velocity are usefull information because they can be used to predict a smaller request zone which covers the possible location of the destination in a relatively close past.

MANET based on Autonomous Clustering and P2P Overlay Network

Since usually nowadays every node in the network can communicate with its neighbors with Wi-Fi or Bluetooth and can access to the wireless infrastructure such as 3G or LTE, in [20] is proposed a solution where MANET is based on a P2P network constructed by using an autonomous clustering.

The network is divided into multiple subnetworks called cluster, and a node is automatically elected as cluster head of that cluster. Aftwerword, this nodes joins the P2P Overlay Network and it communicates with the other cluster heads through the 3G/LTE infrastructure (Figure 2.14).

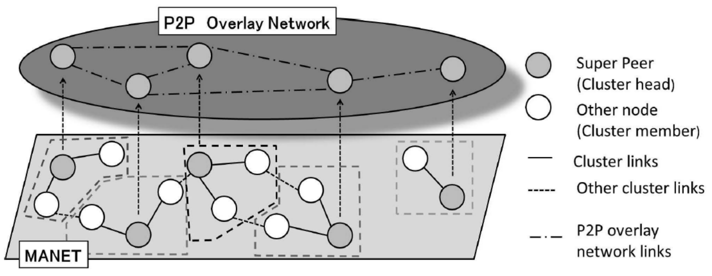

P2P Overlay Network
MANET
Super Peer (Cluster head)
Other node (Cluster member)
Cluster links
Other cluster links
P2P overlay network links

Figure 2.14: Example of P2P overlay network topology [20]

In this paper is shown that the more the density of nodes is hight, the higher is the success delivery rate.

CHAPTER

THREE

DDC ALGORITHM'S DESIGN GUIDELINES

The purpose of this chapter is to show the most important decisions taken during the designing of the DDCA. We are going to code a program that will run on a general purpose Linux machine, and that means that we have got a lot of extra-power that will not really need. The general aim is to let the program to be as lightweight as possible, to let other developers and researchers in the future to make a porting to a real Android device if they'll find it useful for their purposes, but since the core of the P2P's protocol is still studied by other researchers, maybe it will not be true in all the project's parts.

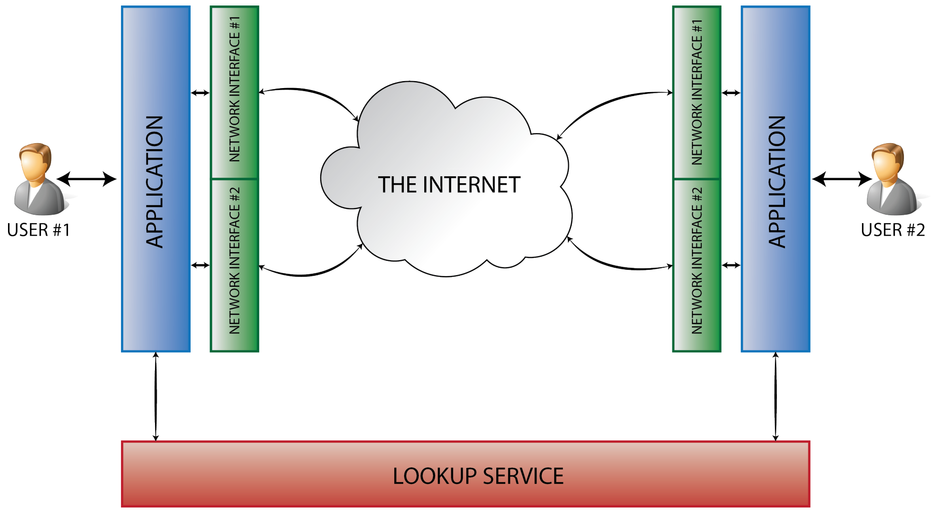


Figure 3.1: Working schema of the handover system

To let devices be continuously accessible from the *Internet of Things*, it's important to have a reliable handover layer underlaying the entire mobile infrastructure. In the previous chapters we understood what the problem of mobility is and which are the current solutions that researchers have developed to solve it. The best way to find out a solution is to explain why it would be preferable to make use of one protocol rather than another and if it would fit our purposes.

For first, our aim is to build a good environment in which tryout the entire switching system. It means that we'll need a device able to connect to two or more interfaces at once and that is able to switch from one to another depending on a series of considerations made over physical data like signal strength and for example the distance from the Access Point in 802.11 connections. We also need a result infrastructure whose purpose is to continually collect data from the device that will help us to make some final considerations about the time of the handover.

Then, regarding the second aim, we need to try to dynamically add and remove groups from the Overlay to understand its response time and we will not need to use it on a real device.

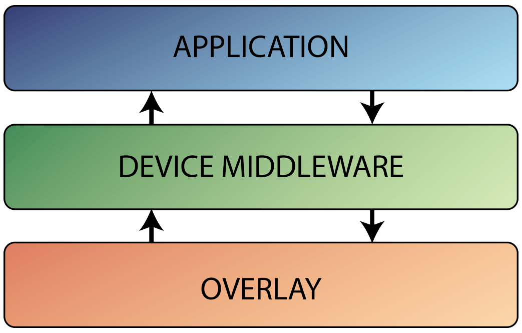
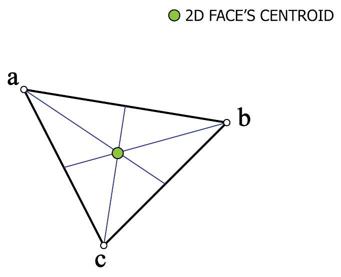


Figure 3.2: Where is positioned the middleware in the communication

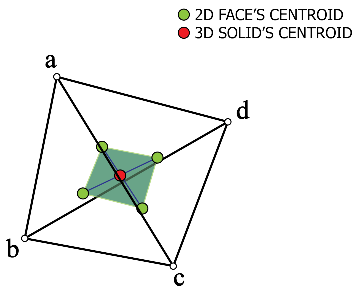
3.1 Grouping design's methodology

In the network a node corresponds to a sensor or a user's device which is attached to the Internet thanks to IP connectivity. The aim of the thesis work is to create a support for grouping in a heterogeneous network, and P-Grids have being created to allow the nodes to persist data in a distributed scenario. MediaSense Framework is a P-Grid enhancement project that can be extended with such functionalities. Both heterogeneous networking and necessity of creating groups are the reasons why the work focused on a distributed scenario rather than a centralized one, especially because the second can't scale globally. Moreover, the work requires that all the nodes participate actively to find out the groups. The ability of assigning one entity to a specific group of elements that are more similar to each other than to those in other groups is called by researchers as *clustering*. To simplify the work, we are assuming that the grouping algorithm that makes the decision whether a node is important for current node or not already exists.

Importance algorithm What is really important to focus is how the algorithm should work. Actually, it would be interesting to have a mathematical approach to solve this issue. In fact, is possible to create a model, instant by instant, of what is currently happening in the network. Every node can create its own model by its own context information. Using such information, every node is able to understand which of the nearby nodes are important or related to it. A node is related to another when their respective distance stays inside a specific range that can be calculated. To simplify one step, we will use a simplified context model that only works on latitude and longitude and calculates the geographical distance between the nodes in order to select if a node is important or not. Then, when all the important node have been found, it's possible to select the most important one as the closer to the *baricenter* (or *centroid*, Figure 3.3b) of the group. That node will be the reference point for that specific group.



(a) 2D Figure Baricenter



(b) 2D Solid Baricenter

Figure 3.3: Centroid examples

The typical scenario in which groups could effectively work, depends on the number of nodes that are currently active in the network. Thus, is possible to argue that the minimum number of nodes of the network should be not less than 3, else it would be useless to group nodes.

Grouping scenarios In order to create groups in the Overlay, we analyzed different solutions. Current DCXP architecture provides the easy-to-use mechanism of the UCIs, as said previously in section 2.5.1 on page 37 that follows this syntax:

`dcxp://user[:password]@domain[/path[?options]]`

What the group system would do, is to add a level for the groups directly into the UCI syntax, like the following one:

`dcxp://user[:password]@domain/group/[/path[?options]].`

First solution The first solution is represented in Figure 3.4. It is based on the assumption that each device has its own Overlay Layer, and in this way every person will have the complete control of his own resources. Actually, that would mean that every node is responsible to create its own tree, within which are stored all the key/value pairs of its resources. This solution is really easy to implement, but there are some drawbacks about performances and scalability. In fact, implementing such a solution would mean to loose all the potentialities that are offered by P-Grids, like the fast search in big trees, that would not be considered at all. Thus, we will not pursue this solution.

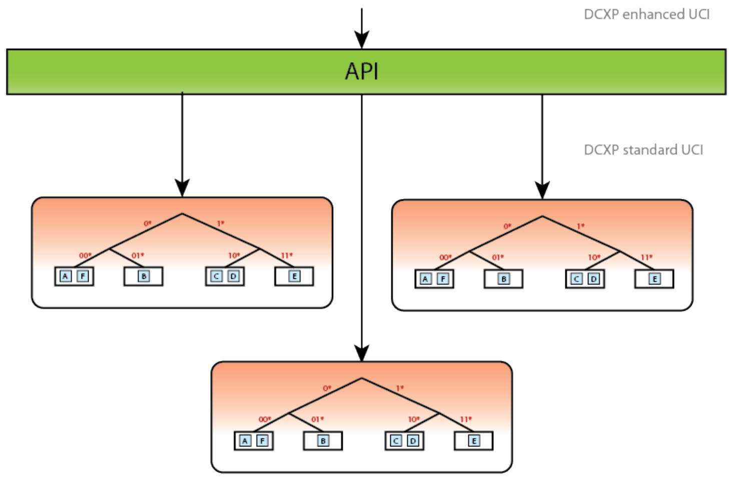


Figure 3.4: First solution

Second solution It consists on the creation of a new abstraction layer upon the existing *Media Sense Framework*. The main idea behind this solution is to create one group for each different context. Groups should be dynamically created and updated upon context changes. Then, resources shall be temporarily cached in the layer to override multiple requests for the same resources. Actually, implementing this solution would add some overhead during the group creation due to the time that involves the nodes request by the Overlay system, and also induces some overhead during node accessing because, despite the use of a cache system, every time it will require the whole time to search for the resource in the tree. However, this solution is really simple to obtain because it will not add neither complexity to the Overlay Core, nor to the APIs system. This is possible due to the fact that there will not be any modifications to the undercore UCI system. In fact, what depends on groups management will be delivered to the highest part of the Overlay and only the underlaying part will manage to find the resources (see Figure 3.5).

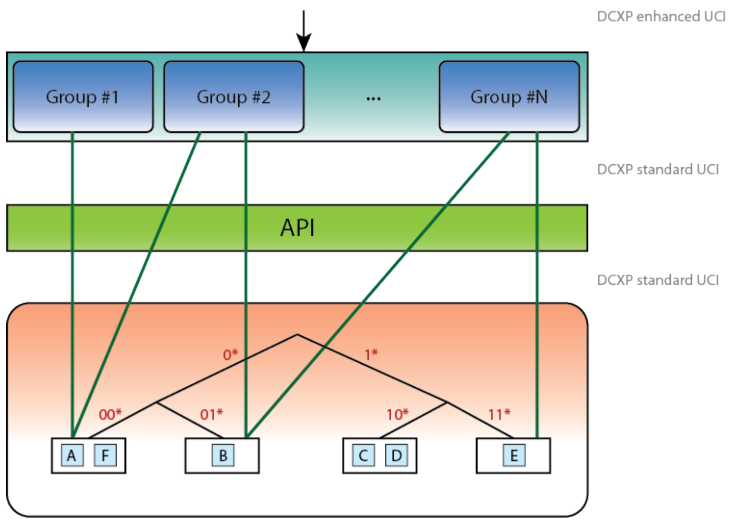


Figure 3.5: Second solution

Third solution Instead, the third solution, represented in Figure **??**, consists in the radical modification of the internal core of the P-Grids. Actually, P-Grids store the information like binary data distributed in the whole tree and referenced by a structure based on the key/value pair's paradigm. Adding the concept of groups inside the tree would mean to create intermediate nodes between the parent nodes and the resource nodes themselves. Moreover, the intermediate node is a resource itself, and that would mean that for every context update in the whole tree (a continuous handling of more than 50 billions of nodes) the tree shall be restructured to be consistent to the physical world from where the context information come. Effectively, this solution would not add any kind of overhead in a static context, because the management of the new UCIs structure is located directly into the Overlay Core, and that means that is more flexible for future enhancements (like involving other subgroups based on the ontology of the nodes, not possible with the first solution). However, the system shall guarantee to work in a fully dynamic context, and even if this solution would be the optimal in terms of load balancing and overhead of the system, it would permanently breaks the modularity principle upon whom the *Media Sense Framework* is based.

In Table 3.1 on page 56 is possible to see all the solutions compared each others.

Finally, our proposal is to design the grouping system upon the current architecture as a new abstraction layer that could be used separately.

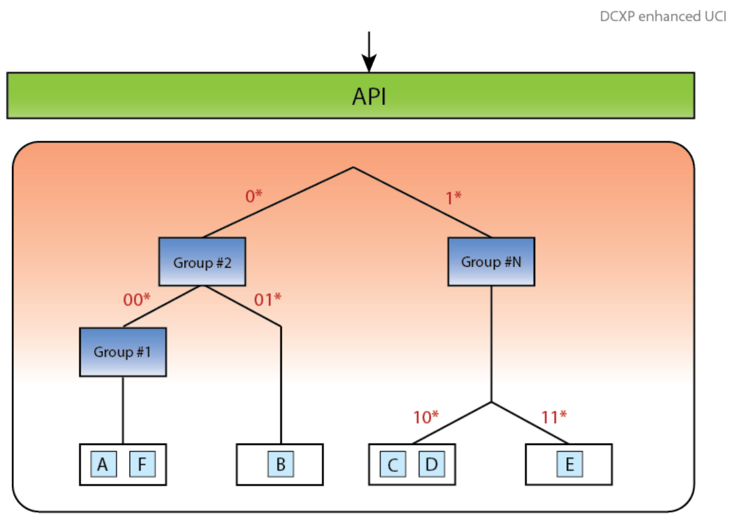


Figure 3.6: Third solution

3.2 Handover design's level methodology

In order to give the maximum flexibility to our handover system on a real Android device, we fetched around some Internet sources like [11], [10] and [22].

We want to use SCTP for this purposes, making a real multi-homing environment on a mobile device where two or more connections can be used contemporary to find out the best usable in a particular moment and location. This can be done arranging a mobile real time usage statistics as it happens in the GSM/UMTS technology. This idea is strengthened by the fact that today's handover protocols like MobileIP or HIP doesn't take advantage of the potentiality of the infrastructure, because they wait for an handover request coming from the device itself when it changes its location. This is not reliable and takes too much time to respond and it would be a great problem for TCP connections.

SCTP is a reliable and robust transport protocol that can handle more interfaces at once. Moreover, the latest version of the SCTP draft as proposed by IETF in 2001 is already implemented in all Linux Kernel since the firsts 2.4 versions and now its maintenance is managed by the LKSCTP team [16].

Linux SCTP's header files are not available for default for developers. Despite to the fact that this protocol has reached a good level of maturity, it's not yet considered a standard. Due to this, to enable the SCTP fruition, developers must activate the *LKSCTP-tools* module in the Linux Kernel, installing that package via a common `sudo apt-get install`

	Personal groups / Different Overlays	Context-based groups / Grouping layer	Context-based groups, integrated in the trie
Pros	• Easy to implement • Every person will have the complete control of his own resources	• Easy to implement • Groups are totally dependent to the current context	• No overhead • Flexible and robust for future enhancements
Cons	• Difficult management of the shared data • No more fast search in big trees offered by P-Grid	• Overhead during the group creation and management	• Hard to implement • Too many reconstruction of the trie due to the mobility

Table 3.1: All the solutions compared

`lksctp-tools` command (for Debian/Ubuntu). Moreover, since *Java JDK 1.7.0*, a Java implementation of SCTP has been made, but it can be used only on Linux OS prior the kernel module activation.

Windows Website [14] provides a huge list of all known SCTP implementations for all the platforms. Most of them are internal implementations created for research scopes and available under purchase. The only Windows version available for free has been implemented by the cooperation between *Siemens*, the *Computer networking technology group* and the *University of Essen and the Münster University of Applied Sciences* [32].

Android Android, as well known, is a Linux porting for mobile environment. However, as not so well explained in [3], to install and activate the *LKSCTP-tools* module here is a little bit more difficult. For first, we should download the clean Kernel version for the device from `kernel.org`, then add the module (as done on Linux via `apt-get`, but manually). Secondly, we should extrapolate the Kernel's configuration file located in `/proc/config.gz` from the Android emulator and push it into the modified kernel directory (in the same directory). Finally, a Kernel rebuild is required and then we can flash the new Kernel directly to the device. Clearly, this methodology is not so safe, because a Kernel rebuild means to have the *root* access to the phone, and a kernel misconfiguration could mean a bricked phone. Hopefully, the SCTP will be implemented in the future Android version, but developers have become interested in it only in the last 2 years. Until the implementation of the protocol directly in the Android stack, it will be quite improbable to see real applications in the next future.

Therefore, the adoptable solutions are two: the first is use a laptop with a Linux distribution (like Ubuntu) and with both a Wifi and a 3G connection, where we can build a real SCTP solution. Otherwise, we can use a real Android device, but that will mean that we cannot rely on the SCTP protocol for the solution and we must build a *middleware* layer ourselves.

For the reasons explained before, we choose the first solution, and hopefully in the next years the Android Consortium will open the SCTP tools to developers.

3.3 Simulation scenarios

For the handover experiment, we will test the application on a real environment with more than one 802.11 Access Point and a 3G connection. The device's application will be able to directly interact with the network interfaces and we want to pass the following experiments:

1. Passing from Wifi to 3G without connection breaks;
2. Passing from 3G to Wifi without connection breaks;
3. Passing from Wifi to 3G to Wifi without connection breaks.

Regarding the dynamic grouping solution, to test its effective functioning, we'll make the following experiments:

1. Dynamically add a group to the Overlay;
2. Dynamically add and remove nodes to the group;
3. Dynamically remove a group from the Overlay both with sub-nodes and not.

3.4 Evaluation of software design transparency and usability

As stated in the first chapter, the handover solution should be implemented as transparently as possible to the developer. This means that during development the user can choose whether to use our framework or not, like he do as when deciding to use UDP or TCP. For testing purposes, the classes will be developed as a standalone *.JAR* file, but further improvement will be discussed in the *Future work* chapter.

The first is reasonably the most useless of the three for our purposes, because it takes too much time and doesn't help us to maintain the connectivity during the switching. The second one is powerful because it isn't so expensive from the point of view of the infrastructure, however we made some tests and in the best case it takes from 5 to 10 seconds to make the switch and it needs some manual configurations from the user to let it work in that manner. So in our solution we will focus only on the last two points.

The results obtained will be used to understand the bottleneck of the solution and will be reported in the *Results' Chapter*.

3.5 Test driven development

To be in step with the times, we considered some general software development processes and the one that seem to be more correspondent to our needs relate to the Test driven development (TDD). This is due to the fact that what we want is a framework that works with to SCTP to make some special operations like the ones specified in Chapter 3. That means that we have a starting idea of what the program should do and in which way we wants to interact with it or not. This software development process provides a very short development cycle that is repeated several times. As represented in Figure 3.7[1], for first the developer *writes the test cases* that define the desired improvement or functionality, then he *writes the new code* to succeed the test and finally he *refactors* the code to be as standard as possible [2].

It's is also known that TDD can have some problems with specific network configurations, because these depend on external and unpredictable specifics. Due to this reason, only the business logic of the application will be covered by the tests.

[1]Original URL `http://en.wikipedia.org/wiki/File:Test-driven_development.PNG`

[2]`http://en.wikipedia.org/wiki/Test-driven_development`

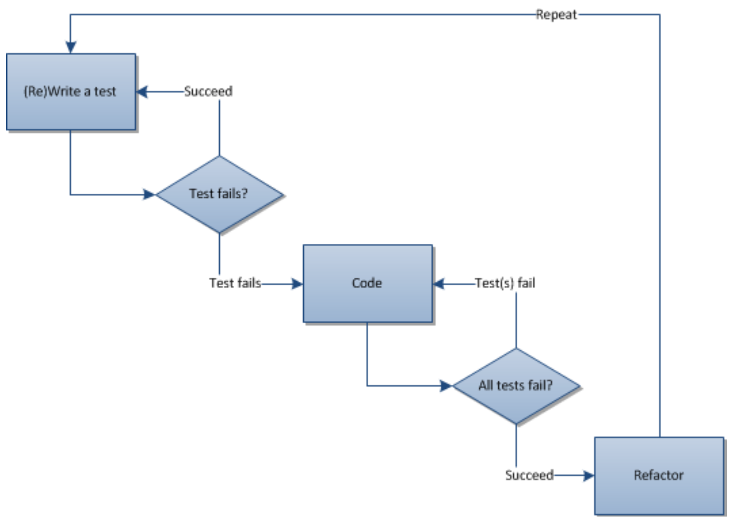


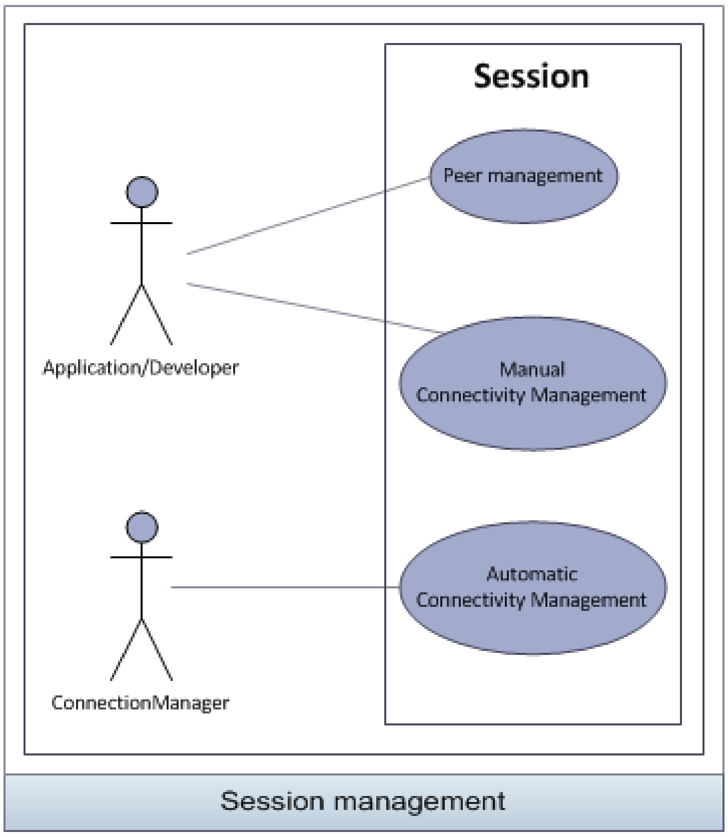
Figure 3.7: Test driven software development process

3.6 Analysis of requirements upon session management

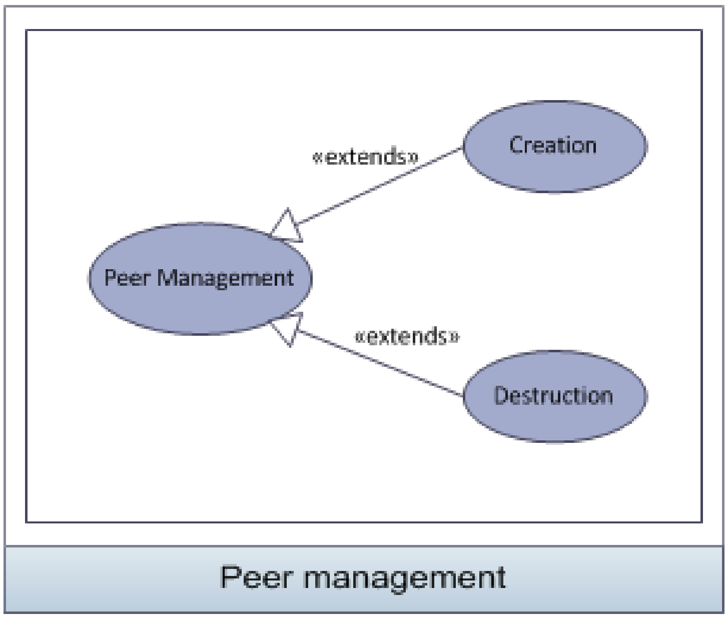
The first thing to do when approaching a new project is to understand which is the use a common user will have working with it. This process is called analysis of requirements, when the analyst talks with the customer and tries to understand what are the final characteristics of the project. Since this project hasn't got a customer, we should make some hypothesis about the most common uses that a developer would have with the framework, trying to be as general as possible defining some simple and useful APIs for him.

Due to this, we try to create some typical *use cases* that are represented in Figure 3.7 on page 61. Let's analyze them separately.

Session management In Figure 3.8a on the following page is represented the typical use case that will have our system. The user/developer can decide to use SCTP for his solution. Clearly, as explained before, this protocol is already defined into the Kernel of the Operating System, so we don't have to care about all the aspects about packet handling and delivery. What we shall do is to create an environment where the user can easily handle the switching between two connection. As it's possible to see in the figure, the developer can instantiate a new *peer* for the machine and can both creates and handles the active connections with other peers. Instead, the *ConnectivityManager* is the system itself that can only handle the existing connections. All this entities have been wrapped by a unique object called *SctpSocket*.



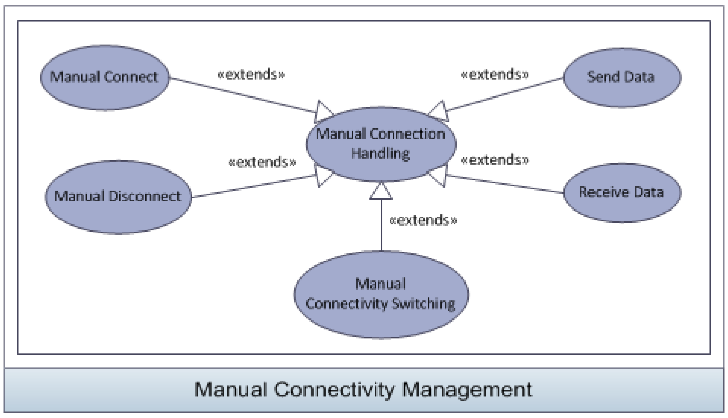
(a) Session management



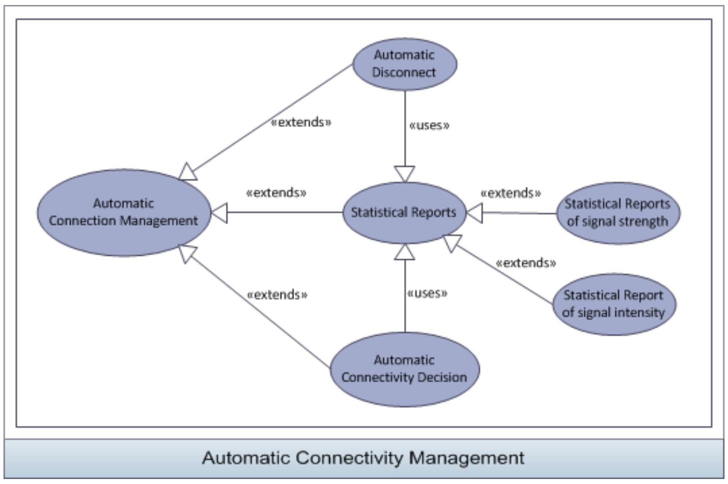
(b) Peer management

Peer management As represented in Figure 3.8b, the developer can decide both when to create and when to destroy the *peer* object.

Manual connection management As represented in Figure 3.7c on the facing page, only the user/node/developer can decide when to connect to



(c) Connection handling


Automatic Disconnect
«extends»
«uses»
Automatic Connection Management
«extends»
Statistical Reports
«extends»
Statistical Reports of signal strength
«extends»
«extends»
«uses»
Statistical Report of signal intensity
Automatic Connectivity Decision
Automatic Connectivity Management

(d) Connectivity management

Figure 3.7: Typical use cases for the handover manager

another peer. After the connection, it can both sends and receives data, decide to manually disconnects from the other peer and finally to switch between the existing connections.

Automatic connection management The use case represented in Figure 3.7d on the previous page is probably the most important, because it represents the core of the system. After a connection, the system will use statistical data coming from the connection to decide what to do. In case of a connection's dropping, the system must act with a kind of *Automated decisioning system* and should be able to reestablish the connection between the two hosts. This can be really hard to do, but we'll try to insert some buffering system that can prevent from the disconnection.

3.7 System behavior

It's really important to figure out what is the behavior of the system when facing particular circumstances. This can be done using the *sequence diagrams*, that are very useful in environments such as networks where it is important to understand which are the high level messages that are being exchanged between peers and also between internal software parts.

Peer connection In Figure 3.8 is represented the process that being activated by the connection request by the user/developer. Working directly with the API's layer, we want to have one *cache system* that helps us maintaining the throughput as higher as possible. This is due to the fact, as well explained in section 2.3 and in section 2.5, that probably we want to ask the system to open a connection between us and our *home's thermostat*, that is more user friendly rather than an address like 91.15.128.32. In case of a *cache miss*, we need a kind of Mobile Naming System (MNS), that acts like a *DNS* (but with all the potentiality of the *Overlay*) to give us the current resource's IP/port pair. We'll talk about the MNS later. After the address resolution, we have got all what we need to connect to the other peer.

As it is possible to see in the figure, every single node in the system is predisposed to accept multiple IP/port pairs from the other peer. This can be useful in the future, where there will be an easiest way to take advantage of multi-homing features from the peers. One important note is that the *connectionResults* object is the instance of a Network Stream that can be write directly from the user.

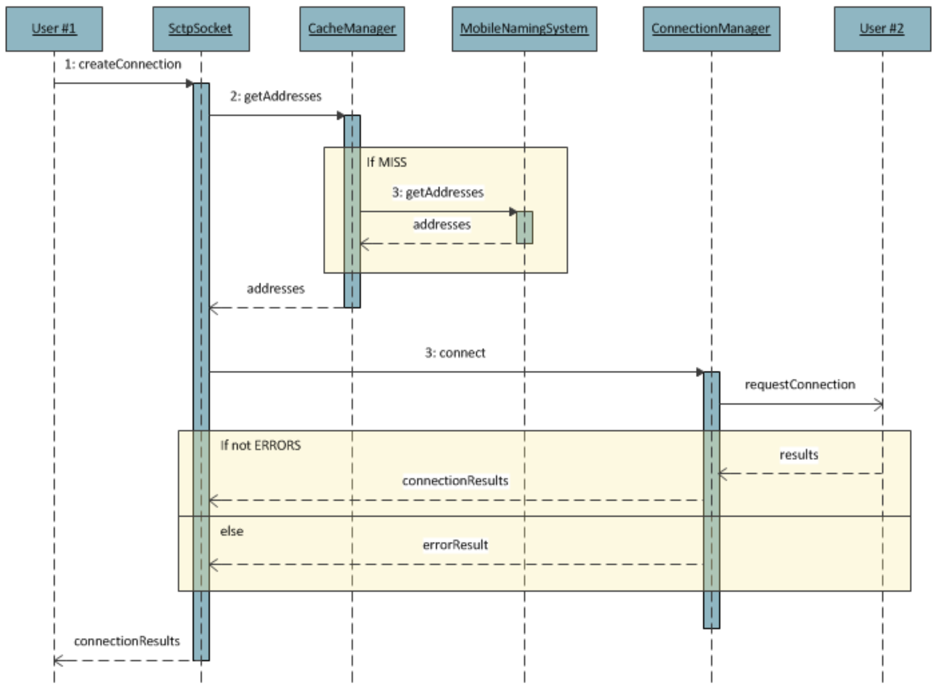


Figure 3.8: Sequence diagram of the connection process

Peer connection's swapping One of the biggest issue is related to the fact that in a publish/subscribe system, the packet delivering time can be higher than the time required to swap the connection. This can be exemplified with the scenario represented in Figure 3.9.

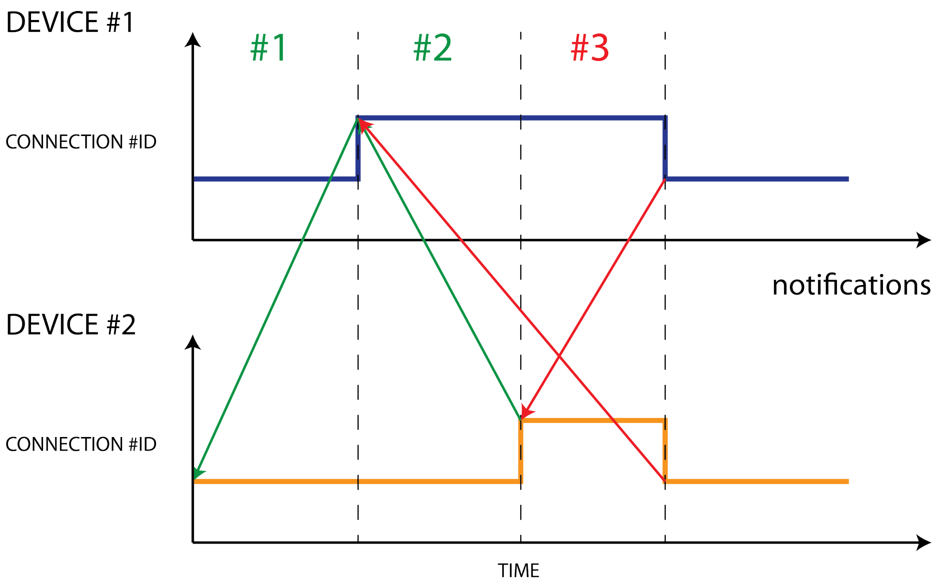


Figure 3.9: Connection's swap notifications between peers

In the figure are represented three major situations. In the *first*, device number 1 swaps from connection number 1 to connection number 2. After the operation, it can correctly notify peer number 2 because it knows its

IP/port pair. The same happens in *second* scenario, where in this case is peer number 2 that swaps its connections. Instead, in the *third* scenario, neither the first nor the second device can notify the other peer, due to the fact that they changed their IP/port pairs simultaneously.

There are two major fixes for this problem. The first is to monitor all the outgoing traffic and always bufferize it, but it's extremely expensive in terms of computation requisites. The second solution is to take advantage of SCTP, that is able to manage multiple IP addresses dynamically. The idea beneath this capability is to open the second connection before the swap mechanism begins to work and notify other peers before the swapping. This means that the prediction mechanism should be able to understand when change the connection very early.

Peer data exchange The most important feature of the *Sensor Socket*s is the possibility to exchange directly binary data. The ratio behind this functionality is to to permit the exchange of both raw data and complex files (like media files) through the P2P network. This can be done by using Streams as communication medium. Four different API have been designed for the different level of communication that can be established between two peers:

1 Write Plain Text (Custom Charset);

2 Write File;

3 Write Streaming Media;

4 Write Raw Data

Each of them is designed to be optimized in its particular field. For example, sending a file through the network is different than streaming a media. In fact, the former needs complete reliability during the transfer, the latter needs highest speed in the data transfer. Anyway, we could synthesize the behavior of each communication between peers as shown in Figure 3.10 on the next page.

3.8 Handover system's architecture

The handover system's architecture has been studied to be as flexible as possible for the end-user. The entire structure takes advantage of the Peer-to-Peer paradigm, so each node of the network can be both Client and Server for the other nodes because each operation is executed in a different thread.

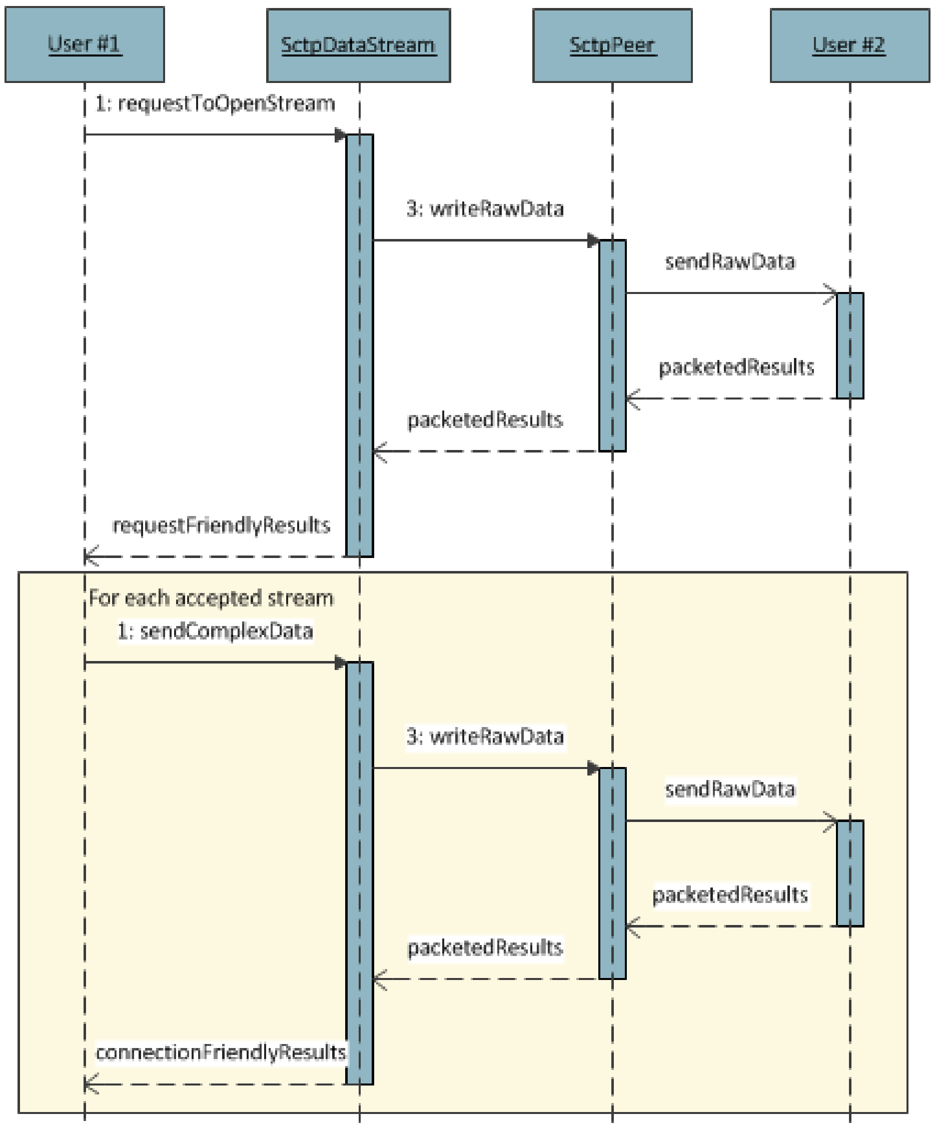


Figure 3.10: Communication between peers

Data Buffering In order to maximize the payload during heavy data transferts (e.g. multimedia streaming), it would be necessary to optimize the sockets with the use of IO data buffering, both in output and in input routines. This can be exemplified by the following scenario: we are moving towards different connections, so if we are streaming a large amount of data, it would be very useful to have bufferized streams both in input and in output. Clearly, the faster the switching is, the better it is for the connection. Thus, the dimension of the buffer should be modeled on different circumstances: firstly, the *data rate* of the streamed source. Secondly, the worst case of time delay's switching. Taking Dr as the bit-rate of the media source, T as the time needed to buffer and looking for the dimension of the Media Buffer (M_b) in bytes, then it's easy to calculate:

$M_b = \frac{Dr \times T}{8}$

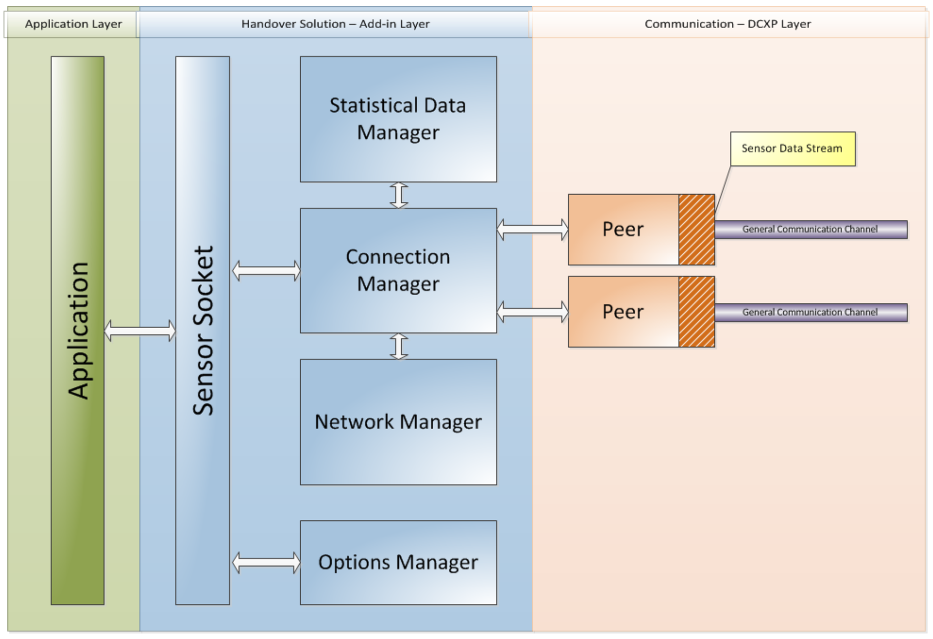


Figure 3.11: Handover system's architecture

Thus, presuming that it's easy to extrapolate the bit-rate from a media source (Dr), we need to know only the worst case of the connection switching (T).

3.9 Mobile Naming System

As reported in section 2.5.1, the Overlay creates an abstraction layer able to resolve IP/port pairs from the UCI of the node whereby we want to connect with and directly return the result of the request. The idea is to take advantage of this behavior to resolve the IP/port pairs of the peers involved in the communication and retrieve them in order to connect with them. In fact, the Overlay acts as a real-time lookup service for all kinds of resources, and treating the IP address like a resource of the node is exactly what the system needs. Thus, we will use a public MNS that is able to query the Overlay in order to find the relative IP address/port pair from the highest-level UCI of the Peer. Taking the following UCI:

```
dcxp://user@domain/IP
```

It will return the following value:

```
192.168.1.123:54768
```

Where the first is the IP address of the peer, and the second is its active listening port for the data exchange.

3.10 DCXP integration

As stated in the previous chapter, groups are not stored directly into the Overlay, but they are maintained in a superior layer by each peer. Thus, groups information are peer-dependent, and that means that a peer could belongs to a particular group only if the peer is currently active in the network. To better understand this passage, in Figure **??** on page ?? is explained how a peer can contain groups. At the moment of the creation of the peer itself, it belongs to one or more groups depending on the number of different context the user could be part of. If for example the user is interested both in music and sports, he could be member both of the context relative to his preferred band and football's team. These groups are not named groups. This is due to the fact that the DCXP protocol is not an active protocol, that means that peers exchanges messages only at a low level of the protocol. In fact, the abstraction level of DCXP only permit to have a passive communication with the Overlay and being informed when it changes.

Platform With Groups.png

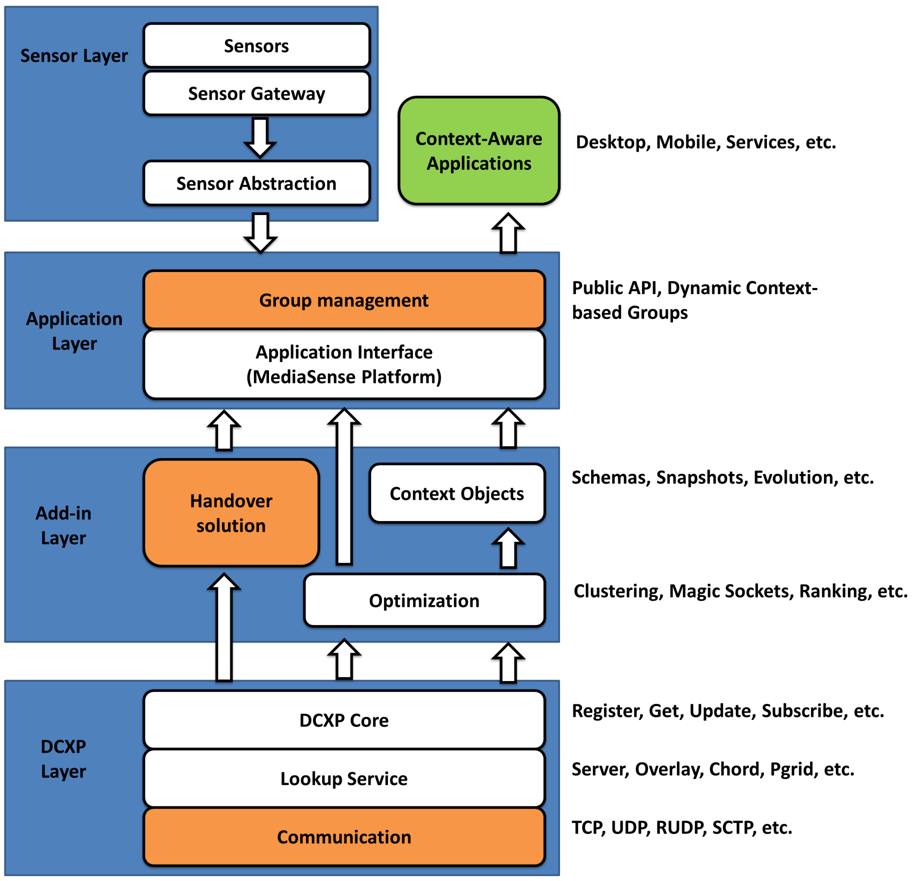

Sensor Layer
Sensors
Sensor Gateway
Sensor Abstraction
Context-Aware Applications
Desktop, Mobile, Services, etc.
Application Layer
Group management
Application Interface (MediaSense Platform)
Public API, Dynamic Context-based Groups
Add-in Layer
Handover solution
Context Objects
Schemas, Snapshots, Evolution, etc.
Optimization
Clustering, Magic Sockets, Ranking, etc.
DCXP Layer
DCXP Core
Register, Get, Update, Subscribe, etc.
Lookup Service
Server, Overlay, Chord, Pgrid, etc.
Communication
TCP, UDP, RUDP, SCTP, etc.

Figure 3.12: DCXP Integration Architecture

We decided to put this as the strong point of this research, because having a passive protocol means to not take care of how the communication happens between the peers and just being informed when we have to take some decisions. This behavior highly increases the scalability of the system.

3.10.1 Group creation and management

When a node physically moves around the space, it shall take decisions every time something important happens around it. For our purposes, into the real world, this measure can be dimensioned to once every 30 seconds. In fact, it's really difficult that more than 1 thing can happen for context changes in so a short time. Actually, even 1 minute has been examined as a good solution, but the answer to the question "Which is the good threshold for a context update?" could be postponed for future works. Thus, every 30 seconds and for each context update, the system shall be woken up by the underlaying DCXP Layer and shall performs multiple *range queries* (one for each context attribute) to find out which peers can be interesting in terms of context's distance.

Context distance Context distance calculation is the most important part of this thesis work. Once the range queries have been sent, each of the nodes that are inside the range returns their context attribute values. The range represents the maximum distance between two different contexts. To determine the distance between two different locations is quite easy, therefore we assumed that each node contains the attributes relative to its current *latitude*, *longitude* and its desired *range*. However, every context information could be used, like music interests and sports, but it's up to the future workers on the system to take care of calculating these context distances.

Once the distance is calculated, all the nodes within the range are queried from the peer to know their centroid position. After that, only the ones that have got a centroid closer to the node are took in consideration to create a group.

3.10.2 Peers and centroids

After the peer has found all the surroundings nodes, it queries the Overlay to know the centroid position of each of them and the name of the node's group. Actually, this operation is really fast because statistically only a few groups could be realistically close to you. In fact, thinking in terms of geographical location, if the node is close to one group's centroid then it

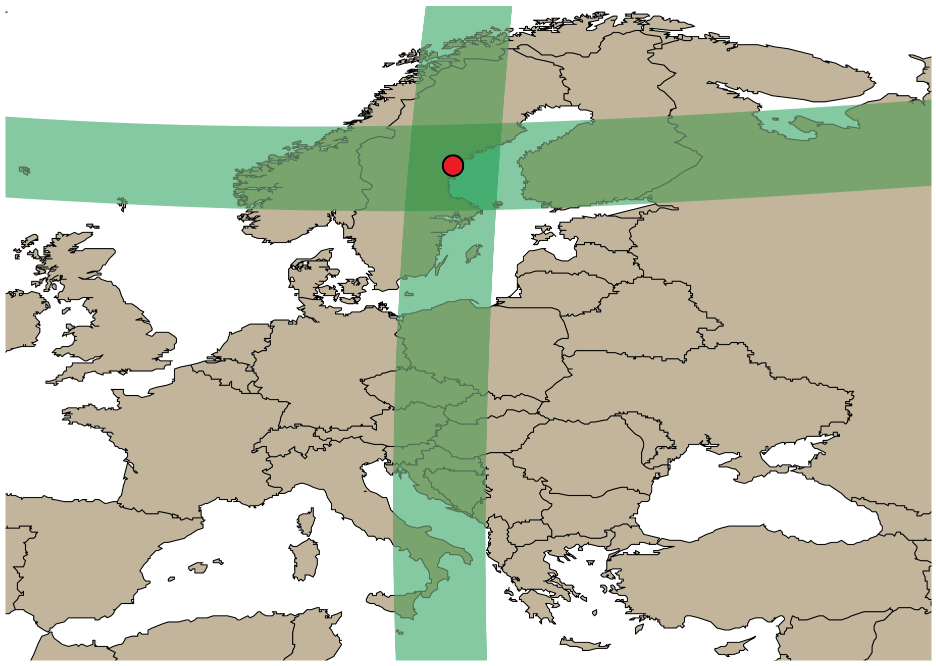
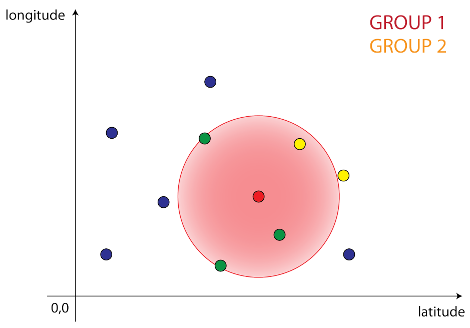


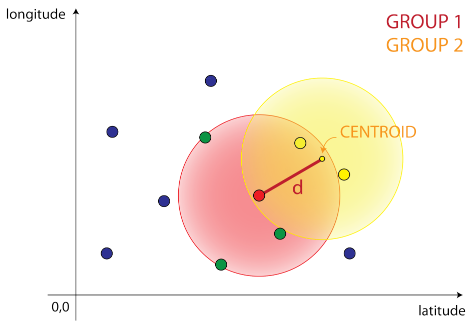
Figure 3.13: Ranged queries for context distance returns the nodes that are inside the quadrangle

means that all the other nodes close to that node belongs to that particular group. As shown in Figure 3.14a on the next page, whether the peer wants to join an existing group, it queries the Overlay to know where is positioned the centroid in the context's space, and calculates the distance between its position and the group's centroid, as shown in Figure 3.14b on the following page. Therefore, because of the fact the Overlay is a passive system (as reported in the second chapter), is the peer itself that changes its group name's attribute.

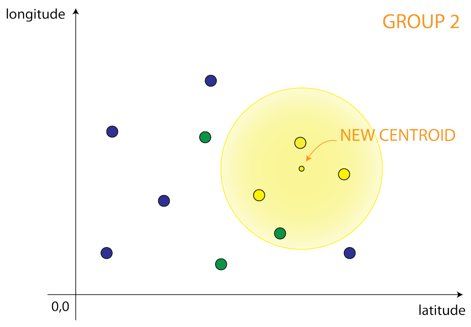
Moreover, with the operation of joining a group, the peer should change its centroid attributes and range. The centroid position is calculated as the baricenter of the previous centroid and the peer's position, and the range is calculated as the mean value of the old range size and the peer's range. Thus, if the old range was 500 meters and the new one is supposed to be 1000 meters, the new range will be set to 750 meters. This permits to avoid centroid size additions that would exponentially brings centroid range to infinite (Figure 3.14c on the next page). Actually, this system presents a leak that is represented by the fact that, being the Overlay a passive system, only the last node that joined the group knows the new position (and range) of the group's centroid. This could be avoided introducing a timestamp attribute during the calculation of the new centroid. If a node during the lookup finds another peer that belongs to the same group, then it also asks for the centroid timestamp. If it's more recent than its own, then it shall



(a) Peer queries the Overlay to know the surrounding nodes and calculates each distance (ΔX)



(b) Peer decides which of them is the best choice and calculates the distance to the group's centroid (ΔC)



(c) Peer decides which of them is the best choice and calculates the distance to the group's centroid (ΔC)

Figure 3.14: Algorithm's first stage - Nearby node lookup

update its attributes with the new ones.

3.10.3 The grouping algorithm

Since in the next chapter we are going to describe the tests made in order to prove the functioning of the whole system, it's important to give a formal definition of the grouping algorithm in a real scenario.

Testing Scenario

We choose to simulate an advertising service that operates thanks to the localization information of the nodes. These nodes are positioned randomly in a map (0 to 100 on the graphs). The paths followed by them through the map are predefined). Some areas of the map are forbidden, since they represents the buildings. Thus, these are the terms used in our study:

- *M nodes* represent the stores
- *N nodes* represent the people moving on the map
- *R nodes* represent the routers at which the nodes are phisically connected
- T represent the topology area
- O represents the overlay
- C represents the context of the node

Each node (N and M) maintain its own context. When consulted, O supplies an answer containing the information of the closest node in terms of euclidean distance of the contexts; therefore, these two nodes shares their knowledge about their respective groups and both join the same one. Finally, after some iterations, nodes tend to dynamically group themselves.

Formal definition of the algorithm

Initially, N nodes are placed randomly on the map (100x100 units). They are not logically divided, since they represents the people moving on the map. M nodes are placed at fixed positions, since they represent the stores. Routers' nodes are stationary, and responsible to create different domains (cellular, wifi and so on) and to reroute the packets to and from the Overlay.

We decided to use the random way-point mobility model in order to route the nodes on the map at a random speed. The variation have been assumed from 0 to 5 meters per seconds and 0 seconds of pause time. Way points are uniformly distributed over the topology area.

The following are the life-steps of an alive node:

Phase I The nodes begins a lookup operation for its context information and receives a list with the K-most closest nodes (1 in our study);

Phase II The node which started the lookup operation joins the closest node's group and updates its timestamp.

CHAPTER

FOUR

IMPLEMENTATION INSIGHTS AND EXPERIMENTAL RESULTS

In this chapter we decided to explain how DDCA was implemented, the tests we took with the MediaSense Framework and our grouping algorithm, and to note all the issues (and the corresponding solutions for them) encountered during the use.

4.1 Grouping system

Grouping layer is responsible to create, manage and destroy the groups. Actually, the groups are never created or destroyed physically, because groups are nothing more than attributes of the nodes. For this reason, joining and leaving a group means just to change the value of the *group*'s attribute of the peer.

4.1.1 GroupLayer

GroupLayer, as shown in Figure 3.12 on page 67, is the external level of the MediaSense architecture. It's responsible of the maintenance of every single context's group whom belongs the node. So it's important to remember that there are as many GroupLayers as many context the peer wants to keeps track of.

As shown in Figure 4.1 on the next page, the class is *abstract* in order to maintain the dependency level as lower as possible. Anyway, the basic functionalities are described by the publicly accessible methods.

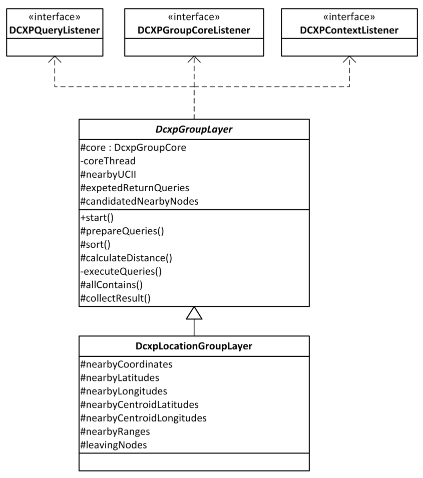


Figure 4.1: GroupLayer UML Scheme

prepareQueries method This method is called every time the context is updated. As written in the previous chapter, we decided to opt for a 30 seconds update timing, but this parameter could be easily changed in the configuration files.

executeQueries method This method simply launches the queries to the Overlay in order to receive the results.

executeQueries method This method is called every time a new result from the Overlay is received in order to know if the responding node could be interesting to the peer. *Being interesting* would simply means that in our particular case of geographical context, both the *latitude* and the *longitude* have been returned from the Overlay and collected in their specific container (*nearbyLatitudes* and *nearbyLongitudes*). If so, the node is added to the *candidatedNearbyNodes* list till the next context update. When a new candidate node has been found, the peer can finally query the Overlay in order to know the node's information regarding its centroid position, its range, group name and timestamp. As soon as all these information are arrived to destination, if the node is inside the peer's range then the system starts the joining routine.

Listing 4.1: Joining request

```
double distance = calculateDistance(nearbyLatitudes.get(uci), nearbyLongitudes
     .get(uci), nearbyRanges.get(uci));
boolean areIntersectable = DCXPLocationHelper.areIntersectable(this.core.
     getRange(), new String[] { nearbyLatitudes.get(uci), nearbyLongitudes.get(
     uci) }, Long.parseLong(nearbyRanges.get(uci)));

if(areIntersectable && (distance < this.core.getRange())) {
        this.core.joinGroup(this.newGroupName, nearbyCoordinates.get(mainUCI))
            ;
}
```

4.1.2 GroupCore

Group Core class is responsible of the context updates. It runs on its own Thread in order to avoid synchronization's issues with the main application's thread.

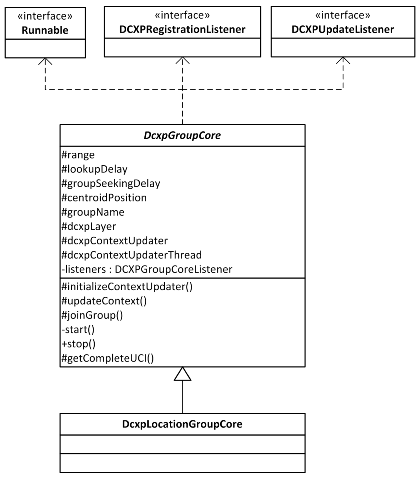


Figure 4.2: GroupCore UML Scheme

Its main task is to update the context accordingly to the time saved in the configuration file and in case of joining operations.

Listing 4.2: Normal context update operation

```java
LatitudeContextAttribute newLatitude = new LatitudeContextAttribute(
    getCompleteUCI("latitude"), getCompleteUCI("latitude"), currentPosition
    [0]);
LongitudeContextAttribute newLongitude = new LongitudeContextAttribute(
    getCompleteUCI("longitude"), getCompleteUCI("longitude"), currentPosition
    [1]);
RangeContextAttribute newRange = new RangeContextAttribute(getCompleteUCI("
    range"), getCompleteUCI("range"), String.valueOf(getRange()));

Log.getInstance().log(Level.INFO, "Updating␣group's␣information");
this.dcxpLayer.update(newLatitude, this);
this.dcxpLayer.update(newLongitude, this);
this.dcxpLayer.update(newRange, this);
```

Listing 4.3: Group's updated information retrieving and updating

```java
CentroidLatitudeContextAttribute newCentroidLatitude = new
    CentroidLatitudeContextAttribute(getCompleteUCI("centroidlatitude"),
    getCompleteUCI("centroidlatitude"), centroid[0]);
CentroidLongitudeContextAttribute newCentroidLongitude = new
    CentroidLongitudeContextAttribute(getCompleteUCI("centroidlongitude"),
    getCompleteUCI("centroidlongitude"), centroid[1]);
TimestampContextAttribute timestampAttribute = new TimestampContextAttribute(
    getCompleteUCI("timestamp"), getCompleteUCI("timestamp"), ((Long) Calendar
    .getInstance().getTimeInMillis()).toString());

Log.getInstance().log(Level.INFO, "Updating␣group's␣attributes␣from␣more␣
    recent␣peer's␣info.");
this.dcxpLayer.update(newCentroidLatitude, this);
this.dcxpLayer.update(newCentroidLongitude, this);
this.dcxpLayer.update(timestampAttribute, this);
```

Listing 4.4: Group's joining with consequent updating of the info

```
String[] currentPosition = dcxpContextUpdater.getNewContext();
String[] positionWithRange = new String[] { currentPosition[0],
    currentPosition[1], String.valueOf(range) };
String[] calculatedPosition = DCXPLocationHelper.mergeCentroids(
    positionWithRange, otherGroupPosition);

DCXPContextAttribute newGroupNameAttribute = new GroupNameContextAttribute(
    this.mainUCI + "/group", this.mainUCI + "/group", groupName);

CentroidLatitudeContextAttribute newCentroidLatitude = new
    CentroidLatitudeContextAttribute(getCompleteUCI("centroidlatitude"),
    getCompleteUCI("centroidlatitude"), calculatedPosition[0]);
CentroidLongitudeContextAttribute newCentroidLongitude = new
    CentroidLongitudeContextAttribute(getCompleteUCI("centroidlongitude"),
    getCompleteUCI("centroidlongitude"), calculatedPosition[1]);
TimestampContextAttribute timestampAttribute = new TimestampContextAttribute(
    getCompleteUCI("timestamp"), getCompleteUCI("timestamp"), ((Long) Calendar
    .getInstance().getTimeInMillis()).toString());

Log.getInstance().log(Level.INFO, "Setting␣new␣group␣attributes");
this.dcxpLayer.update(newGroupNameAttribute, this);
this.dcxpLayer.update(newCentroidLatitude, this);
this.dcxpLayer.update(newCentroidLongitude, this);
this.dcxpLayer.update(timestampAttribute, this);
```

4.1.3 Context Updater

ContextUpdater is the interface behind which is possible to create the context retrievers. Specifically, for the current project it wasn't possible to do some tests in a real environment, therefore a *mockup* context creator have been developed. Specifically, the *DCXPLocationHelper* simply simulates a person that moves in the environment with its own speed and direction, enclosed in the specified latitude and longitude coordinates.

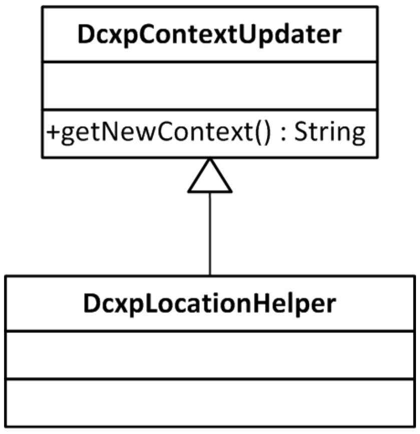


Figure 4.3: GroupLayer UML Scheme

4.2 Sensor Sockets

Sensor Sockets are the generic wrapping layer of the second part of this work. How it's possible to see in Figure 4.4, they are simple objects responsible of creation and the high level management of the underlying SCTP sockets.

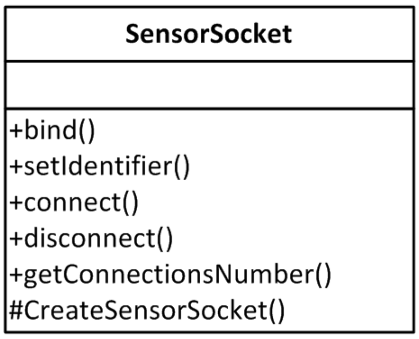


Figure 4.4: Sensor Socket UML scheme

They presents the common APIs of a connectivity socket such as *bind*, *connect* and *disconnect*. Particularly interesting is the second one. In fact, at the moment of the connection to another peer, the system returns an object of type *SensorDataStream* that is responsible of all the future communications between the two peers.

4.2.1 Sensor Data Stream

SensorDataStream object is created when a connection is established between two peers and is responsible for the management of the communication's channel. All the information for the channel are contained in the *Peer* object passed as parameter to the constructor. In the specific case of the SCTP, it already provide an *SctpChannel*. As written in the theory chapter, each SCTP channel is bi-directional, therefore it needs to be both read and write simultaneously. Fortunately, a common SCTP socket already provides a synchronization system in case of data collision.

In order to write data in the channel, four different routines are provided: *write* simply accept a *byte* array (or eventually a *ByteBuffer* or a single byte) as input and directly writes data. Then it's possible to send plain text using the *writeText* and finally two more for multimedia have been created: *writeFile* and *writeMedia*. The difference is that the first don't need any particular optimization, this is due to the fact that files are not streamed and they need only reliability for the amount of data sent. Instead, *writeMedia* necessitates of particular attentions for the amount of data sent in each message and the size of the data buffer used both for the transmission and

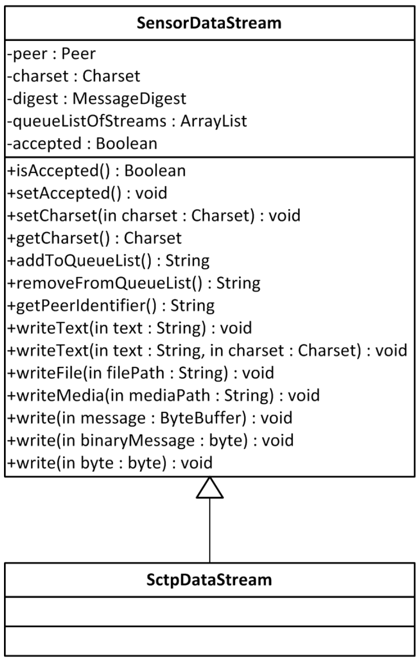

SensorDataStream
-peer : Peer
-charset : Charset
-digest : MessageDigest
-queueListOfStreams : ArrayList
-accepted : Boolean
+isAccepted() : Boolean
+setAccepted() : void
+setCharset(in charset : Charset) : void
+getCharset() : Charset
+addToQueueList() : String
+removeFromQueueList() : String
+getPeerIdentifier() : String
+writeText(in text : String) : void
+writeText(in text : String, in charset : Charset) : void
+writeFile(in filePath : String) : void
+writeMedia(in mediaPath : String) : void
+write(in message : ByteBuffer) : void
+write(in binaryMessage : byte) : void
+write(in byte : byte) : void
SctpDataStream

Figure 4.5: Sensor Data Stream UML scheme

the reception. All these streaming functionality have not been implemented yet.

Finally, it's interesting to see how the streams are requested. In fact, all is done asynchronously using the asynchronous method *beginStreaming* of the *Peer* object. This method takes the transmission request in the form of a specifically formatted *XML* document that is created by the *SensorSocketXmlFactory*.

Listing 4.5: writeText routine of the SctpDataStream object

```
ArrayList<SensorSocketXmlDataStream> streams = new ArrayList<
    SensorSocketXmlDataStream>();
SensorSocketXmlDataStream newStream = new SensorSocketTextXmlDataStream(text.
    length(), charset.displayName());
newStream.setIdentifier(identifier);
streams.add(newStream);
```

4.2.2 Peer

The *Peer* object is an abstract class that contains and manages all the information regarding the current established connection between our peer and another. Specifically, inside the *SctpPeer* is contained the *SctpChannel* that is used for both the transmission and the reception of the data.

Particularly interesting are both the asynchronous routines for this two operations. The *beginStreaming* method executes in a different thread in order to don't block the main application's thread, and simply encodes the

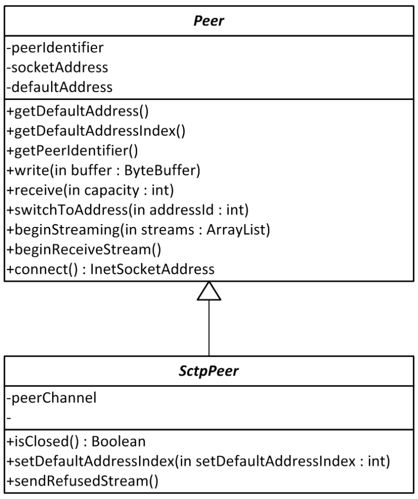

Peer
-peerIdentifier
-socketAddress
-defaultAddress
+getDefaultAddress()
+getDefaultAddressIndex()
+getPeerIdentifier()
+write(in buffer : ByteBuffer)
+receive(in capacity : int)
+switchToAddress(in addressId : int)
+beginStreaming(in streams : ArrayList)
+beginReceiveStream()
+connect() : InetSocketAddress
SctpPeer
-peerChannel
-
+isClosed() : Boolean
+setDefaultAddressIndex(in setDefaultAddressIndex : int)
+sendRefusedStream()

Figure 4.6: Peer UML scheme

data accordingly to the system *Charset* and then sends the transmission request. In fact, the protocol used for the data exchange is represented in Figure 4.7.

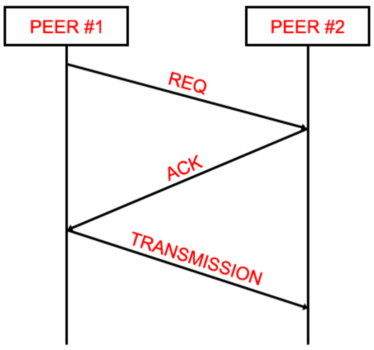


Figure 4.7: Transmission protocol used with the Sensor Sockets

Thus, the reception of the header acts accordingly to the protocol:

Listing 4.6: receiveStream routine of the Peer object

```java
Document doc = SensorSocketXmlDataStream.getXmlFromString(result);

Map<String, String> peerInfo = SensorSocketXmlDataStream.getPeerInfo(doc);
for (String peerInfoKey : peerInfo.keySet()) {
        if(peerInfoKey.equals("identifier"))
                peerIdentifier = peerInfo.get(peerInfoKey);
}

for (SensorSocketXmlDataStream streamInfo : SensorSocketXmlDataStream.
    getStreamInfo(doc)) {
        OnStreamOpen(streamInfo, peerIdentifier);
}
```

4.2.3 Sensor Socket Xml

In order to correctly send data and let the other peer understand what the current node's trying to send to it, the protocol represented in Figure 4.7 on the facing page has been proposed. However, all the data exchanged in both the request and acknowledgement operations is formatted in *XML* in order to keep it simple the future management of the different transmissions.

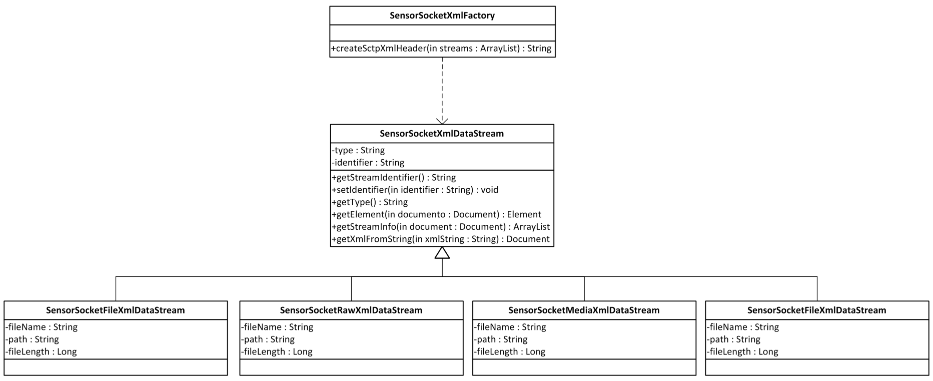

SensorSocketXmlFactory
+createSctpXmlHeader(in streams : ArrayList) : String
SensorSocketXmlDataStream
-type : String
-identifier : String
+getStreamIdentifier() : String
+setIdentifier(in identifier : String) : void
+getType() : String
+getElement(in documento : Document) : Element
+getStreamInfo(in document : Document) : ArrayList
+getXmlFromString(in xmlString : String) : Document
SensorSocketFileXmlDataStream
-fileName : String
-path : String
-fileLength : Long
SensorSocketRawXmlDataStream
-fileName : String
-path : String
-fileLength : Long
SensorSocketMediaXmlDataStream
-fileName : String
-path : String
-fileLength : Long
SensorSocketFileXmlDataStream
-fileName : String
-path : String
-fileLength : Long

Figure 4.8: Sensor Socket XML UML Scheme

In Figure 4.8 is represented the UML scheme of the objects whom takes care of the creation and the reading of the header. Specifically, the Factory is responsible for the creation of the *XmlDataStream* object whom contains the stream identifier and the type of the stream. Instead, the object itself is specifically implemented to generate itself accordingly to its nature, thus four different *XmlDataStream* sons inherit these functionalities. These are one for each writing routine created in the *Peer* object: *raw*, *file*, *media* and *text*.

4.2.4 Policy Manager

In order to manage the switch between two or more different connection, a *PolicyManager* object has been created. This *singleton* class is queried every predefined amount of time in order to know if a switch between the connections is required. This calculation could be done in different ways. Accordingly to the GSM handover explained in the theory chapter, it's possible to understand whether the first connection is loosing too much strength and the second is increasing its power and then decide to start the handover routine. However this is not the only possible solution, because in some cases a signal power loss do not necessarily means that an handover is required. Then we decided to leave the *PolicyManager* as more general as possible, and all the decisions are made instantiating the specific policies needed for the current use-case.

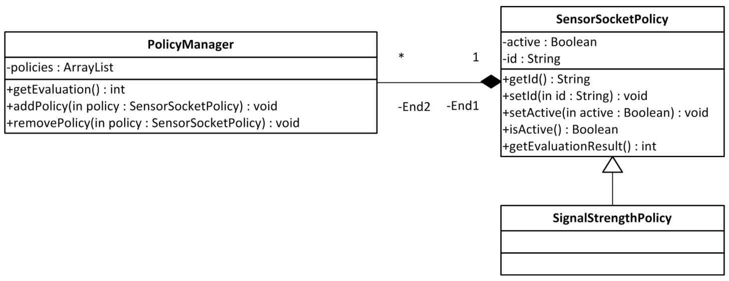

PolicyManager
-policies : ArrayList
+getEvaluation() : int
+addPolicy(in policy : SensorSocketPolicy) : void
+removePolicy(in policy : SensorSocketPolicy) : void
*
1
-End2
-End1
SensorSocketPolicy
-active : Boolean
-id : String
+getId() : String
+setId(in id : String) : void
+setActive(in active : Boolean) : void
+isActive() : Boolean
+getEvaluationResult() : int
SignalStrengthPolicy

Figure 4.9: PolicyManager UML Scheme

In fact, how it's possible to see in the evaluation's method, the given result is nothing more than the connection whom received more votes from each policy singularly queried:

Listing 4.7: getEvaluation's method of the PolicyManager

```java
public int getEvaluation() {
	try {
		int maxId = 0;
		int max = 0;
		int[] evaluations = new int[NetworkManager.getInstance().
		    getNumberOfConnections()];

		for (SensorSocketPolicy policy : policies) {
			if(policy.isActive())
				evaluations[policy.getEvaluationResult()]++;
		}

		for (int i = 0; i < evaluations.length; i++) {
			if(evaluations[i] > max)
				maxId = i;
		}

		return maxId;
	} catch (SocketException e) {
		Log.LogError("Impossible␣to␣evaluate␣a␣new␣connection", e);
	}
	return 0;
}
```

Assuming that the *connection 1* is selected by two policies and the *connection 2* is selected by only one policy, then the routine simply returns the first because more policies decide that it was the best connection currently active in our environment.

4.2.5 Network Manager

Network Manager is a *singleton* class with the responsibility of manage the current peer connections.

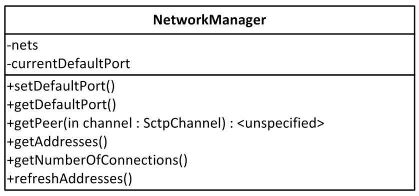


Figure 4.10: Network Manager UML Scheme

It's functionalities are limited to simply offer a friendly view of how the underlying network is organized.

4.2.6 Connection Manager

Connection Manager is the most important *singleton* class of the second part of the project. In fact, it's responsible of the main coordination of the data transmission and reception of the current node. It offers the *API*s used by the overlying *SensorSocket* object to which is implicitly related.

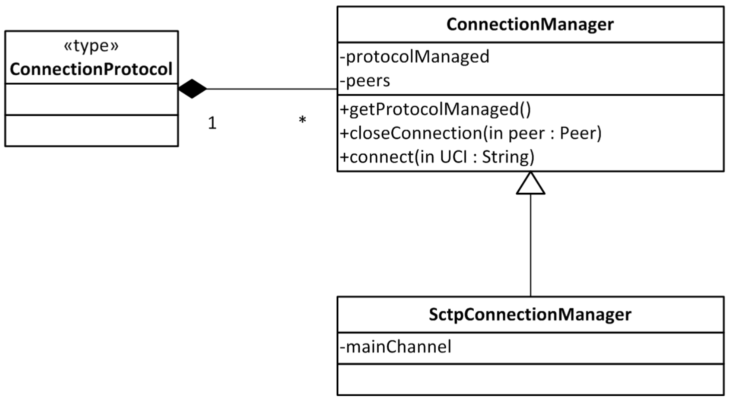


Figure 4.11: Connection Manager UML Scheme

It's important to notice that the Connection Manager is not directly connected to the SCTP protocol. In fact, it's possible to specify the typology of connection used passing the correct value of *ConnectionProtocol* to the *getInstance* method.

4.3 Model's behavior

In order to verify the behavior of the algorithm in a real network, we decided to take some simulations in a safe environment inside the University's laboratories. Anyway, before taking these tests in a real network, we made some in a simulated environment and took the opportunity to simulate it under several conditions and with some disturbing agents that changed the balance of the system, due to the fact that we wanted to see the robustness of the model.

4.3.1 Simulations in OMNeT++ and Oversim

Simulations were took inside the OMNeT++ simulator and Oversim [4]. The time of every simulation have been set to 500 seconds, with a TTL of every node set to a variable number between 200 and 240 seconds for the 30% of nodes and infinite for the remaining 70%. The discovery range have been set to a variable number up to 50 units. All the statistics (that will be attached to this thesis) have been calculated on a single machine with a 3GHz CPU with 8 GB of RAM.

This is the INI file that describes how were organized the simulations:

Listing 4.8: The organization of the simulations

```
[Config MediaSenseChordDHT]
description = Chord DHT (SimpleUnderlayNetwork)
*.underlayConfigurator.churnGeneratorTypes = "oversim.common.LifetimeChurn"
**.lifetimeMean = 10000s
**.measurementTime = 500s //tempo della simulazione
**.transitionTime = 100s
**.overlayType = "oversim.overlay.chord.ChordModules"
**.numTiers = 2
**.tier1Type = "oversim.applications.dht.DHTModules"
**.tier2Type = "oversim.tier2.msTestApp.DHTTestAppModules"
**.globalObserver.globalFunctions[0].functionType = "oversim.tier2.msTestApp.
    GlobalDhtTestMap"
**.globalObserver.numGlobalFunctions = 1
**.targetOverlayTerminalNum = N //variabile in base alla simulazione
**.initPhaseCreationInterval = 0.1s
**.debugOutput = true
**.drawOverlayTopology = true
**.tier1*.dht.numReplica = 4
```

The Overlay is based on a CHORD-DHT protocol, as shown in Figure 4.12 on the facing page.

Tier 2 uses the standard KBR (Key-based routing) API for the insertion of all the data inside DHT and also for the discovery of the Top nearest-neighbour.

In fact, the dataset of every node (that is distributed and replicated inside the Overlay) is composed by a $\mathfrak{R}^n$, where n is the number of data saved inside the Overlay for every node. In our specific case we find:

- 2D coordinates of the centroid
- Group of the node
- **2D position of the node**

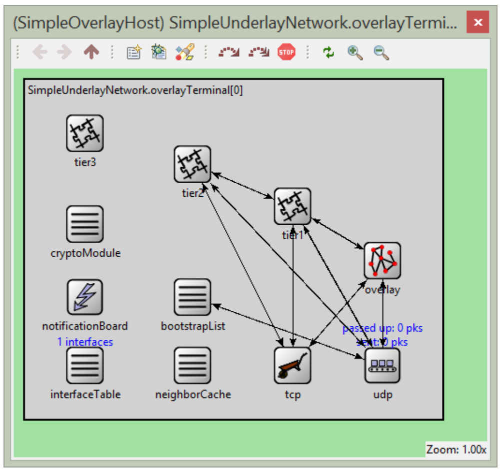


Figure 4.12: Single node

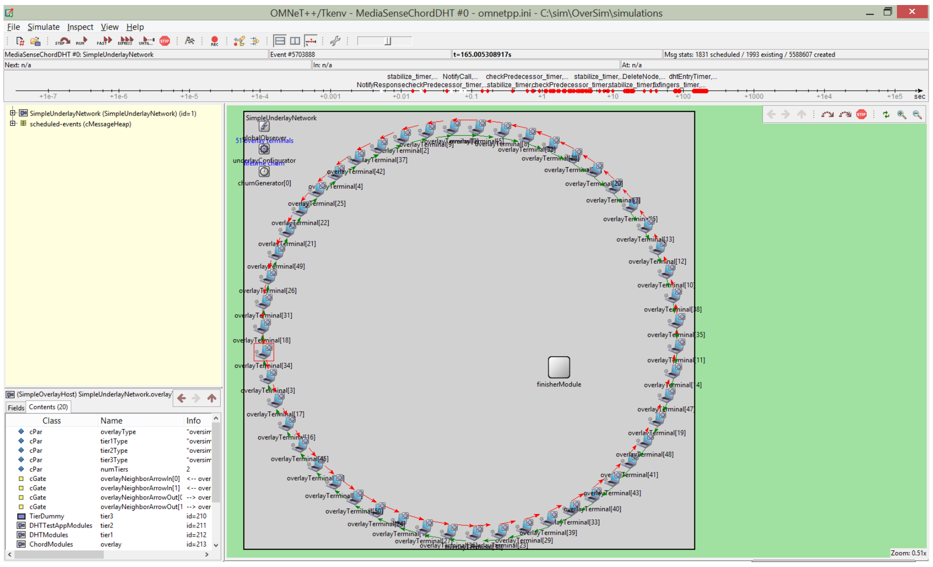
Is easy to notice that there could be a variable number of information (emphasized in bold) that depend only by the scenario of the simulation/use, and others that instead maintain their persistance since they are part of the model. We used the 2D coordinates of the node to simplify the effect, but nothing prevents the use of several context information, such as music preferences, user data, age and so on; thus, what could be changed only the quantity of data.

4.3.2 Differences between CHORD/DHT and MediaSense's P-Grid

MediaSense was born as a framework for the structured insertion of data inside a P2P network based on Chord and DHT protocols. The necessity of using the Range Queries brought the researchers to the use of P-Grid instead, that permits also the absence of a load-balancing algorithm and a bootstrap node (P-Grid trie is self-balanced and bootstrap-less).

Owever, OMNeT++ and Oversim don't provide the use of P-Grid in the simulations, so we decided to use a modified versione of Chord and DHT protocols in order to use the Range Queries during the whole test session.

We underline the fact that the test environment is basically different to the ideal MediaSense Framework, anyway there is a specific relation between the two, and is explained in the following lines: every node collects its context-related data and insert it inside the Overlay using a unique key for the DHT protocol. These information are not responsible for the increasing of the overhead since they would be there anyway. The only packets that could inficiate the scalability of the P2P network are those related to the discovery session, in which each node sends a lookup packet to the Overlay



OMNeT++/Tkenv - MediaSenseChordDHT #0 - omnetpp.ini - C:\sim\OverSim\simulations

MediaSenseChordDHT #0: SimpleUnderlayNetwork | Event #5703888 | t=165.005308917s | Msg stats: 1831 scheduled / 1993 existing / 5588607 created

Next: n/a | In: n/a | At: n/a

stabilize_timer,... NotifyCall,... checkPredecessor_timer,... stabilize_timer,..DeleteNode,... dhtEntryTimer,...
NotifyResponsecheckPredecessor_timer,...stabilize_timercheckPredecessor_timerstabilize_timerfixfingers_timer,...

SimpleUnderlayNetwork (SimpleUnderlayNetwork) (id=1)
scheduled-events (cMessageHeap)

(SimpleOverlayHost) SimpleUnderlayNetwork.overlay

Fields | Contents (20)

Class	Name	Info
cPar	overlayType	"oversim
cPar	tier1Type	"oversim
cPar	tier2Type	"oversim
cPar	tier3Type	"oversim
cPar	numTiers	2
cGate	overlayNeighborArrowIn[0]	<-- over
cGate	overlayNeighborArrowIn[1]	<-- over
cGate	overlayNeighborArrowOut[(	--> over
cGate	overlayNeighborArrowOut[1	--> over
TierDummy	tier3	id=210
DHTTestAppModules	tier2	id=211
DHTModules	tier1	id=212
ChordModules	overlay	id=213

Figure 4.13: Oversim with 50 nodes simulating

and it expects an answer with the K-most nearest neighbor. We know the performance of this discovery, because it has been tested in previous works, and it is

$$0.5 * log(N)$$

where N is the number of nodes inside the tree.

Thus, taking our simulations as an example, the only information that exceed this "normal" situation, is the data related to the centroid and the group identifier. However, the size of this data is relatively small, since we expect only some more bytes to what is normally exchanged between the nodes. Our goal is to see if this data is responsible to any change in the scalability of the P2P network.

4.3.3 Tests goal

As already written, the goal of these tests is to find a relationship between the density of the nodes in the context-space and the number of iterations needed to have a stable situation (no more group aggregation). Moreover, we calculated also the number of exchanged discovery packets (to and from the Overlay) in every test.

To avoid any error related to the duration of the tests, this has been maintained to a very long constant time (500 seconds), the number of nodes have been parameterized and a disturbing agent has been inserted to verify the robustness of the model in case of the failure of some nodes; this element can clearly be seen from the time 200 sec.

Nodes density	Avg It/Nodes	Groups
0.1	6.87	43
0.1	10.18	6
0.05	7,04	4
0.02	18,94	1
0.01	22,67	2

Table 4.1: Relationship between density and iterations

Every 10 seconds we calculated the number of groups inside the Overlay.

4.3.4 Results with nodes inserted into a uniform-distributed space

We made 5 similar tests and a one more at the end:

- AREA 1000 - 10 nodes = low density
- AREA 1000 - 20 nodes = mid-low density
- AREA 1000 - 50 nodes = mid density
- AREA 1000 - 100 nodes = high density
- AREA 10000 - 1000 nodes = high density

These tests helped us to verify the algorithm in a simulated environment, confirming the results we obtained with the MATLAB simulations, that is a progressive gain of efficiency in terms of number of iterations the more the density of the nodes. In fact, the more high the density is, the less number of packets are exchanged to group the nodes. Results are in Table 4.1 and in Figure 4.14 on the following page.

In Figure 4.15 on the next page, 4.16 on page 89, 4.17 on page 89, 4.18 on page 90, and 4.19 on page 90 is possible to see all the results.

It's possible to notice that the algorithm is effective from the beginning of the simulation. In fact, in the first 10 seconds there is a big selection of the predominant groups up to an half of the initial number (every node belongs to a different group at the beginning).

At the introduction of the disturbing element (at 200seconds), every node starts a new discovery session and changes its group to the one closer until a new stable situation.

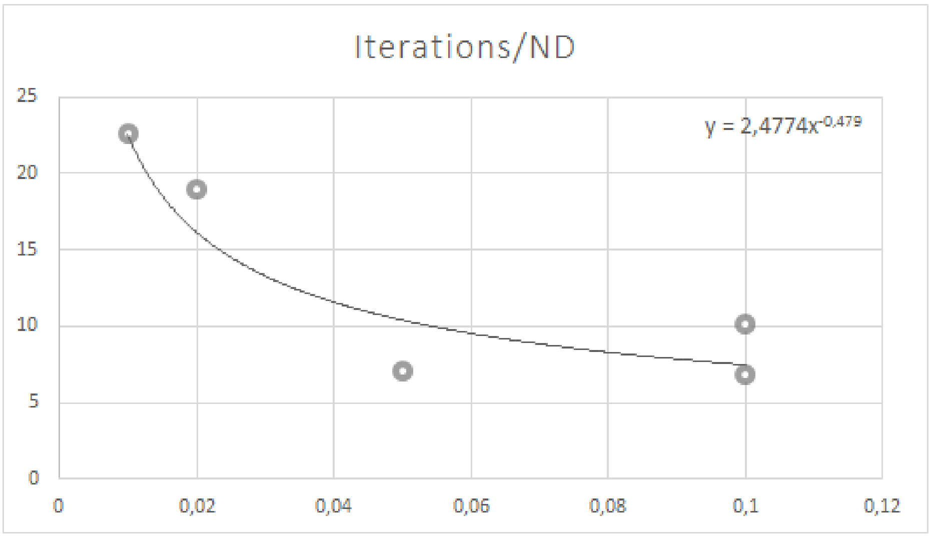


Figure 4.14: Relationship chart between density and iterations

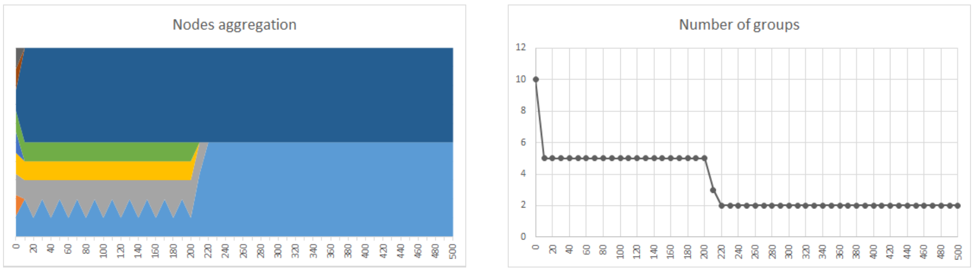


Figure 4.15: Area 1000 - 10 nodes

A typical-use environment's test

After the results obtained we decided to test the model in a more realistic context as could be a typical sitting room: the nodes (appliances) in this situation are in a more static environment, but some of them (tablets, mobile phones, notebooks and so on) could change more often, we decided to reset the groups every 20 seconds.

In Figure 4.20 on page 91 is possible to see a graphical representation of the context. The room has been subdivided in a 2D plan and we assigned a triplet of values to each appliance:

$$PosX, PosY, Radius$$

In this way, our triplets are represented in Table 4.2 on page 90.

The parameters pf this simulation are the following:

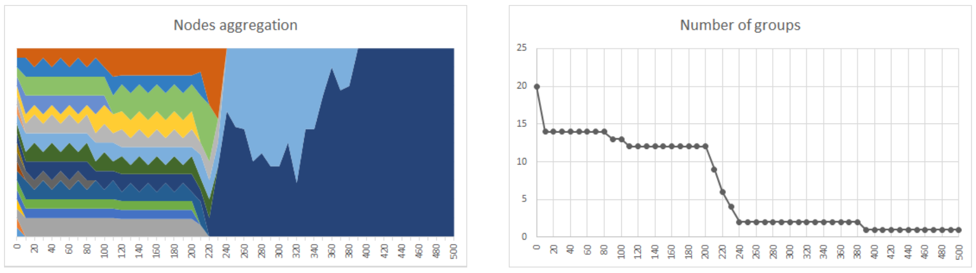


Figure 4.16: Area 1000 - 20 nodes

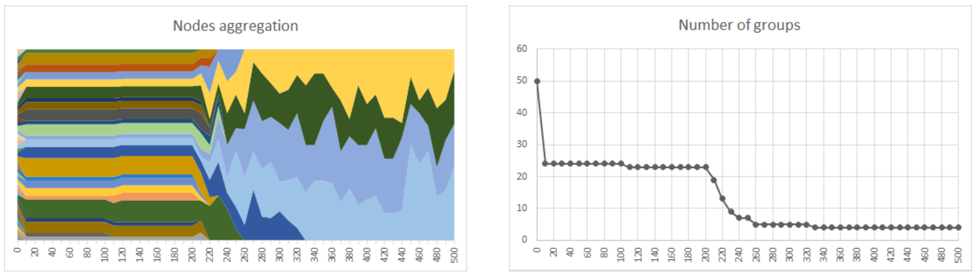


Figure 4.17: Area 1000 - 50 nodes

Listing 4.9: The organization of the real simulation

```
[Config MediaSenseChordDHT]
description = Chord DHT (SimpleUnderlayNetwork)
*.underlayConfigurator.churnGeneratorTypes = "oversim.common.LifetimeChurn"
**.lifetimeMean = 10000s
**.measurementTime = 15s
**.transitionTime = 100s
**.overlayType = "oversim.overlay.chord.ChordModules"
**.numTiers = 2
**.tier1Type = "oversim.applications.dht.DHTModules"
**.tier2Type = "oversim.tier2.msTestApp.DHTTestAppModules"
**.globalObserver.globalFunctions[0].functionType = "oversim.tier2.msTestApp.
    GlobalDhtTestMap"
**.globalObserver.numGlobalFunctions = 1
**.targetOverlayTerminalNum = 8
**.initPhaseCreationInterval = 0.1s
**.debugOutput = true
**.drawOverlayTopology = true
**.tier1*.dht.numReplica = 4
```

For compatibility reasons, the coordinates have been expressed in decimeters, and the groups are represented in prime numbers (since the moltiplication of two prime numbers is unique). The log is in the Appendix.

In Figure 4.21 on page 91 is possible to see the relationship between the nodes and the groups.

Firstly, is possible to notice that two groups have been created since the first iteration, and then they permained till the end (in this simulation, we had no disturbing elements).

With such a behavior, is possible to dinamically create group of objects

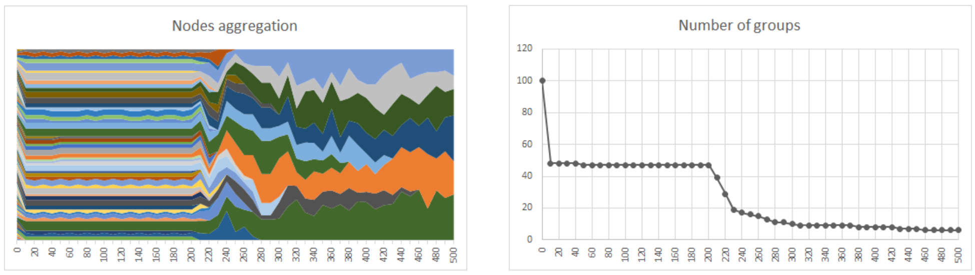


Figure 4.18: Area 1000 - 100 nodes

Node ID	X	Y	Radius
1 - Ceiling lamp	2.0	2.0	0.5
2 - Sound system	3.3	2.0	1.0
3 - TV	3.6	1.5	1.0
4 - Fan	3.5	0.4	1.0
5 - Tablet	2.2	0.5	0.8
6 - Notebook	1.4	0.5	0.8
7 - Mobile phone	1.2	1.6	0.8
8 - Abat-jour	3.5	3.0	0.5

Table 4.2: Appliances' triplets

not related each other (without an initial pairing), even if they belongs to different networks and that they can dinamically distribute theirselves in self-contained areas (clusters) in such a way that it's possible to send them some remote commands (such as to switch on and off single appliances or group of appliances), only thanks to their spatial localization.

4.4 Running the MediaSense Framework

I decided to write this paragraph as help for the future researchers who will work on the MediaSense Framework's project. In fact, at a first run, it was quite impossible to let the P-Grid to structure itself. Firstly, we tried using one private node as the bootstrap and then to run all the other nodes in the corresponding network machines. We found that even if the machines were

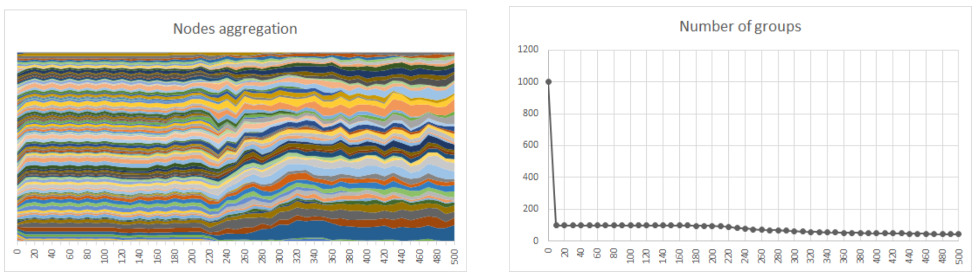


Figure 4.19: Area 10000 - 1000 nodes

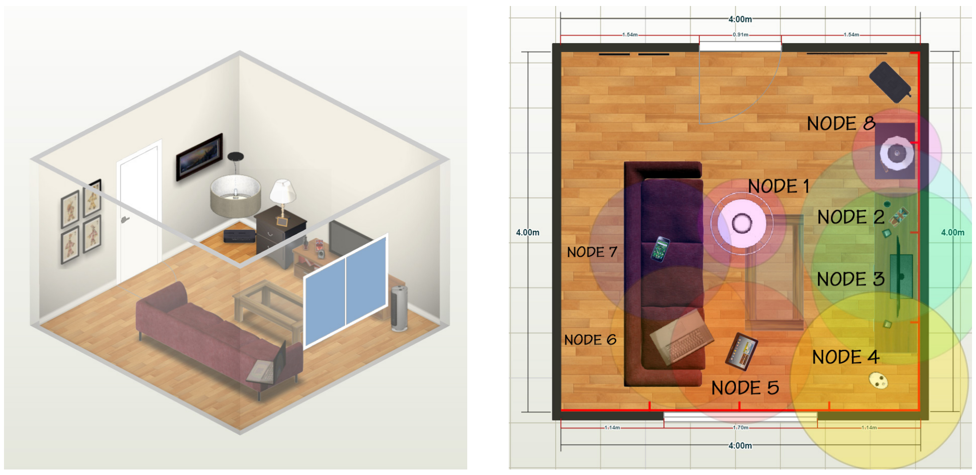


Figure 4.20: Our realistic environment

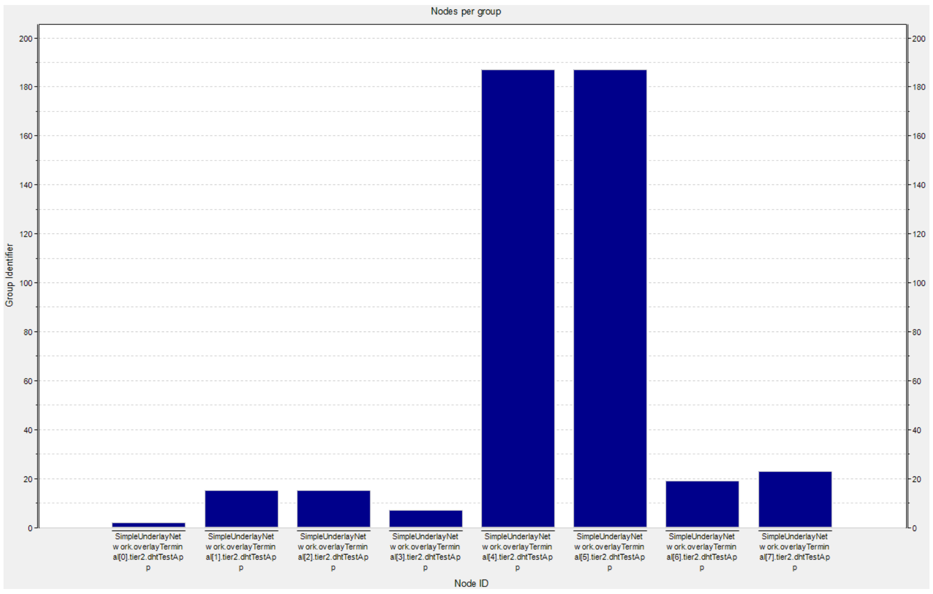


Figure 4.21: Relationship between the nodes and the groups

all running on the same network, they didn't see each other: actually they pinged each other, but with no answer in the protocol. Then we decided to remove all the possible network-related barriers, like firewalls and anti-viruses. This didn't solve the problem. Afterward we tried to see whether the problem was actually related to the topology of the network interfaces, and especially with the number of currently active networks on the machines. We found that the protocol just binds on the first-found interface, with no possibilities to force the bind on a specific interface. So we decided to check inside the protocol's *Core Layer* in order to find if it was possible to change its behavior. Actually, we found that the decision was totally static and with no chance of dynamically set the node's preferences, and we put some changes. First of all, we integrated our SensorSocket version, that is totally network protocol independent. In this way, none of the network related problem should occur anymore. Moreover, thanks to the SCTP protocol, it's totally unimportant whether the node stays on a specific node rather than another. Anyway, we switched off all the networks that were not related to the tests. After running the Framework on the bootstrap and on the nodes, we noticed that the nodes were currently communicate each other, but there was no structuring in the P-Grid. Actually, this has been the hardest point regarding the firsts testing weeks. Anyway, at the end we decided to use a publicly reachable bootstrap node, and that solved the problem.

4.5 Grouping

Tests for the grouping algorithm has been made inside the Network Laboratory of the University, and specifically we used 5 machines with administrative accounts. The reason of having administrative credentials is linked to the fact that current PGrid message exchange and DCXP routing protocols don't work under limited accounts with firewall or proxies between them.

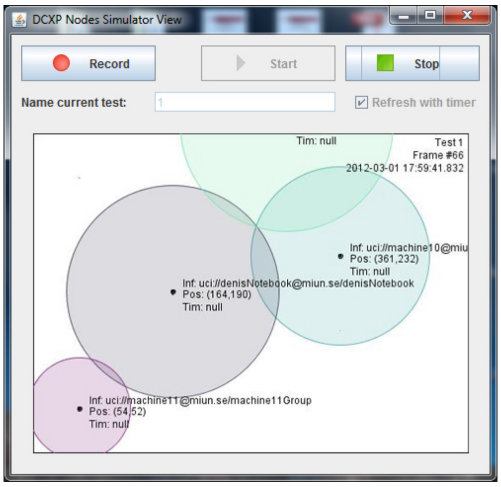


Figure 4.22: Simulator View

On the first machine then PGrid's bootstrap node has been started in order to avoid possible firewall conflicts with an external one. On a notebook we started a node in DEBUG mode in order to see the information's flux and to debug the possible errors. On two machines we ran the Media Sense nodes, and finally on the last machine we run both a normal MediaSense node and the Simulator View.

4.5.1 Simulator View

The Simulator View is a simple Java executable file that is able to listen to the *UDP* port number *52600* in order to receive specifically formatted packets containing the node's information. The executable shall be ran on a publicly visible machine in order to let it receive all the packets correctly. In fact, inside each packet are contained the current context information of the node (specifically, in our case they are the geographical coordinates in terms of X and Y position), the research range with the current centroid coordinates and finally the node name and belonging group's name. The packets are sent as plain text, so there is any kind of formal format of the data:

```
    uci,groupname|latitude,longitude|range|
centroidLatitude,centroidLongitude|timestamp|
```

Clearly, to create a more sophisticated way for exchange the data would be better but it was out of our goals.

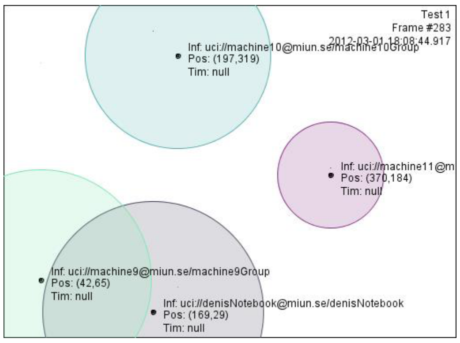
4.5.2 Grouping algorithm

The tests regarding the grouping algorithm are based on the assumption that the node are not moving for real and their position and speed data are totally simulated. Times for position acquisition are set on 10 seconds for each node: after this time, each node starts the lookup for the other nodes.

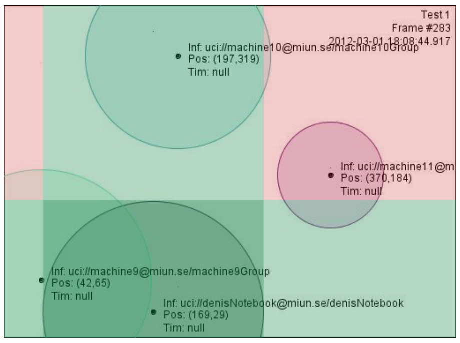
Algorithm's first stage - Nearby node lookup The first stage of the grouping algorithm regards the lookup on the Overlay in order to find out the surrounding nodes positioned inside the quadrangle figure formed with both the latitude and longitude coordinates and the range (see Figure 3.13 on page 69). As stated in the fourth chapter, the lookup is done using the *Range Queries* provided by the Overlay.

In Figure 4.23 on the following page it's possible to see both the normal view of the nodes in the network and the current zone of interests of node called *denisNotebook*.

Algorithm's second stage - Node selection As stated before, after the lookup the nearby nodes are sorted in order to find the best match. In this case, two of the nodes are inside the ranges but only one matches with both the latitude and the longitude, therefore it's the only one that will be queried by the node in order to know its centroid position.



(a) Normal view



(b) Overlayed view with zones of interest for the node

Figure 4.23: Algorithm's first stage - Nearby node lookup

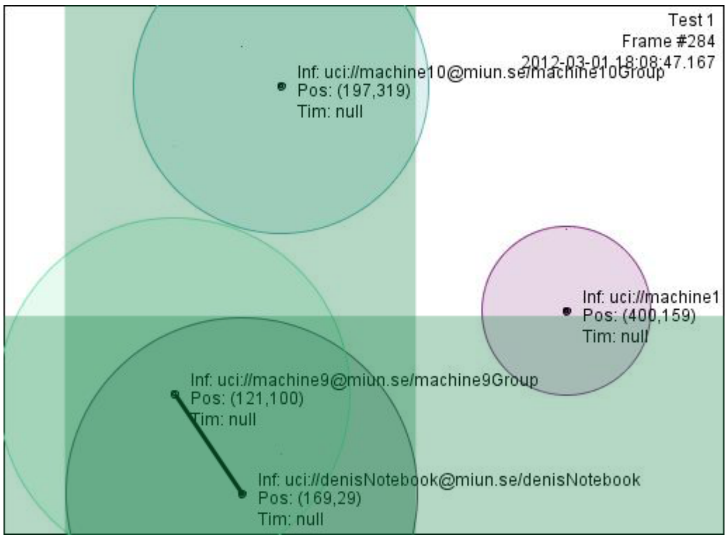


Figure 4.24: Node selection

Algorithm's third stage - Group joining After the selection of the node with whom join into a group, the node queries that node asking it some more information such as its centroid position and the name of the group. This is done in order to minimize the quantity of message exchanged during the first stage of the algorithm.

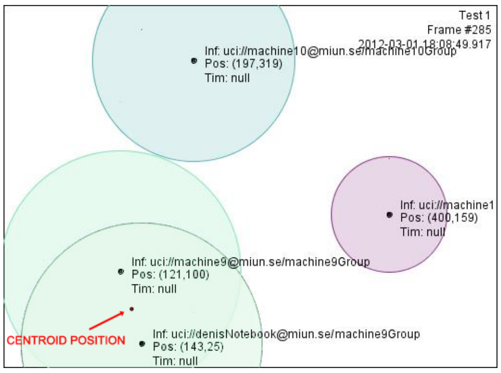


Figure 4.25: Group joining

As soon as all the information are arrived to destination, the node correctly changes its own properties and updates the Overlay with the new values.

4.5.3 Multiple node lookup

In case of multiple node resolution for the peer, the system should correctly understand which of the two is the closest in order to create a group with it. In Figure 4.26 is possible to see that the algorithm correctly understand the right node to choose.

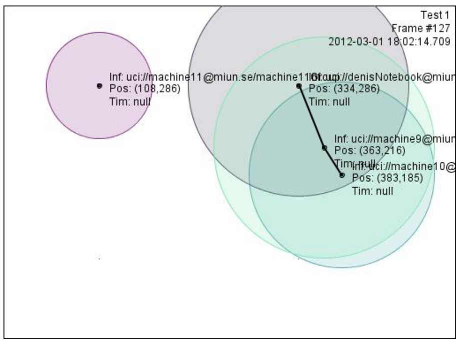


Figure 4.26: Multiple node resolution example

4.5.4 Centroid repositioning

When a peer join a group, the centroid accordingly changes its position basing on the position of the nodes belonging to the group. However, due to the passive-nature of the Overlay, the right position of the centroid is saved only in the last node that joined the group and this means that all the other nodes would need an update of the new position. This update is done during the lookup: whether a node finds another node from the same group during the lookup operation, it asks the timestamp of the centroid and if it's more recent than the other, then updates its own value.

4.6 Testing problems

During the tests some problems have been detected on the Overlay. In fact, in some cases the *range queries* have been resolved with both *false positive* and *false negative* results. This means that some nodes have been returned as good nearby choices even if their position and range were far to the peer that queried the Overlay. Some hypothesis has been formulated in order to answer this problem:

1 The problem could be related to the querying rate: querying the system too fast from different position could create some mistakes in the data persisting and therefore on the routing of the requests;

2 Range queries' routines contains errors;

3 Range queries' routines needs some more specific tests.

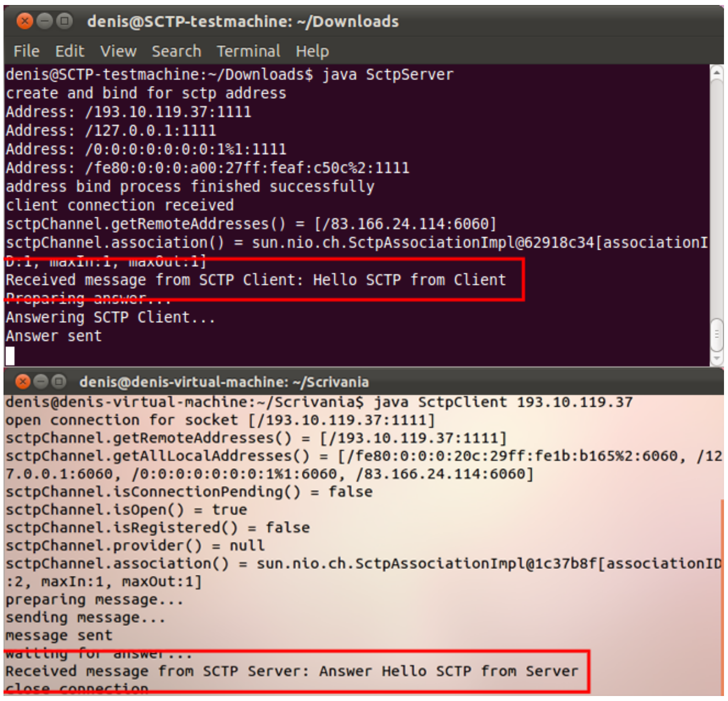
Accordingly to the results of the previous works, the second hypothesis shall be excluded. Thus, we can conclude that the current work is promising but needs further tests in order to understand its real speed capabilities under pressure.

4.7 Mobility management

4.7.1 SCTP over firewall test

To be completely aware of possible future improvements, firstly we needed to test if the SCTP protocol can be used through the Internet. For this reason, and also for testing purposes that have been listed in the first chapter, we mounted a virtual server with a public accessible IP address. A standard *Linux Ubuntu 10.10* distribution has been used on the server. The first test that has been made is the possibility of SCTP to pass the packet filtering system of the current firewalls.

In Figure 4.27 on the facing page is possible to see that SCTP effectively works over the Internet between two distant hosts.


denis@SCTP-testmachine: ~/Downloads
File Edit View Search Terminal Help

```
denis@SCTP-testmachine:~/Downloads$ java SctpServer
create and bind for sctp address
Address: /193.10.119.37:1111
Address: /127.0.0.1:1111
Address: /0:0:0:0:0:0:0:1%1:1111
Address: /fe80:0:0:0:a00:27ff:feaf:c50c%2:1111
address bind process finished successfully
client connection received
sctpChannel.getRemoteAddresses() = [/83.166.24.114:6060]
sctpChannel.association() = sun.nio.ch.SctpAssociationImpl@62918c34[associationI
D:1, maxIn:1, maxOut:1]
Received message from SCTP Client: Hello SCTP from Client
Preparing answer...
Answering SCTP Client...
Answer sent
```

```
denis@denis-virtual-machine:~/Scrivania$ java SctpClient 193.10.119.37
open connection for socket [/193.10.119.37:1111]
sctpChannel.getRemoteAddresses() = [/193.10.119.37:1111]
sctpChannel.getAllLocalAddresses() = [/fe80:0:0:0:20c:29ff:fe1b:b165%2:6060, /12
7.0.0.1:6060, /0:0:0:0:0:0:0:1%1:6060, /83.166.24.114:6060]
sctpChannel.isConnectionPending() = false
sctpChannel.isOpen() = true
sctpChannel.isRegistered() = false
sctpChannel.provider() = null
sctpChannel.association() = sun.nio.ch.SctpAssociationImpl@1c37b8f[associationID
:2, maxIn:1, maxOut:1]
preparing message...
sending message...
message sent
waiting for answer...
Received message from SCTP Server: Answer Hello SCTP from Server
close connection
```

Figure 4.27: First Client/Server test over the Internet

CHAPTER

FIVE

CONCLUSIONS

MediaSense Framework is probably one of the most promising technologies for the management of future networks. Anyway it's still young and need further improvements in order to become a standard. What studied so far is only a little part of the work and for sure the future students will appreciate it's ease of use. However, regarding our particular case, in this last chapter we will discuss about what have been done so far.

Some improvements are required by the current work. For first, the system needs some finishes in order to be adopted in the future. Particularly, the Overlay has not been tested yet for its whole potentials, and it would be useful to understand how many queries could be done in a crowdy network with many concurrent queries. This kind of test are important to extrapolate the correct values for the groups management in order to be sure to limit the number of false positive and negative of the system.

Initially, our main goal has been to simply create groups on a large amount of network nodes. However, this goal required further work in order to succeed in it. Actually, the first problem found has been how to manage the mobility handover between different connections. The solution has been the use of the SCTP protocol, a new promising network standard that implements both the qualities of *TCP* and *UDP*. This permits to manage different connection concurrently thanks to the low weight of the SCTP packets and unordered data delivery (typical of the UDP protocol), mixing reliable and best effort traffic and finally timed reliability, typical of the TCP protocol instead. The handover system created is easy to use because it works as a wrapper over the SCTP itself and gives to the user DataStream objects whom contains predefined APIs to send different types of data both in input and output. Secondly, the second problem faced has been the creation of groups over the MediaSense Framework. Different versions of the framework exists and some of them have been inspected in order to choose the best one. Finally, only one contained all the characteristics needed for our job, and particularly the possibility to create send range queries to the Overlay. The work done so far permitted to create dynamic groups inside a small amount of peers in the network without a big throughput, but it happened only with a small amount of nodes. In fact, as written in the previous chapter, we made tests with up to 5 active nodes. Therefore, more tests would be necessary to verify completely the real scalability and thus the capabilities of the system with a large amount of nodes.

APPENDIX A

Listing 1: The log of the simulation

```
$ cd C:/sim/OverSim/simulations
$ ../src/OverSim.exe -r 0 -n .;../src;../../inet/examples;../../inet/src --
    tkenv-image-path=../images -l ../../inet/src/inet omnetpp.ini
OMNeT++ Discrete Event Simulation (C) 1992-2013 Andras Varga, OpenSim Ltd.
Version: 4.5, build: 140714-c6b1772, edition: Academic Public License -- NOT
    FOR COMMERCIAL USE
See the license for distribution terms and warranty disclaimer
Setting up Tkenv...
Loading NED files from .: 0
Loading NED files from ../src: 114
Loading NED files from ../../inet/examples: 59
Loading NED files from ../../inet/src: 183
Loading NED files from C:\sim\inet\src: 183
Loading NED files from C:\sim\OverSim\src: 114
node id0 initialization:
node0 key => 500d81aafe637717a52f8650e54206e64da33d27
node1 key => f937c37e949d9efa20d2958af309235c73ec039a
node2 key => 2dbf44a68b77b15bfa5bc3d66c97892a57402bbe
node3 key => a46fe0c4dab0453f5d86bed6206040880f59393e
node4 key => 9da30539af3639c600c6256f7691750a581c36c2
node5 key => b0a69b1f9fe82d6c149179ce48e22f9c8411afe3
node6 key => 74e5a4bcab7355b8cab7df73d07747cd85c925e7
node7 key => c03e55d15602a33922858e97664ea33f368ef5de
Closest node entry found: 5
Closest node entry found: 3
Closest node entry found: 17
Closest node entry found: 11
Node 1 >> Received join proposal (cur groupId: 3 last: 3) posX: 36 posY: 15
    group: 5
Node 1 now is in group 15
Node 2 >> Received join proposal (cur groupId: 5 last: 5) posX: 33 posY: 20
    group: 3
Node 2 now is in group 15
Node 5 >> Received join proposal (cur groupId: 17 last: 17) posX: 22 posY: 5
    group: 11
Node 5 now is in group 187
Node 4 >> Received join proposal (cur groupId: 11 last: 11) posX: 14 posY: 5
    group: 17
Node 4 now is in group 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Closest node entry found: 15
Closest node entry found: 15
Closest node entry found: 187
Closest node entry found: 187
Simulation time: 1.028000
```

APPENDIX B

Listing 2: MSTestApp.cc

```
//
// Copyright (C) 2014 Facoltà di Ingegneria Informatica, Alma Mater Studiorum
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
//

/**
 * @file MSTestApp.cc
 * @author Denis Billi
 */

#include <IPAddressResolver.h>
#include <GlobalNodeListAccess.h>
#include <GlobalStatisticsAccess.h>
#include <UnderlayConfiguratorAccess.h>
#include <RpcMacros.h>
#include "CommonMessages_m.h"

#include <GlobalDhtTestMap.h>

#include "MSTestApp.h"
#include "c_tokenizer.h"

#include<sstream>

Define_Module(MSTestApp);

using namespace std;

```

```
MSTestApp::~MSTestApp() {
    cancelAndDelete(msLookup_timer);
    cancelAndDelete(joinMessage);
    cancelAndDelete(resetTimer);
    LUT_nodesKeys.clear();
}

MSTestApp::MSTestApp() {
    // MediaSense lookup Timer
    msLookup_timer = NULL;
    joinMessage = NULL;
    resetTimer = NULL;
}

void MSTestApp::initializeApp(int stage) {
    if (stage != MIN_STAGE_APP)
        return;

    // fetch parameters
    debugOutput = par("debugOutput");
    activeNetwInitPhase = par("activeNetwInitPhase");

    mean = par("testInterval");
    p2pnsTraffic = par("p2pnsTraffic");
    deviation = mean / 10;

    if (p2pnsTraffic) {
        ttl = 3600 * 24 * 365;
    } else {
        ttl = par("testTtl");
    }

    globalNodeList = GlobalNodeListAccess().get();
    underlayConfigurator = UnderlayConfiguratorAccess().get();
    globalStatistics = GlobalStatisticsAccess().get();

    globalDhtTestMap =
            dynamic_cast<GlobalDhtTestMap*>(simulation.getModuleByPath(
                    "globalObserver.globalFunctions[0].function"));

    if (globalDhtTestMap == NULL) {
        throw cRuntimeError("MSTestApp::initializeApp():␣"
                "GlobalDhtTestMap␣module␣not␣found!");
    }

    // statistics
    numSent = 0;
    numGetSent = 0;
    numGetError = 0;
    numGetSuccess = 0;
    numPutSent = 0;
    numPutError = 0;
    numPutSuccess = 0;

    simResetTime = 20;
    simLookupTime = par("simLookupTime");
    numSimNodes = par("simNodes");
    numActiveLookupReceived = 0;
    numLookupSent = 0;
    numLookupResponseReceived = 0;
    numJoin = 0;
    numIterations = 0;

    nodeId = globalDhtTestMap->generateNodeId();
    std::string nodeGroupStr("nodeGroup");
    std::string nodeGroupId= itos(nodeGroupStr, nodeId);
    groupId = par(nodeGroupId.c_str());
    initialGroupId = groupId;

    std::string nodeXStr("nodeX");
    std::string nodeXId = itos(nodeXStr, nodeId);
    posX = par(nodeXId.c_str());

    std::string nodeYStr("nodeY");
    std::string nodeYId= itos(nodeYStr, nodeId);
    posY = par(nodeYId.c_str());
    centroidX = posX;
    centroidY = posY;

    std::string nodeRangeStr("nodeRange");
    std::string rangeStrId = itos(nodeRangeStr, nodeId);
    range = par(rangeStrId.c_str());
    initialRange = range;

    // inizializzazione lookup table
    LUT_nodesKeys.reserve(numSimNodes*2);

    if (nodeId == 0)
        std::cout << "\nnode␣id" << nodeId << "␣initialization:\n";

    for (int i = 0; i < numSimNodes*2; ++i) {
        std::stringstream ss;
        ss << "node" << i;
        std::string node = ss.str();
        BinaryValue binValue(node);

        LUT_nodesKeys[i] = OverlayKey::sha1(binValue);
    }
    std::cout << "node" << nodeId << "␣key␣=>␣" << LUT_nodesKeys[nodeId] << '\
        n';

    //initRpcs();
    WATCH(numSent);
    WATCH(numGetSent);
    WATCH(numGetError);
    WATCH(numGetSuccess);
    WATCH(numPutSent);
    WATCH(numPutError);
    WATCH(numPutSuccess);

    WATCH(numSimNodes);
    WATCH(numActiveLookupReceived);
    WATCH(numLookupSent);
    WATCH(numLookupResponseReceived);
    WATCH(numJoin);
    WATCH(numIterations);
    WATCH(groupId);

    nodeIsLeavingSoon = false;
    closestPackage.groupId = -1;
    lastGroupsIds.push_back(groupId);

    msLookup_timer = new cMessage("ms_lookup_timer");
    joinMessage = new cMessage("join_packet");
    resetTimer = new cMessage("reset_timer");

    scheduleAt(simTime() + simLookupTime, msLookup_timer);
    scheduleAt(uniform(simTime() + 0.1, simTime() + 0.5), joinMessage);
    scheduleAt(simTime() + 20, resetTimer);
}

void MSTestApp::handleRpcResponse(BaseResponseMessage* msg,
        const RpcState& state, simtime_t rtt) {
    RPC_SWITCH_START(msg)
        RPC_ON_RESPONSE( DHTputCAPI) {
            handlePutResponse(_DHTputCAPIResponse,
                    check_and_cast<DHTStatsContext*>(state.getContext()));
            EV << "[MSTestApp::handleRpcResponse()]\n"
                       << "␣␣␣␣DHT␣Put␣RPC␣Response␣received:␣id="
                       << state.getId() << "␣msg=" << *_DHTputCAPIResponse
                       << "␣rtt=" << rtt << endl;
            break;
        }
        RPC_ON_RESPONSE(DHTgetCAPI)
        {
            handleGetResponse(_DHTgetCAPIResponse,
                    check_and_cast<DHTStatsContext*>(state.getContext()));
            EV << "[MSTestApp::handleRpcResponse()]\n"
                       << "␣␣␣␣DHT␣Get␣RPC␣Response␣received:␣id="
                       << state.getId() << "␣msg=" << *_DHTgetCAPIResponse
                       << "␣rtt=" << rtt << endl;
            break;
        }RPC_SWITCH_END()
}

void MSTestApp::handlePutResponse(DHTputCAPIResponse* msg,
        DHTStatsContext* context) {
    DHTEntry entry = { context->value, simTime() + ttl, simTime() };

    globalDhtTestMap->insertEntry(context->key, entry);

    if (context->measurementPhase == false) {
        // don't count response, if the request was not sent
        // in the measurement phase
        delete context;
        return;
    }

    if (msg->getIsSuccess()) {
        RECORD_STATS(numPutSuccess++);
        RECORD_STATS(
                globalStatistics->addStdDev("MSTestApp:␣PUT␣Latency␣(s)",
                    SIMTIME_DBL(simTime() - context->requestTime)));
    } else {
        RECORD_STATS(numPutError++);
    }

    delete context;
}

LookupPacket MSTestApp::extractLookupPckInfo(const DHTEntry* entry) {
    LookupPacket pck;

    if(entry->value == BinaryValue::UNSPECIFIED_VALUE) {
        pck.centroidX = -1;
        pck.centroidY = -1;
        pck.groupId = -1;
        pck.range = -1;

        return pck;
    }

    std::ostringstream os;
    os << entry->value;
    std::string value = os.str();
    vector<string> fields;
    fields = split(value, ":", TOKENIZER_NO_EMPTIES);

    pck.centroidX = to_double(fields[0]);
    pck.centroidY = to_double(fields[1]);
    pck.groupId = to_int(fields[2]);
    pck.range = to_double(fields[3]);

    return pck;
}

void MSTestApp::joinGroup(LookupPacket packet) {
    if (packet.groupId == this->groupId) {
        return;
    }

    if (packet.groupId == -1) {
        return;
    }

    numJoin++;

    std::cout << "Node␣" << nodeId
            << "␣>>␣Received␣join␣proposal␣(cur␣groupId:␣" << groupId << "␣
                last:␣" << lastGroupsIds.back() << ")␣"
            << "␣posX:␣" << packet.centroidX << "␣posY:␣" << packet.centroidY
            << "␣group:␣" << packet.groupId << "\n";

    if(packet.groupId != lastGroupsIds.back()) {
        numIterations++;
    }

    centroidX = (posX + packet.centroidX) / 2;
    centroidY = (posY + packet.centroidY) / 2;
    range = (range + packet.range) / 2;
    groupId = groupId*packet.groupId;

    lastGroupsIds.push_back(groupId);

    std::cout << "Node" << nodeId << "␣now␣is␣in␣group␣" << groupId << "\n";
}
```

```

double MSTestApp::distance(double x1, double y1, double x2, double y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

bool MSTestApp::isInRectangle(double centerX, double centerY, double radius,
        double x, double y) {
    return x >= centerX - radius && x <= centerX + radius
            && y >= centerY - radius && y <= centerY + radius;
}

bool MSTestApp::isPointInCircle(double centerX, double centerY, double radius,
        double x, double y) {
    if (isInRectangle(centerX, centerY, radius, x, y)) {
        double dx = centerX - x;
        double dy = centerY - y;
        dx *= dx;
        dy *= dy;
        double distanceSquared = dx + dy;
        double radiusSquared = radius * radius;
        return distanceSquared <= radiusSquared;
    }
    return false;
}

void MSTestApp::handleGetResponse(DHTgetCAPIResponse* msg,
        DHTStatsContext* context) {
    if (context->measurementPhase == false) {
        // don't count response, if the request was not sent
        // in the measurement phase
        delete context;
        return;
    }

    RECORD_STATS(
            globalStatistics->addStdDev("MSTestApp:␣GET␣Latency␣(s)",
                SIMTIME_DBL(simTime() - context->requestTime)));

    if (!(msg->getIsSuccess())) {
        RECORD_STATS(numGetError++);
        delete context;
        return;
    }

    DHTEntry* entry = const_cast<DHTEntry*>(globalDhtTestMap->findEntry(
        context->key));

    if (entry == NULL) {
        //unexpected key
        RECORD_STATS(numGetError++);
        delete context;
        return;
    }

    if (simTime() > entry->endtime) {
        //this key doesn't exist anymore in the DHT, delete it in our
            hashtable
        globalDhtTestMap->eraseEntry(context->key);
        delete context;

        if (msg->getResultArraySize() > 0) {
            RECORD_STATS(numGetError++);
            return;
        } else {
            RECORD_STATS(numGetSuccess++);
            return;
        }
    } else {
        delete context;
        if ((msg->getResultArraySize() > 0)
                && (msg->getResult(0).getValue() == entry->value)) {
            RECORD_STATS(numGetSuccess++);
            return;
        } else {
            RECORD_STATS(numGetError++);
            return;
        }
    }

}

void MSTestApp::handleTraceMessage(cMessage* msg) {
    char* cmd = new char[strlen(msg->getName()) + 1];
    strcpy(cmd, msg->getName());

    if (strlen(msg->getName()) < 5) {
        delete[] cmd;
        delete msg;
        return;
    }

    if (strncmp(cmd, "PUT␣", 4) == 0) {
        // Generate key
        char* buf = cmd + 4;

        while (!isspace(buf[0])) {
            if (buf[0] == '\0')
                throw cRuntimeError("Error␣parsing␣PUT␣command");
            buf++;
        }

        buf[0] = '\0';
        BinaryValue b(cmd + 4);
        OverlayKey destKey(OverlayKey::sha1(b));

        // get value
        buf++;

        // build putMsg
        DHTputCAPICall* dhtPutMsg = new DHTputCAPICall();
        dhtPutMsg->setKey(destKey);
        dhtPutMsg->setValue(buf);
        dhtPutMsg->setTtl(ttl);
        dhtPutMsg->setIsModifiable(true);
        RECORD_STATS(numSent++; numPutSent++);
        sendInternalRpcCall(TIER1_COMP, dhtPutMsg,
                new DHTStatsContext(globalStatistics->isMeasuring(), simTime()
                    ,
                        destKey, buf));
    } else if (strncmp(cmd, "GET␣", 4) == 0) {
        // Get key
        BinaryValue b(cmd + 4);
        OverlayKey key(OverlayKey::sha1(b));

        DHTgetCAPICall* dhtGetMsg = new DHTgetCAPICall();
        dhtGetMsg->setKey(key);
        RECORD_STATS(numSent++; numGetSent++);
        sendInternalRpcCall(TIER1_COMP, dhtGetMsg,
                new DHTStatsContext(globalStatistics->isMeasuring(), simTime(),
                        key));
    } else {
        throw cRuntimeError("Unknown␣trace␣command;␣"
                "only␣GET␣and␣PUT␣are␣allowed");
    }

    delete[] cmd;
    delete msg;
}

void MSTestApp::handleTimerEvent(cMessage* msg) {
    if (nodeId > numSimNodes) {
        return;
    }

    if (msg->isName("reset_timer")) {
        centroidX = posX;
        centroidY = posY;
        range = initialRange;

        scheduleAt(simTime() + 20, msg);
    }

    if (msg->isName("join_packet")) {
        try {
//            if(closestNodeEntry != NULL)

            joinGroup(closestPackage);

            std::ostringstream os;
            os << centroidX << ":" << centroidY << ":" << groupId << ":" << range;

            DHTputCAPICall* dhtPutMsg = new DHTputCAPICall();
            dhtPutMsg->setKey(LUT_nodesKeys[nodeId]);
            dhtPutMsg->setValue(os.str());
            dhtPutMsg->setTtl(ttl);
            dhtPutMsg->setIsModifiable(true);
            RECORD_STATS(numSent++; numPutSent++);
            sendInternalRpcCall(TIER1_COMP, dhtPutMsg,
                    new DHTStatsContext(globalStatistics->isMeasuring(), simTime(),
                            LUT_nodesKeys[nodeId], dhtPutMsg->getValue()));

            return;
        } catch (...) {
            std::cout << "\n␣Insertion␣failed.";
            return;
        }
    }

    if (msg->isName("ms_lookup_timer")) {
        scheduleAt(simTime() + simLookupTime, msg);

        // Non fa nulla se sono ancora in fase di inizializzazione dell'
            overlay
        if (((!activeNetwInitPhase) && (underlayConfigurator->isInInitPhase())
            )
                || underlayConfigurator->isSimulationEndingSoon()
                || nodeIsLeavingSoon) {
            return;
        }

        double closestNodeDistance = 10000000;
        LookupPacket closestEntryPacket;
        closestPackage.groupId = -1;
        int count = 0;

        for (unsigned int i = 0; i < globalNodeList->getNumNodes(); ++i) {
            //myself
            if (i == nodeId)
                continue;

            const OverlayKey& key = LUT_nodesKeys[i];

            if (key.isUnspecified()) {
                EV << "[MSTestApp::handleTimerEvent() @ " << thisNode.getIp()
                          << " (" << thisNode.getKey().toString(16) << ")]\n"
                          << "    Error: No key available in global DHT test 
                              map!"
                          << endl;
                return;
            }

            numLookupSent++;

            DHTEntry* entry = const_cast<DHTEntry*>(globalDhtTestMap->
                findEntry(key));

            if (entry == NULL)
                continue;

            if (entry->value == BinaryValue::UNSPECIFIED_VALUE) {
                continue;
            }

            LookupPacket packet = extractLookupPckInfo(entry);
            double dblDistance = distance(posX, posY, packet.centroidX,
                    packet.centroidY);

            if (isPointInCircle(posX, posY, range, packet.centroidX,
                    packet.centroidY) && dblDistance < closestNodeDistance) {
                numLookupResponseReceived++;
                closestEntryPacket = packet;
                closestNodeDistance = dblDistance;
                count++;
            }
        }

        if(count > 0) {
            closestPackage = closestEntryPacket;
            std::cout << "Closest␣node␣entry␣found:␣" << closestPackage.groupId << "\n";
            scheduleAt(uniform(simTime() + 0.5, simTime() + 1.0), joinMessage);
        }
    }
}

void MSTestApp::handleNodeLeaveNotification() {
    nodeIsLeavingSoon = true;
}

void MSTestApp::finishApp() {
    simtime_t time = globalStatistics->calcMeasuredLifetime(creationTime);

    if (time >= GlobalStatistics::MIN_MEASURED) {
        // record scalar data
        globalStatistics->addStdDev("MSTestApp:␣Sent␣Total␣Messages/s",
                numSent / time);
        globalStatistics->addStdDev("MSTestApp:␣Sent␣GET␣Messages/s",
                numGetSent / time);
        globalStatistics->addStdDev("MSTestApp:␣Failed␣GET␣Requests/s",
                numGetError / time);
        globalStatistics->addStdDev("MSTestApp:␣Successful␣GET␣Requests/s",
                numGetSuccess / time);

        globalStatistics->addStdDev("MSTestApp:␣Sent␣PUT␣Messages/s", numPutSent / time);
        globalStatistics->addStdDev("MSTestApp:␣Failed␣PUT␣Requests/s", numPutError / time);
        globalStatistics->addStdDev("MSTestApp:␣Successful␣PUT␣Requests/s", numPutSuccess / time);

        recordScalar("MSTestApp:␣Number␣of␣lookups", numLookupSent);
        recordScalar("MSTestApp:␣Number␣of␣joins", numJoin);
        recordScalar("MSTestApp:␣Number␣of␣iterations", numIterations);
        recordScalar("MSTestApp:␣Final␣group␣Id", groupId);

        if ((numGetSuccess + numGetError) > 0) {
            globalStatistics->addStdDev("MSTestApp:␣GET␣Success␣Ratio",
                    (double) numGetSuccess
                            / (double) (numGetSuccess + numGetError));
        }
    }
}
```

Listing 3: MSTestApp.hh

```
//
// Copyright (C) 2014 Facoltà di Ingegneria Informatica, Alma Mater Studiorum
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
//

/**
 * @file MSTestApp.cc
 * @author Denis Billi
 */

#ifndef __DHTTESTAPP_H_
#define __DHTTESTAPP_H_

#include <omnetpp.h>

#include <GlobalNodeList.h>
#include <GlobalStatistics.h>
#include <UnderlayConfigurator.h>
#include <TransportAddress.h>
#include <OverlayKey.h>
#include <InitStages.h>
#include <BinaryValue.h>
#include <BaseApp.h>
#include <set>
#include <sstream>

class GlobalDhtTestMap;

struct LookupPacket {
    LookupPacket() : centroidX(-1), centroidY(-1), groupId(-1), range(-1) {}
    double centroidX;
    double centroidY;
    int groupId;
    double range;
    simtime_t respTime;
};

/**
 * A simple test application for the DHT layer
 *
 * A simple test application that does random put and get calls
 * on the DHT layer
 *
 * @author Denis Billi
 */
class MSTestApp : public BaseApp
{
private:
    /**
     * A container used by the MSTestApp to
     * store context information for statistics
     *
     * @author Denis Billi
     */
    class DHTStatsContext : public cPolymorphic
    {
    public:
        bool measurementPhase;
        simtime_t requestTime;
        OverlayKey key;
        BinaryValue value;

        DHTStatsContext(bool measurementPhase,
                        simtime_t requestTime,
                        const OverlayKey& key,
                        const BinaryValue& value = BinaryValue::
                            UNSPECIFIED_VALUE) :
            measurementPhase(measurementPhase), requestTime(requestTime),
            key(key), value(value) {};
    };

    void initializeApp(int stage);

    /**
     * Get a random key of the hashmap
     */
    OverlayKey getRandomKey();

    /**
     * generate a random human readable binary value
     */
    BinaryValue generateRandomValue();

    void finishApp();

    /**
     * processes get responses
     *
     * method to handle get responses
     * should be overwritten in derived application if needed
     * @param msg get response message
     * @param context context object used for collecting statistics
     */
    virtual void handleGetResponse(DHTgetCAPIResponse* msg,
                                   DHTStatsContext* context);

    /**
     * processes put responses
     *
     * method to handle put responses
     * should be overwritten in derived application if needed
     * @param msg put response message
     * @param context context object used for collecting statistics
     */
    virtual void handlePutResponse(DHTputCAPIResponse* msg,
                                   DHTStatsContext* context);

    /**
     * processes self-messages
     *
     * method to handle self-messages
     * should be overwritten in derived application if needed
     * @param msg self-message
     */
    virtual void handleTimerEvent(cMessage* msg);

    /**
     * handleTraceMessage gets called of handleMessage(cMessage* msg)
     * if a message arrives at trace_in. The command included in this
     * message should be parsed and handled.
     *
     * @param msg the command message to handle
     */
    virtual void handleTraceMessage(cMessage* msg);

    virtual void handleNodeLeaveNotification();

    // see RpcListener.h
    void handleRpcResponse(BaseResponseMessage* msg, const RpcState& state,
                           simtime_t rtt);
    void joinGroup(LookupPacket entry);

    LookupPacket extractLookupPckInfo(const DHTEntry* entry);

    UnderlayConfigurator* underlayConfigurator; /**< pointer to
        UnderlayConfigurator in this node */

    GlobalNodeList* globalNodeList; /**< pointer to GlobalNodeList in this
        node*/

    GlobalStatistics* globalStatistics; /**< pointer to GlobalStatistics
        module in this node*/
    GlobalDhtTestMap* globalDhtTestMap; /**< pointer to the GlobalDhtTestMap
        module */

    // parameters
    bool debugOutput; /**< debug output yes/no?*/
    double mean; //!< mean time interval between sending test messages
    double deviation; //!< deviation of time interval
    int ttl; /**< ttl for stored DHT records */
    bool p2pnsTraffic; //!< model p2pns application traffic */
    bool activeNetwInitPhase; //!< is app active in network init phase?

    // statistics
    int numSent; /**< number of sent packets*/
    int numGetSent; /**< number of get sent*/
    int numGetError; /**< number of false get responses*/
    int numGetSuccess; /**< number of false get responses*/
    int numPutSent; /**< number of put sent*/
    int numPutError; /**< number of error in put responses*/
    int numPutSuccess; /**< number of success in put responses*/

    // MediaSense statistics
    int simLookupTime;
    int numSimNodes;
    int numActiveLookupReceived;
    int numLookupSent;
    int numLookupResponseReceived;
    int numJoin;
    int numIterations;
    int nodeId;
    int groupId;
    int initialGroupId;
    int simResetTime;
    std::vector<int> lastGroupsIds;

    int posX;
    int posY;
    double centroidX;
    double centroidY;
    int range;
    int initialRange;
    simtime_t lastJoin;

    LookupPacket closestPackage;

    std::vector<OverlayKey> LUT_nodesKeys;

    cMessage *msLookup_timer, *joinMessage, *resetTimer;
    bool nodeIsLeavingSoon; //!< true if the node is going to be killed
        shortly

    static const int DHTTESTAPP_VALUE_LEN = 20;

public:
    MSTestApp();

    double distance(double x1, double y1, double x2, double y2);
    bool isInRectangle(double centerX, double centerY, double radius, double x
        , double y);
    bool isPointInCircle(double centerX, double centerY, double radius, double
         x, double y);

    /**
     * virtual destructor
     */
    virtual ~MSTestApp();

};

#endif
```

BIBLIOGRAPHY

[1] Karl Aberer et al. “Advanced Peer-to-Peer Networking : The P-Grid System and its Applications *”. In: *Information Systems* 5005 (), pp. 1–6.

[2] F. Al-Shraideh. “Host Identity Protocol”. In: *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL’06)* (2006), pp. 203–203. DOI: 10.1109/ICNICONSMCL.2006.112. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1628448.

[3] *Android SCTP Support*. 2011. URL: http://code.google.com/p/android/issues/detail?id=3272&q=sctp&colspec=ID%20Type%20Status%20Owner%20Summary%20Stars.

[4] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. *OverSim: A Flexible Overlay Network Simulation Framework*. May 2007.

[5] Stephen Boyd. “Gossip Algorithms: Design, Analysis and Applications”. In: 00.C (2005), pp. 1653–1664.

[6] Guanling Chen and David Kotz. “A Survey of Context-Aware Mobile Computing Research”. In: *Time* (), pp. 1–16.

[7] Suk Kim Chin and Robin Braun. “A Survey of UDP Packet Loss Characteristics”. In: *America* (2007), pp. 200–204.

[8] Sally Floyd, Senior Member, and Kevin Fall. “Promoting the Use of End-to-End Congestion Control in the Internet”. In: 7.4 (1999), pp. 458–472.

[9] Sajal K. Dasc Haitao Lin. “TCP performance analysis of CDMA systems with RLP and MAC layer retransmissions”. In: *Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (2002), pp. 313–

320. DOI: 10.1109/MASCOT.2002.1167091. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1167091.

[10] M. Tim Jones. *Better networking with SCTP*. URL: http://www.ibm.com/developerworks/linux/library/l-sctp/?ca=dgr-lnxw07SCTP.

[11] Ivan Skytte Jörgensen. *Java SCTP Library*. URL: http://i1.dk/JavaSCTP/.

[12] Theo Kanter et al. “Distributed context support for ubiquitous mobile awareness services”. In: *2009 Fourth International Conference on Communications and Networking in China* (Aug. 2009), pp. 1–5. DOI: 10.1109/CHINACOM.2009.5339728. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5339728.

[13] Victor Kardeby and Theo Kanter. “Sensor Sockets Enabling Reliable Communication Using a Context Based Grouping Mechanism”. In: *Context* (2011), pp. 161–167.

[14] *List of SCTP Implementations*. URL: http://www.sctp.org.

[15] Wang Lixin et al. “HMIPv6-based handover optimized solution and performance analysis”. In: *2009 International Conference on Test and Measurement* 3 (Dec. 2009), pp. 133–136. DOI: 10.1109/ICTM.2009.5413092. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5413092.

[16] *LKCSTP Project*. URL: http://lksctp.sourceforge.net/.

[17] Arif Mahmud, Rahim Rahmani, and Theo Kanter. “Deployment of Flow-Sensors in Internet of Things’ Virtualization via OpenFlow”. In: *2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing* (June 2012), pp. 195–200. DOI: 10.1109/MUSIC.2012.41. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6305848.

[18] Ricardo Matos et al. “Context-based wireless mesh networks: a case for network virtualization”. In: *Telecommunication Systems* (Mar. 2011). ISSN: 1018-4864. DOI: 10.1007/s11235-011-9434-3. URL: http://www.springerlink.com/index/10.1007/s11235-011-9434-3.

[19] Wolf R Mende, Femuniversitat Hagen, and P Box. “Evaluation of a proposed Handover algorithm for the GSM cellular system”. In: (1990), pp. 264–269.

[20] Shoma Nakahara, Tomoyuki Ohta, and Yoshiaki Kakuda. “Experimental Evaluation of MANET Based on Autonomous Clustering and P2P Overlay Network”. In: *2013 First International Symposium on Computing and Networking* (Dec. 2013), pp. 480–483. DOI: 10.1109/CANDAR.2013.85. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6726947.

[21] Pekka Nikander, Andrei Gurtov, and Thomas R Henderson. “Host Identity Protocol (HIP): Connectivity , IPv4 and IPv6 Networks”. In: *Communications* 12.2 (2010), pp. 186–204.

[22] phil-news nospam@ipal.net. *Application protocols over SCTP*. 2006. URL: http://www.phwinfo.com/forum/comp-protocols-tcp-ip/134474-application-protocls-over-sctp.html.

[23] K. Omae et al. “Handoff performance of mobile host and mobile router employing HMIP extension”. In: *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.* 2.C (2003), pp. 1218–1223. DOI: 10.1109/WCNC.2003.1200546. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1200546.

[24] Charles E. Perkins. “Mobile Networking through MobileIP”. In: *Ieee Internet Computing* (1998), pp. 58–69.

[25] Steffen Reidt and Stephen D. Wolthusen. “Connectivity augmentation in tactical mobile ad hoc networks”. In: *MILCOM 2008 - 2008 IEEE Military Communications Conference* (Nov. 2008), pp. 1–7. DOI: 10.1109/MILCOM.2008.4753198. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4753198.

[26] Sajal Saha. “THMIP- A Novel Mobility Management Scheme Using Fluid Flow Model”. In: *Update* (2011).

[27] Bill Schilit and New York. “Context-Aware Computing Applications”. In: *System* (), pp. 1–6.

[28] Dong-cheol Shin and Sung-gi Min. “Fast Handover Solution Using Multi-tunnel in HMIPv6 (FM-HMIPv6)”. In: *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)* 6 (2008), pp. 833–838. DOI: 10.1109/SENSORCOMM.

2008.83. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4622763.

[29] K.G. Shin. “Smooth Handoff with Enhanced Packet Buffering-and-Forwarding in Wireless/Mobile Networks”. In: *Second International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'05)* (2005), pp. 39–39. DOI: 10.1109/QSHINE.2005.56. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1551099.

[30] R. Stewart et al. *Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration.* RFC 5061 (Proposed Standard). Internet Engineering Task Force, 2007. URL: http://www.ietf.org/rfc/rfc5061.txt.

[31] I. Stojmenovic. *Handbook of Wireless Networks and Mobile Computing.* Wiley Series on Parallel and Distributed Computing. Wiley, 2003. ISBN: 9780471462989. URL: http://books.google.it/books?id=V5HFbsHgn_wC.

[32] *Stream Control Transmission Protocol (SCTP).* URL: http://www.sctp.de/sctp-download.html.

[33] S. Tarkoma. *Mobile middleware: architecture, patterns and practice.* Wiley, 2009. ISBN: 9780470740736. URL: http://books.google.com/books?id=wsnBA4aD2h0C.

[34] Jamie Walters, Theo Kanter, and Enrico Savioli. “A Distributed Framework for Organizing an Internet of Things”. In: *Context* (), pp. 1–17.

[35] Weihong Wang et al. “Markov-Based Hierarchical User Mobility Model”. In: *2007 Third International Conference on Wireless and Mobile Communications (ICWMC'07)* (Mar. 2007), pp. 47–47. DOI: 10.1109/ICWMC.2007.52. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4138152.

[36] Yi Wang et al. “Cluster based location-aware routing protocol for large scale heterogeneous MANET”. In: *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)* (Aug. 2007), pp. 366–373. DOI: 10.1109/IMSCCS.2007.12. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4392627.

[37] Yi Wang et al. “WACHM: Weight based adaptive clustering for large scale heterogeneous MANET”. In: *2007 International Symposium on Communications and Information Technologies* (Oct. 2007), pp. 936–941. DOI: 10.1109/ISCIT.2007.4392150. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4392150.

[38] Bo Yang, Weimin Wu, and Guangxi Zhu. “Distributed averaging in wireless sensor networks with triplewise gossip algorithms”. In: *IEEE 2013 Tencon - Spring* (Apr. 2013), pp. 178–182. DOI: 10.1109/TENCONSpring.2013.6584436. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6584436.

[39] Guanhua Ye and Tarek N Saadawi. “IPCC-SCTP: An Enhancement to the Standard SCTP to Support Multi-homing Efficiently”. In: *New York* (2004), pp. 523–530.

[40] Stefanos Zachariadis, Cecilia Mascolo, and Wolfgang Emmerich. “SATIN : A Component Model for Mobile Self Organisation”. In: (2004).

[41] Zhun Zhong and Sunghyun Clioi. “IEEE 802.1 1 Link-Layer Forwarding For Smooth Handoff”. In: *Communication* (2003), pp. 1420–1424.