Universitat Politècnica de Catalunya
Escola Tècnica Superior d´Enginyeria de Telecomunicació de Barcelona

ADDRESSING MOBILITY ISSUES IN MOBILE ENVIRONMENT

Master's Thesis completed in fulfilment of the requirements for ERASMUS exchange program in Lappeenranta University of Technology
Department of Information Technology
Communications Software Laboratory

August 15, 2008

Ji Zhang

Supervisor: Professor Jari Porras

Instructor: Arto Hämäläinen

Resumen

Desde el principio de la última década, el uso de los dispositivos móviles creció exponencialmente impulsado por el avance tecnológico y la personalización. Sin embargo, la movilidad y limitación de recursos son dos características innatas de los dispositivos móviles, y dichas características provocan que continuemos tratando las redes fijas y móviles como dos tipos de redes independientes y de difícil interacción mutua.

PeerHood es un diseño de red P2P que considera tanto dispositivos fijos como móviles como parte esencial de un escenario real de distribución de red. sus características principales estan basadas en *Environment Awareness*, interacción entre dispositivos en diferentes protocolos de red, y un diseño P2P destructurado. Como resultado, PeerHood abre un amplio abanico de posibilidades, tal como intercambio de ficheros entre dispositivos fijos y móviles, control remoto, distribución de recursos de la red y *Social Networking*.

No obstante, las características citadas anteriormente de los dispositivos móviles representan un serio obstáculo y desafío para PeerHood y en este proyecto proponemos una solución basado en PeerHood, añadiendo funciones como *Total Environment Awareness* y *Node Interconnectivity*, para lograr una conexión fiable y flexible que se adapta a un entorno móvil cambiante de la maneras más eficiente posible. Entre las tecnologías inalámbricas existentes, Bluetooth fue elegido para la implementación.

La estructura de este proyecto será la siguiente: tras la introducción en el primer capítulo, en el segundo capítulo realizaremos un repaso a la última versión de PeerHood, sus características y principales funcionalidades. En capítulo tres, vamos a analizar en detalle el algoritmo de descubrimiento de dispositivos y sus ventajas. En capítulo cuatro discutiremos el sistema de interconexión de dispositivos remotos. En capítulo cinco analizaremos el escenario problemático y la implementación de los diseños anteriores como solución. Finalmente en el capítulo seis expondremos nuestra conclusión basada en los resultados obtenido a partir de la implementación.

Abstract

Since the beginning of the decade, the use of mobile devices has increased dramatically due to the continuous advances in capability and personalization of the devices. Nevertheless, the dynamic nature and resource limitation of mobile devices make us still consider fix- and mobile networks separately and easy wireless interaction between these two networks is difficult.

PeerHood is an emerging mobile peer to peer network solution which considers both (fix and mobile devices) as essential parts of the real wireless environment. It offers environment awareness, connection between devices under different network technology and an unstructured peer to peer network design. As the result, it opens a potential range of applications and possibilities, such as free interaction between fix and mobile network, remote control, resources distribution or social networking.

However, the characteristics of mobile devices represent a serious obstacle and challenge for PeerHood and make it different than other existing static Peer to Peer network. In this work we propose an approach, based on Total Environment Awareness and Node Interconnectivity, to allow consequently reliable adaptive task migration and connection among mobile devices depending on the environment. Among the existing wireless technologies, Bluetooth has been chosen for the implementation.

KEYWORDS

Mobile P2P, PeerHood, Task Migration, Mobile Handover, seamless connection

Acknowledgements

This master’s thesis has been done for Lappeenranta University of Technology Department of Information Technology in February – August 2008. The thesis has been done as part of a larger project, which concentrates on mobile environment PeerHood.

I want to thank my supervisor for the thesis, Professor Jari Porras, for supporting during my thesis and for giving me valuable insight into PeerHood peer-to-peer networking, and into its potential applications in mobile networks.

I specially would like to thank Arto Hämäläinen, who has been the instructor for my thesis and lead for the PeerHood protocol design. Arto gave many ideas for my thesis and was a great teacher in many occasions.

My gratitude also goes to Teemu Reisbacka and Bishal, who have been working with me in the same department, and giving good ideas for my thesis. I want to thank Teemu for his invaluable help during the writing process. Finally, I would like to thank my family and my girlfriend Laura for supporting me during all my studies.

August 15, 2008

Ji Zhang

Contents

FIGURES LIST

Chapter 1

Introduction

The number of mobile terminals has increased dramatically during the last years. The use of such devices has become more and more important and universal in our lives due to continuous improvement of hardware performance and advances of wireless communication technologies. Since mobile phone was born, we have passed from basic phone functions to TV broadcasting, Global Positioning System (GPS) locationing, social networking, file transferring and any kind of applications that we could only enjoy in the desktop computer in the past. Many mobile devices, such as mobile phones and PDAs, have become essential communication tools in the modern society. Even though they are getting more processing power, battery capacity and other hardware performance advances, mobile devices still are not suitable for carrying out most of high energy consumption applications due to their size and battery limitation.

In the last decade the number of communication networks designed for mobile devices, such as GPRS, Bluetooth, WIFI, 3G, HSDPA, ZIGBEE and IR, have increased enormously. Due to this new coexistence of mobile devices, wireless connection and static computers, many researchers believe that it's possible to distribute the resources of environment in a more efficient way between mobile and static devices and have a better interaction between them. Mobile connectivity solution PeerHood [1, 2] was created to satisfy this resource distribution need. In PeerHood network, mobile devices can take advantage of the nearby computing resources, and migrate their processing tasks to a fixed computation server to execute the task more powerfully and conserve battery energy [8]. Nevertheless, during the process the mobile environment is changing constantly and randomly due to its mobility characteristic. The initial connection has a high probability to be lost. Consequently the performance of the task processing will be seriously limited by the time duration and device mobility which is not desirable at all.

As several researchers [5, 6] have already demonstrated the viability and benefits of this task migration, in this work we assume the benefit of this remote task execution and we will focus on the behaviour of device's connections in a changing mobile environment. Other features as the power consumption saving, transmission cost and time delays are outside the scope of this thesis.

1.1 The Problem: Mobility

One of the main goals of PeerHood is this resource distribution for mobile devices. For example, mobile device has a task that is not suitable for execution in it. This could, for example, be analysis of pictures that requires high processing power. Mobile device looks for the PeerHood environment and selects a suitable device (typically fixed) to migrate the task to. This device starts solving the task while mobile continues its work. After the processing the task is returned back to the device.

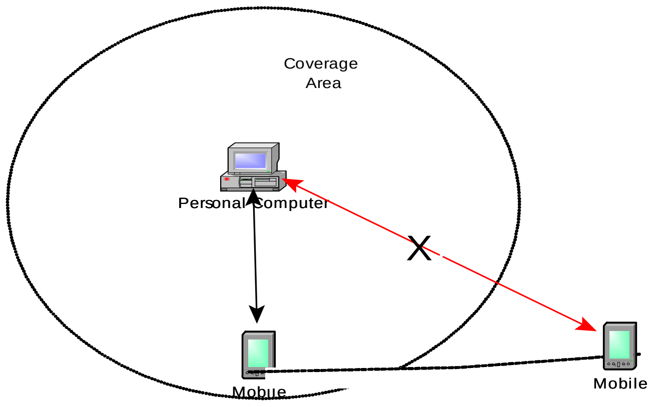


Figure 1.1 Connection loss during Task Migration

However, there exists the possibility to lose the connection due to the device's movement and consequently the coverage loss. Mobility is the essence of mobile devices and the established connection could be lost in any moment due to the unavoidable coverage limitation. Whenever the connection is lost, any migrated task is forced to be finished and it will affect seriously the remote execution performance of PeerHood. In figure 1.1 we have the simplest scenario of connection loss due to the device's mobility.

Obviously there is no easy solution for the previous scenario. Nevertheless, in the real wireless communication environment, like in figure 1.2, during the mobile device's movement new elements as other mobile devices, task servers of other services and more network elements also appear inside this changing environment. Although these new elements are not directly useful to migrate the task, we believe these elements might provide to us with the interconnection capability between the mobile device and the first task executor. In other words, these elements could be used to construct a continuous adaptive connection through different network nodes to solve the mobility problem during Task Migration.

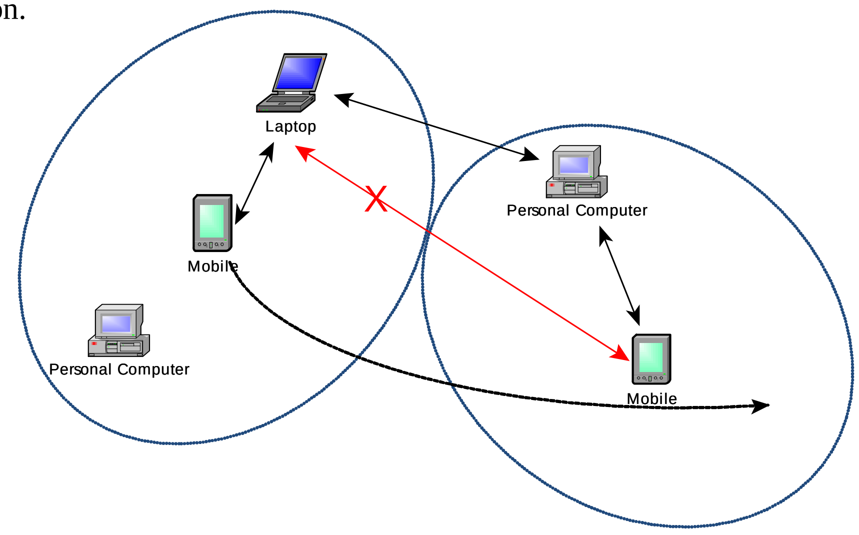


Figure 1.2 Real task migration scenario

1.2 Objectives and Scope

Based on the idea of last subchapter; the objective of this thesis is to find out how to efficiently adapt task migration for mobile devices within a continuous changing network environment. To solve the problem of avoiding the connection loss and completing the task, we realized it is essential to have a total environment awareness to discover not only the direct neighborhood but all the devices inside the total coverage area, a connection selection system to arrive to each one and an automatic interconnection system that allows connections between remote devices through jumps. Thus the devices can choose freely different connection configuration according to the network environment and the application need. All the mentioned changes were made based on the previous version of PeerHood and the results will be analyzed to draw conclusions about task migration in a mobile peer-to-peer environment.

1.3 Structure

The structure of the thesis will be the following: in the second chapter we will take a review of PeerHood, its characteristics and its main functionalities. In chapter three, we will analyze in detail the new device and service discovery and its advantages. In chapter four we will discuss the interconnection system of remote devices. The chapter five will discuss the scenario of task migration and the seamless handover implementation as the solution. Finally, in chapter six, we will analyze the result of the implementation and draw conclusions on the work done.

Chapter 2

PeerHood Environment

2.1 Overview of PeerHood

PeerHood [1, 2] was created with the goal to offer an unstructured peer to peer neighborhood communication in a mobile environment. Mobile and static PeerHood devices with are aware of the nearby device's existence and are able to communicate directly with each other without any centralized servers. In order to achieve this type of mobile ad-hoc network, the information of each device's immediate neighbours are monitored and updated continuously through device discovery inquiries and then stored in the system for future usage. Such design of discovery process provides devices the environment awareness, making possible the following wireless peer-to-peer connection establishment. The basic scenario of PeerHood is presented in Figure 2.1, where every device is aware of the environment and ready to connect and be connected.

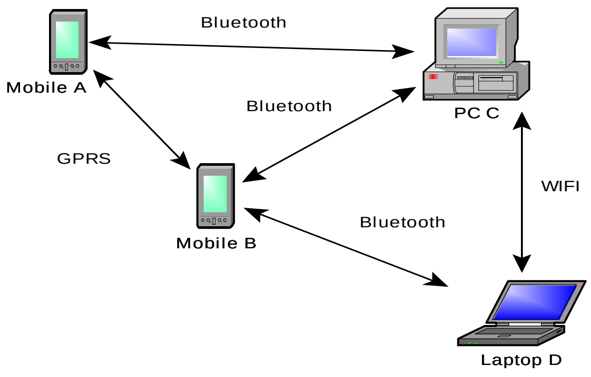


Figure 2.1 PeerHood basic scenario

Another important feature of PeerHood is the creation of a common (abstract) interface to unify different network technologies so the underlying network structure is invisible from the application layer. Complex tasks like device discovery, service discovery, connection establishment and error checking are handled by the PeerHood system and the application only has to consider functionalities of the highest layer. As the consequence, the application development difficulty will be reduced considerably. Following the idea of remote execution and optimal network resource distribution, PeerHood has also been designed to offer devices the capability to share services and applications with other devices within the same PeerHood environment. Currently PeerHood works with Bluetooth, Wireless LAN (WLAN) and General Packet Radio Service (GPRS).

2.2 PeerHood Implementation

We consider that the PeerHood consists of two main independent parts: daemon and library. Daemon is the process in charge of searching permanently for remote devices and their services through different network plugins as well as act as the storage of the information. On the other hand, Daemon is in charge of the configuration parameters from the system, and sends stored information as response to other PeerHood device inquiries. The library interface, which is connected to the Daemon using local sockets, is in charge of taking information from the Daemon and offering PeerHood functionality to the applications layer. The structure of PeerHood implementation is presented in Figure 2.2.

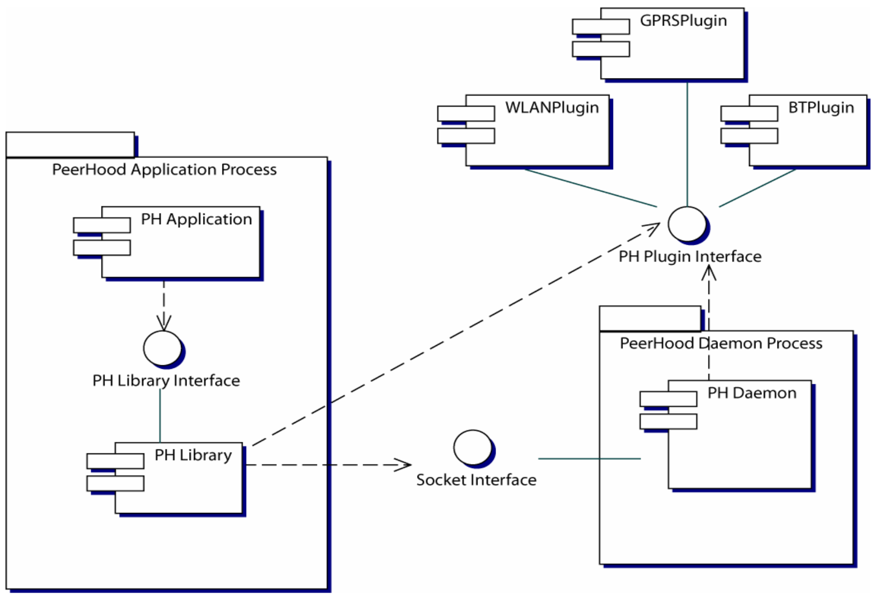


Figure 2.2 PeerHood Implementation

It’s important also to notice the usage of abstraction in PeerHood structure. Due to the fact that PeerHood has been designed for more than one network technology and the possibility to add new classes in the future has been left in it, the backbone structure of PeerHood is made by abstract devices, abstract plugins and abstract connections. Singleton pattern design was used for Daemon and Library class.

2.2.1 Daemon

Daemon is the main class of PeerHood which consists of a group of network plugins in charge of information exchanging with other devices, a device storage where all the remote devices information of the environment and a local socket connection system to listen to the application/library petitions are stored. During the daemon initialization, after the plugins creation, 2 threads are also created by each plugin. Inquiry thread is the one in charge to search for other devices, create the appropriate connection to them and fetch the necessary information in the neighbourhood. On the other hand, listening to advertise is who adverts about own device and sends daemon and device storage’s information to other devices. As we commented before, PeerHood application layer doesn’t have any direct contact with daemon. PeerHood library should be used to access the daemon, and the local sockets are used to send all required information. Device storage is the class where all the remote devices information is stored. The daemon class structure is presented in figure 2.3.

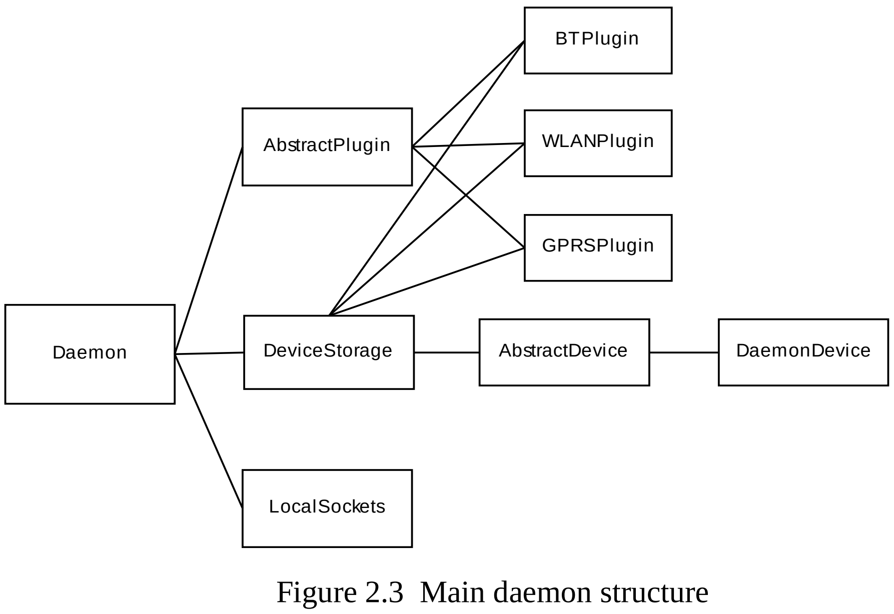


Figure 2.3 Main daemon structure

2.2.2 Library

Library is the main class and we can summarize it in 4 fields: connection establishment, requesting neighbourhood information from the daemon, connection quality monitoring and incoming connection listening. Although the daemon also establishes a short duration connection to other devices to exchange information, the real connection creation for data transmission between devices is managed by PeerHood library. Applications can easily use the Connect( ) function of the library to establishment the data transmission with neighbour devices. Functions like GetDeviceList( ), GetServiceList( ) and RegisterService( ) interact with daemon through local sockets in order to get neighbourhood information or indicate new service to the daemon.

Engine is an element of the library that listens to connection request from other devices and the acceptance will be sent back to applications through callbacks. To offer a seamless connectivity, PeerHood library also includes connection monitoring, which listens to the connection quality permanently to detect the possible connection losses and reacts to them accordingly.

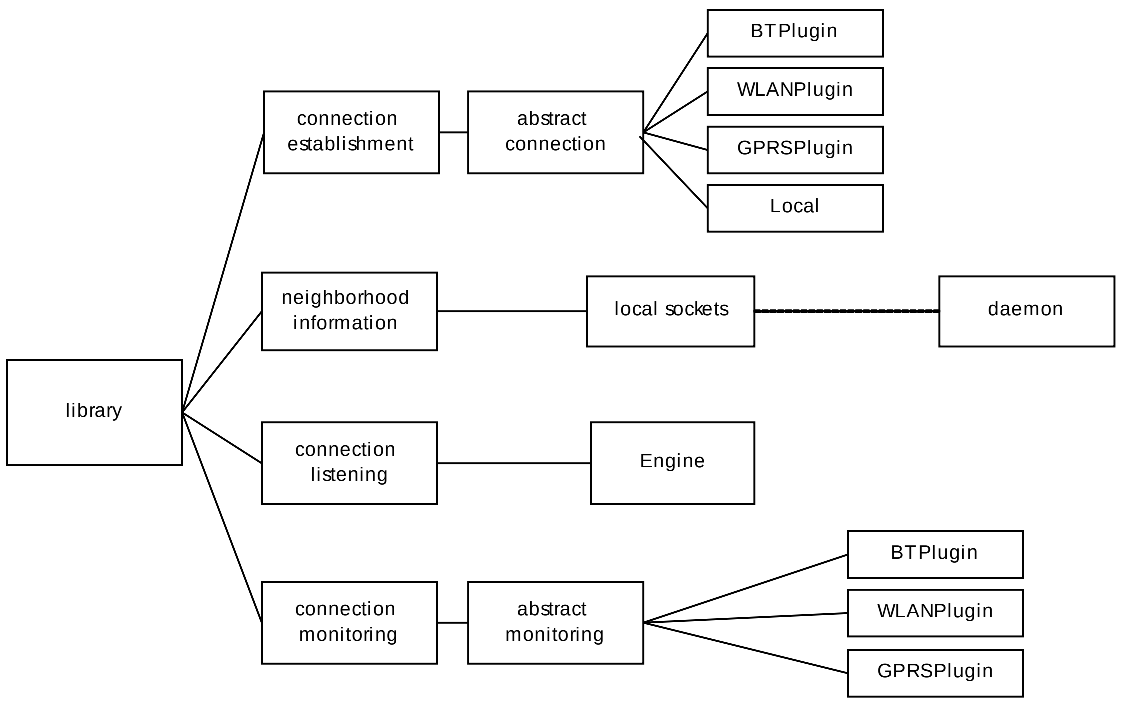


Figure 2.4 Main library structure

2.3 Functionalities of PeerHood

The following key functionalities are included in the previous PeerHood implementation:

- Device Discovery: Device Discovery is the process which provides information about all devices inside the original one’s coverage. According to number of allowed networks technologies, the same number of plugins executes discoveries processes to detect corresponding technology devices. To be able to distinguish devices from each other, the devices must contain some unique information. MAC-Address of network interfaces is the most appropriate due to the singularity of each interface, even inside the same device. Checksum number is also included as device parameter. Currently checksum is the same as daemon process ID number and is not used. Once a device is detected, its PeerHood availability will be checked by device discovery inquiry. In the case of Bluetooth, the SDP query is used. For each found device The SDP query will try to find the PeerHood tag. If this tag is found, the device will be marked as PeerHood capable. For each PeerHood capable device the neighbour devices list is extracted from the device storage and sent to the discovery inquiry as neighbourhood information.

- Service Discovery: According to the principle of resource distribution, any PeerHood registered service will be discoverable by the other device’s inquiries. These services could be also accessed by any PeerHood device in the environment by means of wireless connections. During the device discovery process, for each PeerHood available device the services information will also be sent to discovery inquiry the same way as the neighbourhood list. PeerHood service is described by the following parameters: ServiceName, ServiceAttribute and Port Number.

- Connection establishment: PeerHood offers connection and transmission between 2 devices in the same neighbourhood. Method Connect is used to establish connection in the application level. In figure 2.5 the basic connection diagram is explained.

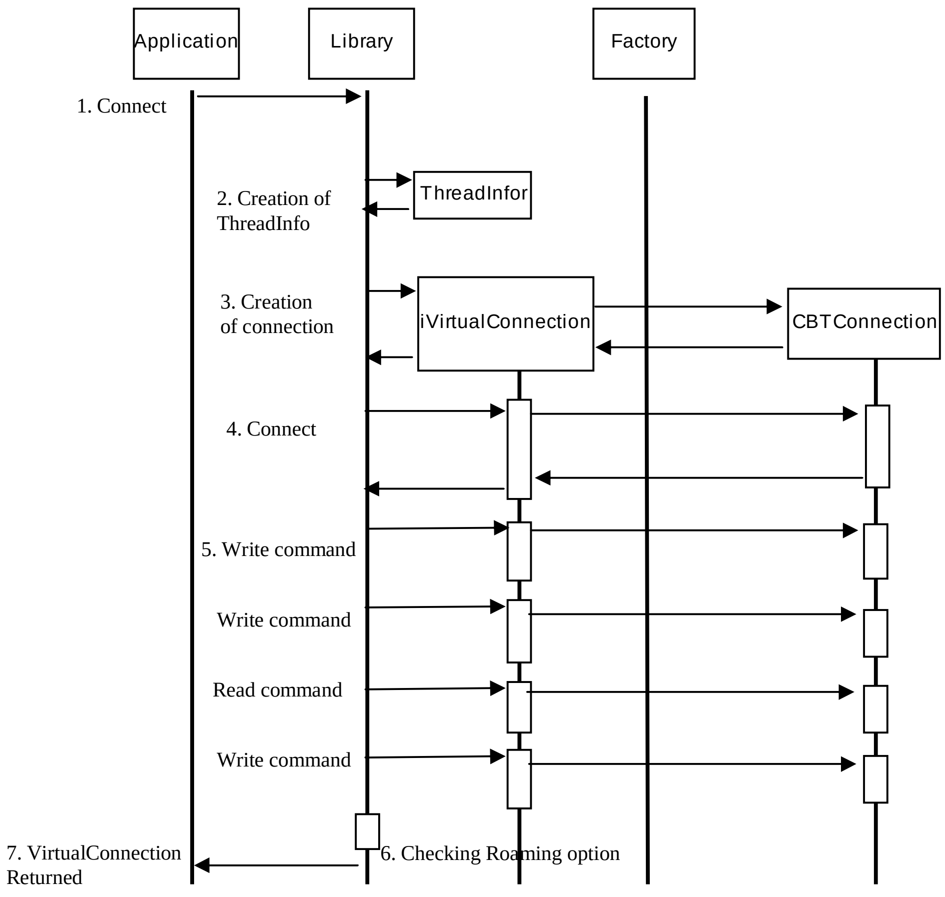


Figure 2.5 PeerHood Connect

1. Connect: Application calls PeerHood interface method ‘connect’.
2. Creation of ThreadInfor: Connection information is stored here and put in iThreadlist for the further use.
3. Creation of connection: VirtualConnection uses factory to create a new connection according to the network prototype.
4. Connect: PeerHood Library calls VirtualConnection’s method ‘connect’, creation of Bluetooth sockets.
5. Write & Read commands: Exchanging commands and information with the remote device.
6. Checking Roaming configuration and creation of Roaming thread.
7. Connection Returned: Created Connection is returned to application.

- Data Transmission: After the connection establishment, PeerHood supports data transmissions between connected devices through MabstractConnection interface. Methods Write and Read are used to send and receive information directly in the application layer.

- Seamless Connectivity: When link quality gets weak or breaks, PeerHood will try to keep the transmission of data by establishing a new alternative wireless technology connection. While the connection is established, a roaming thread is continuously searching for a second way to connect to the same service in the same device. Once the alternative is found, handover can be done instantly, thus restabilising the old connection. Connection ID is used to identify the connection to substitute from the connection list.

Previous functions are already implemented successfully in PeerHood. In the following chapters we will proceed to discuss our improvement to provide the connection adaptation capacity in mobile environment. Respectively they are Dynamic Device Discovery and Interconnection.

Chapter 3

Dynamic Device Discovery

3.1 Coverage Exclusion

Due to the non-central server and the total random distribution nature of mobile devices, one device can only fetch information from devices inside its own coverage. It means that the size of the network is drastically limited by the device coverage. If we consider that in the beginning the PeerHood protocol was assigned to work in a close environment and interact only with direct neighbour devices inside the inquiry coverage, the coverage exclusion problem is still present. For instance, in figure 3.1 the mobile device 2 can send inquiry to the laptop and other mobile device inside its coverage area and achieve the total network knowledge.

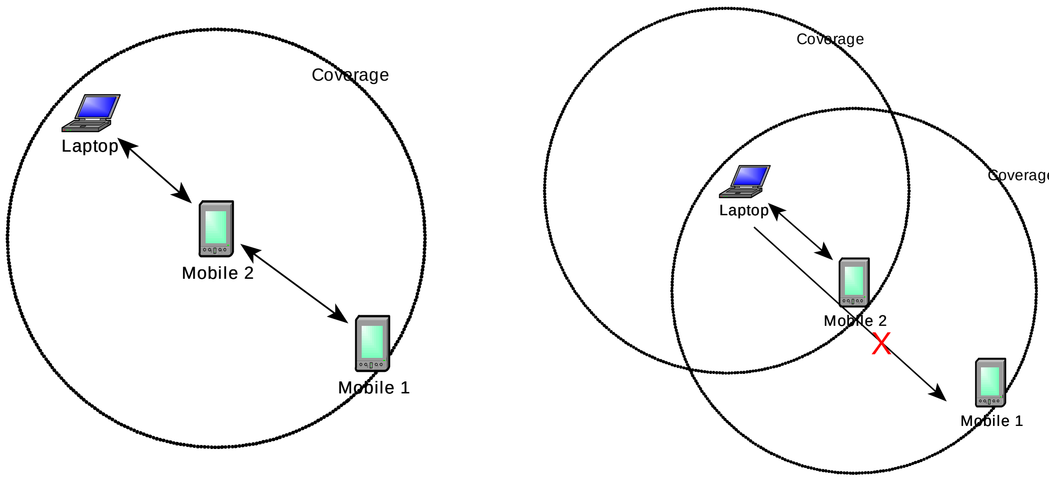


Figure 3.1 Coverage Exclusion

However, if we suppose the laptop has the same coverage area as the mobile device 2, then it can only be aware of the mobile device 2's presence and the mobile device 1 will be invisible to the laptop. To achieve the total network knowledge of all devices, wide enough coverage for each device is required and the distribution of the devices shouldn't be too dispersed. Thus this behaviour will seriously affect the performance of PeerHood network.

In the last version of PeerHood [2] a certain neighbourhood information fetching was included in the device discovery function. The direct neighbourhood information is sent to the inquiry and stored inside each device as list of neighbourhood devices, consequently achieving a better knowledge about a more extense nearby environment. The DeviceStorage structure is presented in figure 3.2.

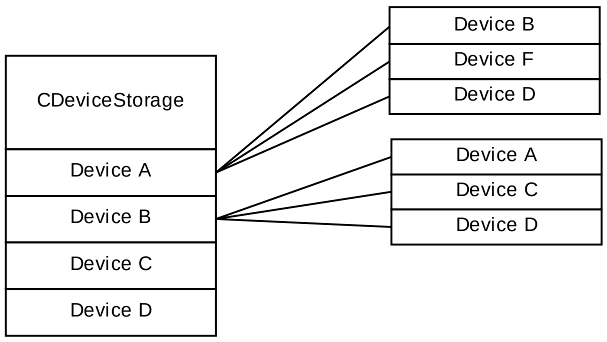


Figure 3.2 Device information storage

Curiously, the goal of mentioned implementation was not to achieve a better awareness about the neighbourhood and solve the problem of coverage exclusion. In fact this implementation was done to achieve faster neighbourhood device information and later check the coverage availability sending a specific verification inquiry. First we consider the possibility to use the implemented topology to solve the coverage exclusion problem. Effectively such implementation improved the PeerHood performance in network acknowledge and size limitation. Nevertheless, the problem of coverage exclusion is not solved yet. In network configuration, such as Figure 3.3, the Device A is aware of the whole network information inquiring to its direct neighbours B,C,D and E. Similarly E is aware of its own direct neighbourhood devices F and G. Process works perfectly for A and E.

However, the situation is not the same for devices B, C and D. If we keep the same network distribution, they will never be notified of the existence of devices F and G and vice verse.
Following the device discovery process logic, any device out of direct neighbour’s coverage won’t be seen by the inquiry process. The neighbourhood information fetching provides only an extra coverage jump vision to the device inquiry process. In other words, the vision of

device discovery process is limited to two jumps and such problem of coverage exclusion will still appear inside the network depending on the devices distribution. If we increase the number of DeviceStorage levels (number of jumps), the visibility problem will be solved. Nevertheless the storage size would increase exponentially and also the transmission data volume, producing a high energy consumption to mobile devices. Due to these reasons we believe other way to resolve the coverage exclusion limitation is needed.

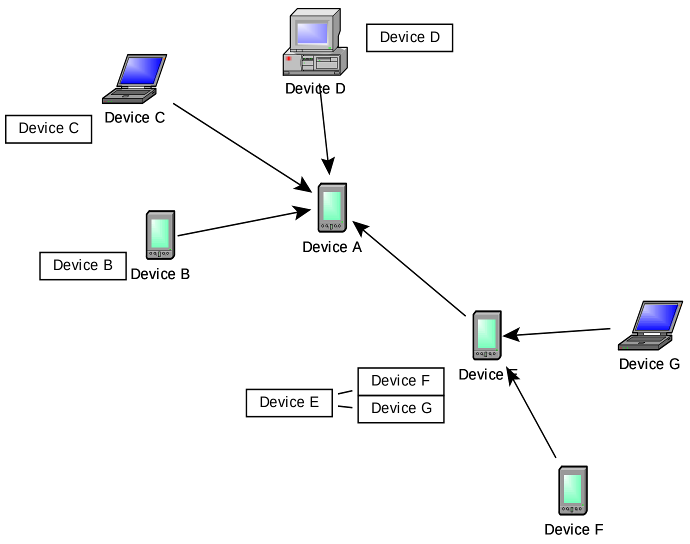


Figure 3.3 Neighbourhood information fetching

3.2 Gnutella P2P network

To solve the problem of coverage exclusion, we propose a searching algorithm inspired by the Gnutella P2P network [17, 19]. Gnutella is one of the most popular unstructured peer to peer file sharing systems. If we consider each user has Gnutella client software as nodes, on initial start up, the client software has to find at least one other node. Different methods have been used for this, including a pre-existing address list of possibly working nodes shipped with the software, using updated web caches of known nodes. Once connected, the client will request a list of working addresses. Whenever the user wants to do a search, the client would send the request to each node it is actively connected to. The number of actively connected nodes for a client was usually quite small (around 5), so each node then forwards the request to all the nodes it is connected to and they in turn forward the request, and so on, until the packet is from a predetermined number of "hops" from the sender. If a search request turns up a result, the node that had the result will contact the searcher, sending the information back along the same route the query came through.

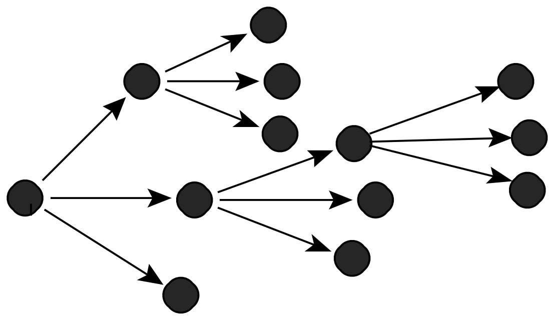


Figure 3.4 Gnutella network structure

One of the biggest performance problems is the huge network traffic generated due to the high number of query messages [9]. The importance of this is less for fixed internet network that doesn't have to take into account the network bandwidth and energy consumption. However, these two factors are critical for the PeerHood protocol due to its focus on mobile devices. and evidently the same inquiry process of Gnutella won't work appropriately in PeerHood. On the other hand, this sort of device discovery process by node jumps would provide the whole network information to any device in the network and it would make PeerHood a definitely scalable network.

3.3 Dynamic Device Discovery

Based on the same principle of Gnutella's network distribution and other researcher's results[10, 11, 12, 13], we have created the new device discovery that considers each PeerHood device as an independent node. Each node can search the nearest devices information in a certain coverage area and later these devices are stored in the neighbourhood list. As presented in figure 3.5, whenever a device receives the discovery inquiry, all its neighbourhood information will be sent to the inquiry owner. The inquiry owner device will process the received neighbourhood list and store them as other direct devices inside the coverage in its neighbourhood list, adding the routing information as bridge name and jump number. The same process will continue with the next node. The final result is a device list with information about the whole network with its routing information.

Thus every device of the network will achieve the total environment awareness and the way to connect to each other. The resource consumption will be the same, because the inquiry petition is not repeated like Gnutella network, but only sent to the direct neighbours. Compared to the previous version of PeerHood, the use of Bridge address and Jump number are the most relevant elements that transform the DeviceStorage into an Ad-hoc routing address table[14, 15]. Several similar studies were also carried by other researchers to demonstrate the viability of the mobile environment awareness.

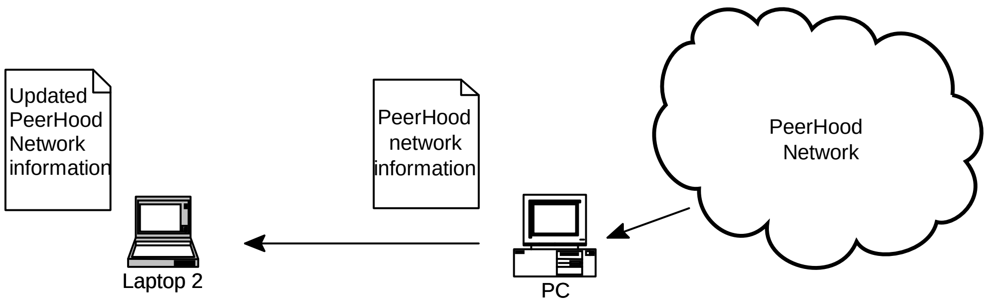
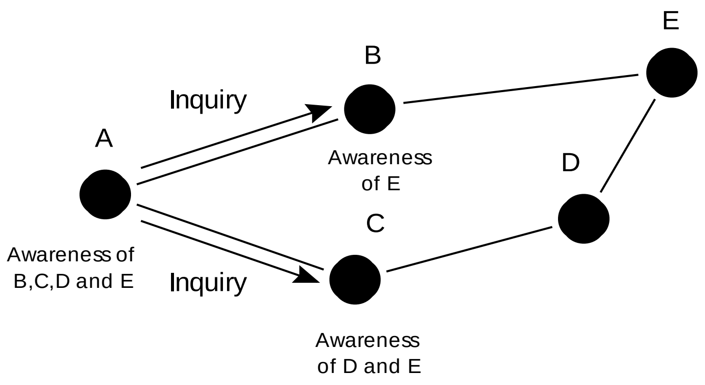

Updated PeerHood Network information
Laptop 2
PeerHood network information
PC
PeerHood Network

Figure 3.5 Network information transferring

- Bridge: Bridge address is the gateway node to connect when we want to connect to another remote device which is not inside our local coverage. After the remote connection petition, every bridge analyzes its own device list table and selects the suitable node to continue the connection establishment.

- Jump: the number of jumps of nodes to get to the final device. Direct devices have jump number as 0. This parameter is considered as the cost of the connection.

In the example presented in Figure 3.6 there are five elements: A, B, C, D and E. In principle B can only see its direct neighbours A, B and C inside the same coverage. Meanwhile, D is aware of the presence of device E. During the device information searching process of B, the whole neighbourhood information (DeviceStorage) of D is also sent and the new device E will be stored in B’s DeviceStorage with the corresponding Bridge device name and number of jumps. Finally, A will also be aware of the presence of E and D after it analyses the neighbourhood information of B and C.



DeviceStorage A		
Neighbours	Jumps	Bridge
Device B	0	empty
Device C	0	empty
Device D	1	C
Device E	1	B

Figure 3.6 Dynamic device discovery

Whenever A wants to connect to remote device, such as E, the only extra parameter it needs to know is the bridge name, which in this case is B or C. After the connection petition arrives to B or C, they are in charge of selecting the next step to achieve the final connection between A and E. However, as the network size increases, the number of ways to reach a device approaches infinity. The size and process limitations make it so that it is impossible and unnecessary to store all of the possibilities to connect with the remote device. The optimal way is required to guarantee the optimal size of the storage and reliability of the connection.

3.4 Essential parameters

In a real PeerHood environment, the distribution of devices is totally random and there exist infinite possibilities to reach the destination. As we commented before, the best route selection is necessary to guarantee the optimal size of information storage [17]. For each device, the number of jumps is the best cost parameter to determinate the time delay and traffic generated for the network. Bigger number of jump means also more transferring traffic and connection delay. However, there still will be several routing options with the same jump number and more patterns are needed to select the most efficient way. The next discussed parameters have been taken into account during the device discovery implementation.

3.4.1 Link Quality

One of the most important new parameters of the neighbour devices is the link quality value. Due to the dynamic nature of mobile devices, the link quality is changing continuously. A weak link quality might mean the device is almost leaving the coverage area and the probability for connection loss is higher. For each network technology there is a different link quality parameter and different way of use.

For Example, in the Bluetooth protocol, Received Signal Strength Indicator (RSSI) is what measures the existing connection parameter. During the device discovery four short duration connections will be established to get the remote device, service, prototype and neighbourhood information as presented in figure 3.7. RSSI could be obtained by listening to the connection channel during this short connection time and stored as link quality parameter. Furthermore, we could unify these 4 short connections to an only one longer connection to get a more reliable value. However, there can be differences of link quality parameters between different manufacturers. Thus, further studies are needed to determinate the suitability and reliability of the link quality value.

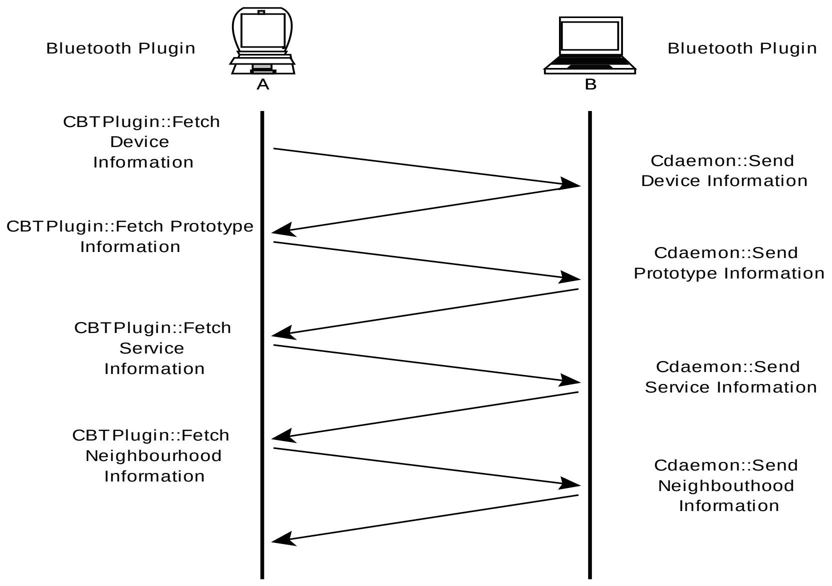

Bluetooth Plugin
A
Bluetooth Plugin
B
CBTPlugin::Fetch Device Information
Cdaemon::Send Device Information
CBTPlugin::Fetch Prototype Information
Cdaemon::Send Prototype Information
CBTPlugin::Fetch Service Information
Cdaemon::Send Service Information
CBTPlugin::Fetch Neighbourhood Information
Cdaemon::Send Neighbouthood Information

Figure 3.7 Bluetooth Plugin discovery process

To find the route with the best quality, more than one measurement is needed. In next figure there exist two possible routes for device A to connect D: A-B-D and A-C-D. As we described before, quality parameter should be achieved by listening to the short connection established to get device and service information. Thus A can only get directly the quality parameter of B and C. These are not enough to assure the best route. To solve this need, the link quality parameters for B-D, C-D will be stored in each device's information field and sent to A as neighbourhood information as well.

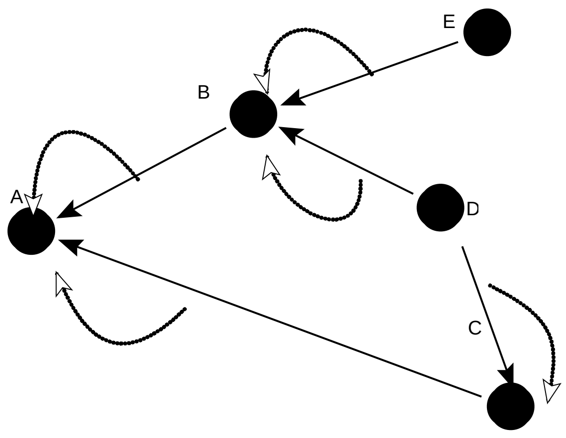


Figure 3.8 Link quality storage algorithm

Once we have link quality parameters from two routes with the same number of jumps, we will select the route with biggest absolute quality value. If AB + BD are bigger than AC + CD, route A-B-D will be stored as the definite route.

One curious case is the equity of the quality parameter's addition. In the case presented in figure 3.9 the result of addition of both routes are the same, which would be the best route for the connection? Particularly we think once the quality value is higher than the minimum demanded, both routes are suitable for the connection. In this case, the route A-C-D won't be accepted due to A-C being lower than the minimum threshold 230.

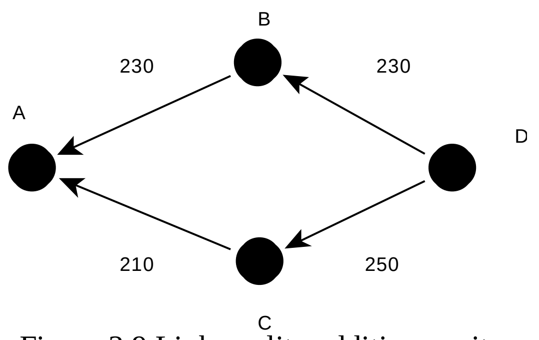


Figure 3.9 Link quality addition equity

3.4.2 Device & Service Search Time Delay

Whenever we have an extensive PeerHood network, several jumps are needed to detect all possible neighbour devices. Due to the nature of the searching engine, the bigger the network is, the bigger is the possible maximum delay to any eventual change of situation.

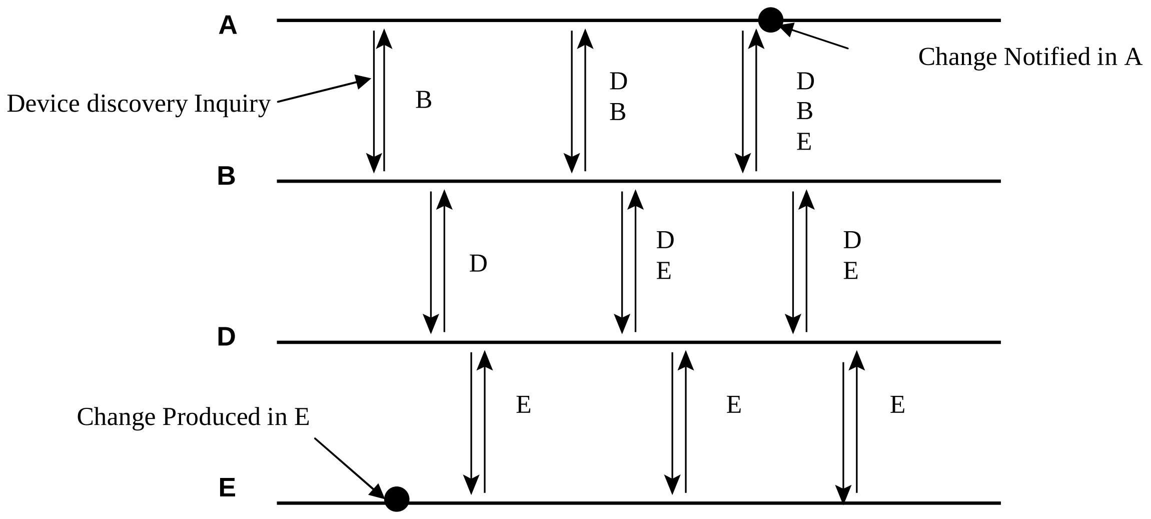


Figure 3.10 Maximum Time Delay

If we have a device A which is situated two bridge nodes B, D far away from E, we see that the maximum time delay of A to detect any change of E would be two whole device searching cycles in the imagined worst case presented in figure 3.10. It means Max Delay = Num Jump * searching cycle time. Moreover, if the prototype of the device is Bluetooth, the maximum delay might be even bigger. According to the asymmetric characteristic of Bluetooth in the device discovery process [4], when a given device is searching for other devices and services, it is not discoverable by other devices. There exists the probably that on random occasions the Bluetooth device won't be searched by the discovery inquiry and thus the notification time of any change in the remote device which requires several jumps would be much bigger. Thus, a limitation of Num Jumps for moving devices should be taken into account depending on the network technology. A big time delay would cause connection loss frequently due to the late knowledge about the environment.

3.4.3 Static & Dynamic

We classify the devices to three big groups: static, dynamic and hybrid. Static terminals are usually fixed providers of services and their behaviour is completely different from dynamic mobile devices due to the permanent position and electricity supply. Static terminals are more suitable for functioning as a bridge between other devices. There are less possibilities of connection loss, the device searching cycle can be shorter and the energy consumption spent in the data transmission won't be taken in account. Thus dynamic devices are normally clients that give the maximum priority to low battery consumption and it's not suitable to carry out connection retransmission due to the resource consumption and the mobility characteristic. Hybrid devices could be low mobility mobile devices or static devices that want to reserve their own resources and limit the bridge retransmission function. In the device searching algorithm we will always give preference to static terminals as a bridge so that the network traffic will concentrate on them and consequently converting them to the backbone of the network. In next figure we have two different scenarios with static and dynamic devices as bridge, where we can observe clearly which is the most reliable connection configuration.

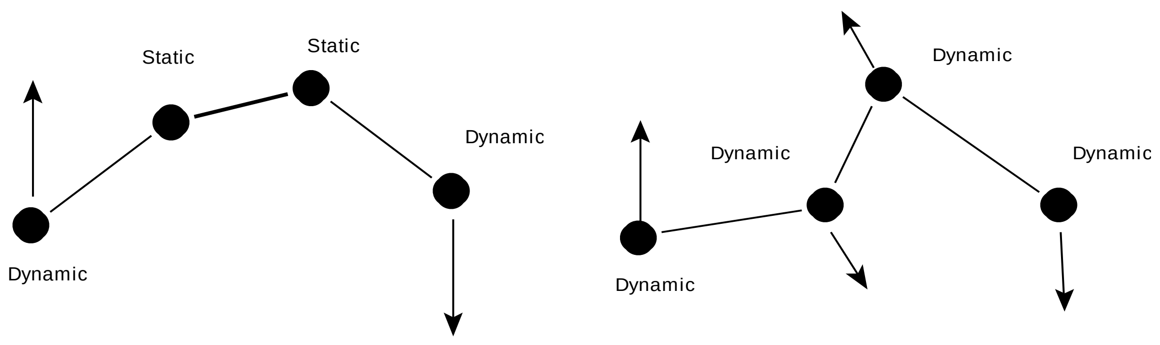


Figure3.11 Static & Dynamic Bridge

Mobility values are added to Daemon as a system parameter in the initialization. Respectively they are {Static, hybrid, dynamic} = {0, 1, 3} to make easier the comparison during the device discovery process.

We also have considered the possibility to make the addition of mobility parameters in the same way as link quality in situations where there exist more than one routes with several number of jumps. Taking the same scheme of figure 3.10 we will have the following table of possible values.

0 + 0	0 + 1	1 + 0	1 + 1	0 + 3	3 + 0	1 + 3	3 + 1	3 + 3
Static static	Static hybrid	Hybrid static	Hybrid hybrid	Static dynamic	Dynamic static	Hybrid dynamic	Dynamic hybrid	Dynamic Dynamic

0	1	1	2	3	3	4	4	6

As you can notice, the smaller the mobility number is, the better would be the stability of the connection. However, currently we consider it's important to maintain the single mobility value, because it is an important device property. Therefore only the nearest device's mobility numbers are considered.

3.5 Discovery Process

To find the surrounding devices information, Bluetooth Plugin uses an inquiry thread that is continuously searching neighbourhood information. At the same time, a SDP query is also used to find the PeerHood tag to identify PeerHood capable device. For every inquiry loop, a certain number of other device's responses are received. For each response device information will be fetched and stored in the Plugin's device list and later stored definitely in DeviceStorage. Time stamp is used to check the device's existence. If one device doesn't respond to the inquiry during certain loop, it means the device has probably already left the coverage area and the device information should be removed from the device list. PeerHood considers it not to be necessary to establish information fetching connection with an existent device in every discovery loop. A service checking interval defines a longer interval time for stored devices to achieve the energy saving. The detection process is described in figure 3.12.

Comparing to the previous version of PeerHood BTPlugin, the analysis of the neighbourhood is the main new element of the discovery process to analyze the neighbours of each response device according to the most efficient way principle. New direct devices will be added to the device list with its corresponding incremented jump number and bridge address. Own device comparison filter is used to avoid duplicated route. And jump, mobility and link quality number are used to select the best route once a previous route is already stored in the device list. The implementation diagram is presented in figure 13.

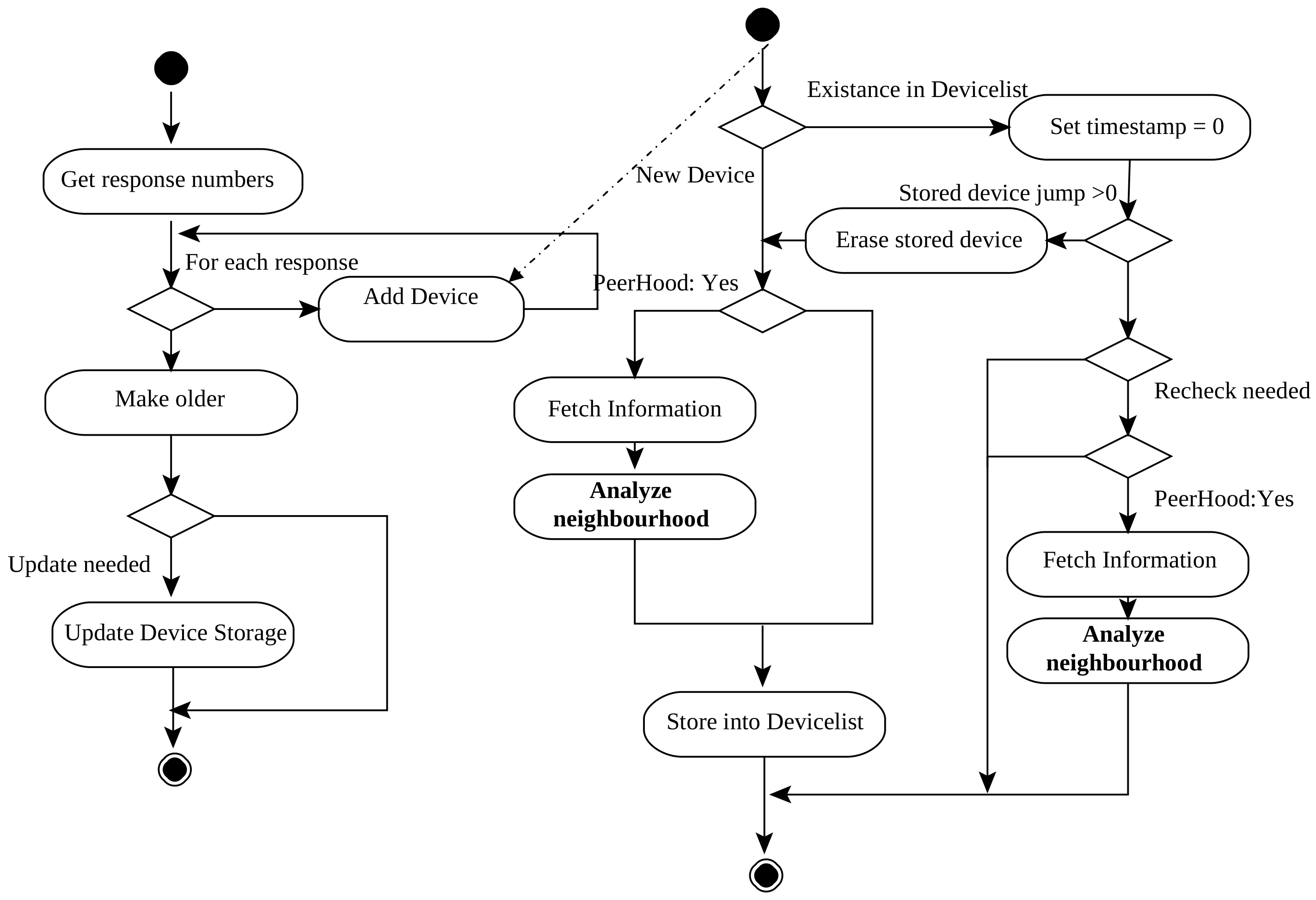


Figure 3.12 BTPlugin activity diagram

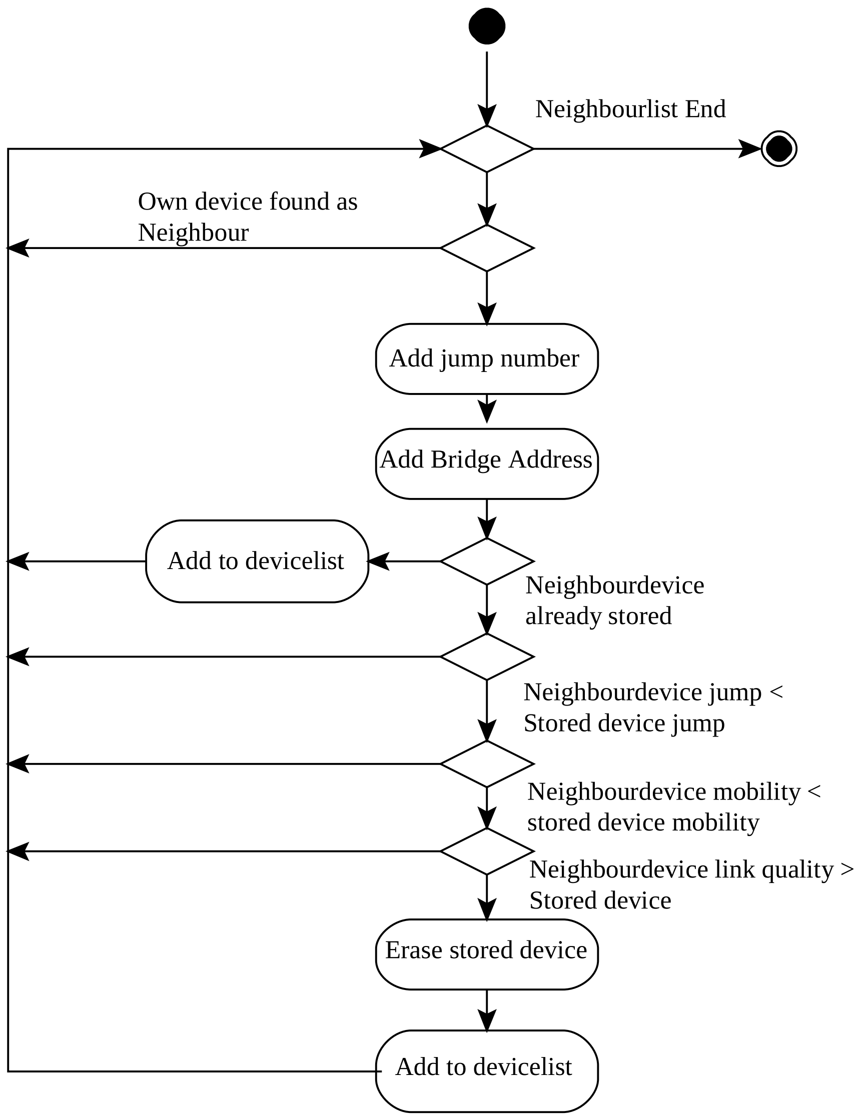

Neighbourlist End
Own device found as Neighbour
Add jump number
Add Bridge Address
Add to devicelist
Neighbourdevice already stored
Neighbourdevice jump < Stored device jump
Neighbourdevice mobility < stored device mobility
Neighbourdevice link quality > Stored device
Erase stored device
Add to devicelist

Figure 3.13 Activity diagram of AnalyzeNeighbourhoodDevices

Nevertheless, this implementation of PeerHood Device Discovery is carried out mainly to demonstrate the viability of the new device discovery. The mentioned implementation would work appropriately only in case we that have one Plugin. The reason is that in a normal situation, the DeviceStorage is also shared and accessed by other network Plugins. Although a critical zone control is created it’s unviable to lock the DeviceStorage during the entire Inquiry thread due to the low speed of information fetching process. To avoid the mentioned problem, the design of previous PeerHood Plugin should be changed. There might be one local device list to control the service checking interval, all the information fetching process should be done before accessing the DeviceStorage and the data processing will be done during the update phase of DeviceStorage.

Chapter 4

Interconnection System

The main goal of this thesis is to search one solution to the mobility problem during the task migration. If Dynamic Device Discovery was thought to get a better awareness about neighbourhood environment, the Interconnection functionality is created with the target to allow connection to remote device through different network nodes. In the next example (figure 4.1) if device A wants to establish connection with remote device E, B would be the bridge node which A will try to connect. Moreover, Device B should be notified the connection intention of A as an intermediate connection, receive the final destination address and service and select the next suitable bridge which is C. Device C will receive the connection request from B and establish the connection with final device E. After the connection establishment, B and C will limit to re-transmit every data they receive between A and E.

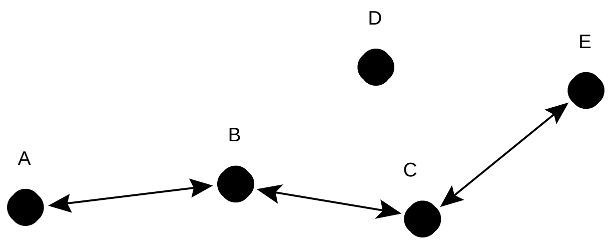


Figure 4.1 Interconnection between 2 devices

To achieve this target, the interconnection system gives every device the chance to be a bridge node to redirect the traffic to other device. One hidden bridge service will be included in each PeerHood package and executed in the initialization of Daemon. Bridge service listens continuously for connection requests in order to establish a new connection with the next bridge or final destination. The suitable prototype and route selection of next connection will be always carried out by the bridge server and not the original device. The scheme is presented in figure 4.2.

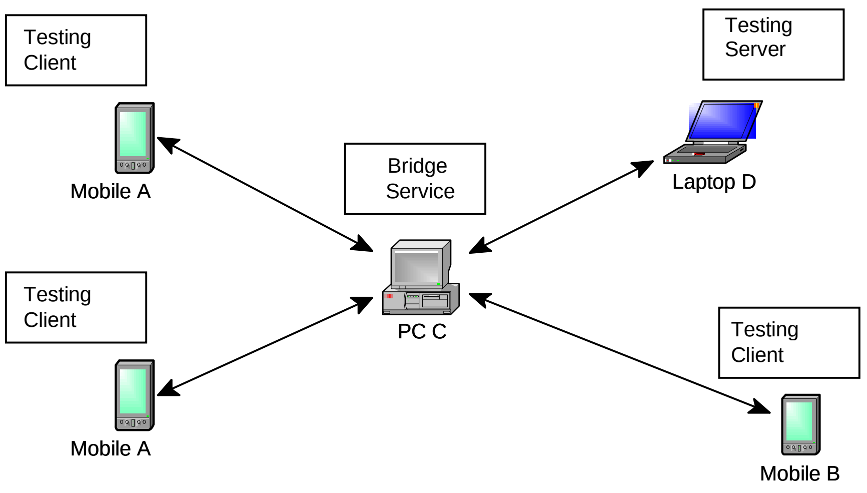


Figure 4.2 Multiconnection bridge service

Although static devices with high link quality are more likely to exercise the bridge function, in a totally random distribution stage, bridge service could also be needed to run in the mobile devices. In many cases multiple connections should be allowed at the same time in order to satisfy the random mobile environment. However, in the last case the bridge service would produce extra processing need and energy consumption to the node device. Due to the battery limitation of mobile devices, this situation is highly undesirable for them, but at the same time unavoidable. One of the possibilities is switching off the bridge service of devices that have the mobility parameter as “mobile”, although the network performance will be seriously affected due to the decreased visibility. Other option is that the maximum connection number is adjusted by the device owner and whenever the maximum is reached, it is notified back to the request device. Nevertheless, as the device discovery process will always try to find only the best connection route measuring link quality, it would be very interesting to modify the link quality value according to the maximum connection number and avoid the “bottle neck” situation. An extra connection number/maximum connection number percentage could be transmitted during the device discovery process and proportionally the link quality parameter is decreased.

4.1 Connection Process

In principle, the bridge connection process doesn’t differ too much from the normal connection process of PeerHood. Before we start to analyze the bridge connection process, three important types of classes should be clarified to get a better understanding.

- Applications: Applications are the highest level class of PeerHood protocol. In this level the connection is created through PeerHoodImp::Connect and incoming connection is notified by class Engine.
- PeerHoodImpl::Connect: The connect method of PeerHood library includes all connection steps and parameter exchange with the remote device. This method should be called from application level or special system monitoring threads.
- Engine: Engine is the PeerHood class which is continuously listening for possible connections in different network technologies. Once connection is recognized and accepted, it will proceed to identify the connection intention to discover if they are new connection, bridge connection or connection re-establish. Therefore different connection parameters and received according to the connection type.

It’s important to understand the singleton design pattern of PeerHoodImp library and Engine. This method ensures that at any given moment of time only one instance of mentioned class is running, while many applications are allowed. Any network event to the application will be notified by the engine using methods included in application callback class.

Basically there exist two main differences between the normal connection process and bridge connection process. First of them is that there exists the need to transfer the destination address and service name to the bridge connection. The bridge service will receive these two parameters from engine callback function and proceed to find the next step to continue with the connection. The second of them is the connection acknowledgement. Due to the fact that connection is constructed by more than one connection between different nodes, if one of them fails all the connection chain would fail and it should be notified to the connection request device. In figure 4.3 the connection process is described with detail.

In next figure we can see an interconnection example between two remote devices using a bridge service. As we can notice, the interconnection consumes double amount of time. Although this time consumption is totally logical and unavoidable, the maximum connection time will seriously limit the application performance and should be taken in to account according to different network connection speeds.

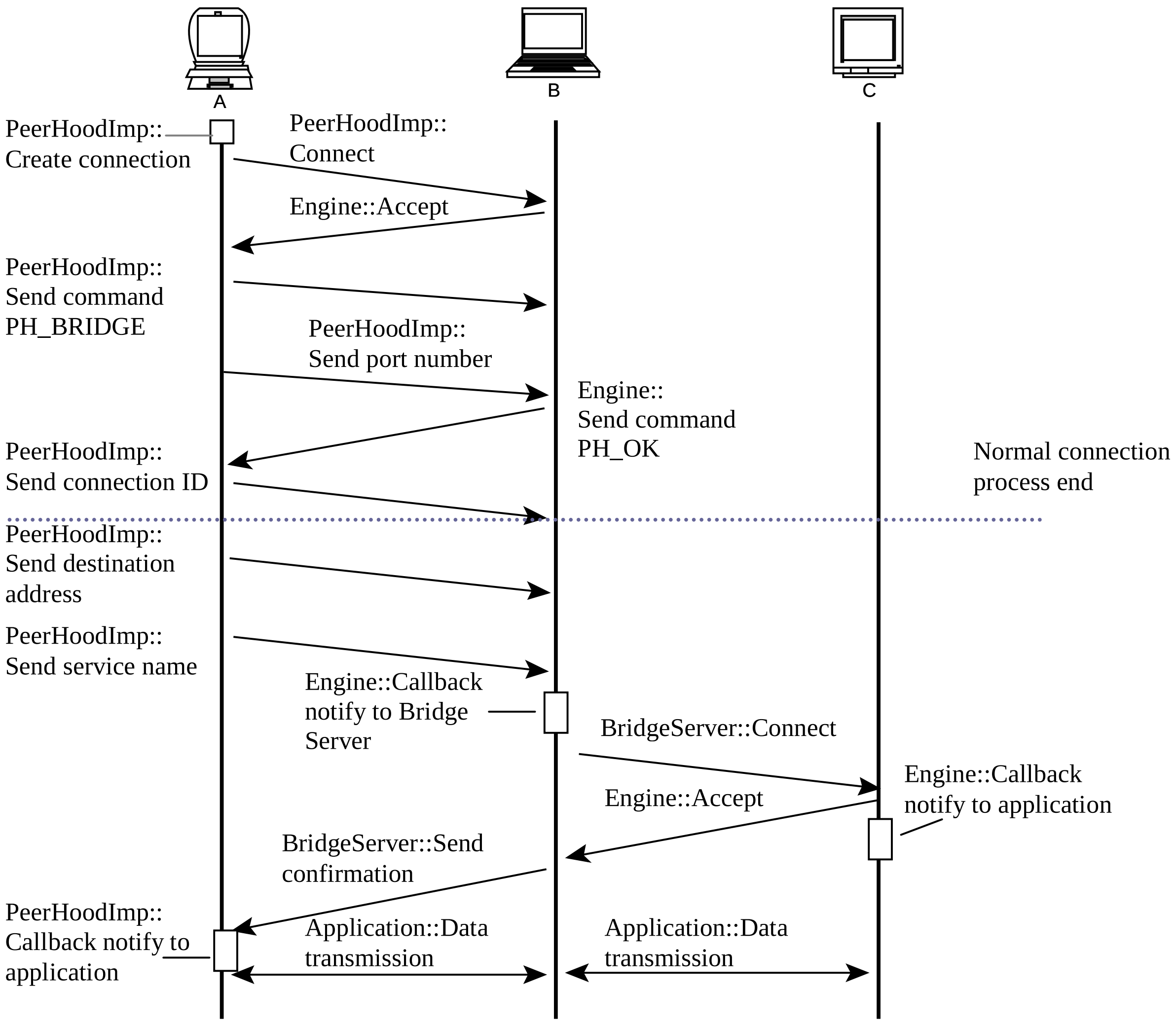

A
B
C
PeerHoodImp:: Create connection
PeerHoodImp:: Connect
Engine::Accept
PeerHoodImp:: Send command PH_BRIDGE
PeerHoodImp:: Send port number
Engine:: Send command PH_OK
PeerHoodImp:: Send connection ID
Normal connection process end
PeerHoodImp:: Send destination address
PeerHoodImp:: Send service name
Engine::Callback notify to Bridge Server
BridgeServer::Connect
Engine::Callback notify to application
Engine::Accept
BridgeServer::Send confirmation
PeerHoodImp:: Callback notify to application
Application::Data transmission
Application::Data transmission

Figure 4.3 Bridge Connection Process

4.2 Bridge Service

Bridge service is implemented like other PeerHood application using the library functions. One abstract connections list will store all the connections from both directions. As every connection in bridge requires one pair of connections, each incoming connection will be stored as even and the corresponding connection in other direction as odd to avoid the creation of two connection list. The BridgeConnection method is called from Engine using callback system and it is responsible to find the next node to connect, create the connection, store them into the service connection list and write the acknowledge command to the request owner. The main loop is listening continuously from file descriptor of every listed connection and once traffic is detected from one direction, it will be sent to the corresponding connection in other direction until one of the connections is over. Then the pair of connection would be removed from the connection list. The activity diagram of BridgeService is presented in figure 4.4 in next page.

The implementation has been taken into account the following patterns:

- Bridge service should be bi-directional in order to accept traffic from both sides. Even and Odd are used to distinguish the connection direction.
- BridgeConnection method is callbacked from Engine to establish new connection once PH_BRIDGE command is detected.
- Multiple connections should be permited to achieve a real Bridge functionality.
- As connection list could be modified by main loop and BridgeConnection, access control is necessary to avoid undesired index confusion even the time interval is small.
- After the connection establishment, bridge won’t interpret the traffic. Every traffic data it receives will be sent directly to the destination, with the exception of disconnection. In this case, corresponding connections are disconnected and erased from the connection list.

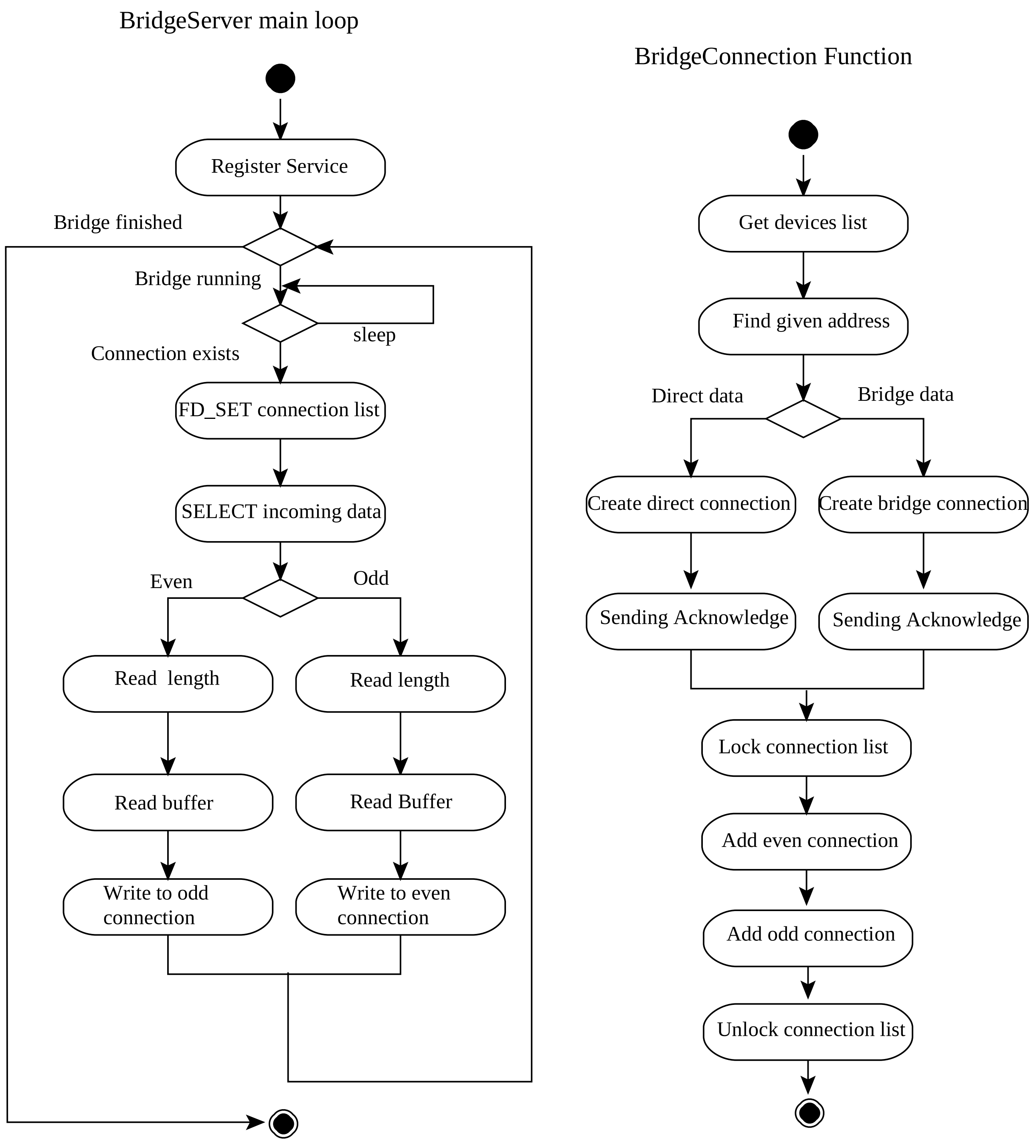


Figure 4.4 Bridge service & BridgeConnection function activity diagram

4.3 Performance Testing

The performance of BridgeService was tested with two simple clients and one server. The configuration is presented in the following figure. The function of the client is to send a message 20 times with 1 second of intervals to the server through the bridge and server will just print the message in the screen. Bluetooth was the chosen network protocol.

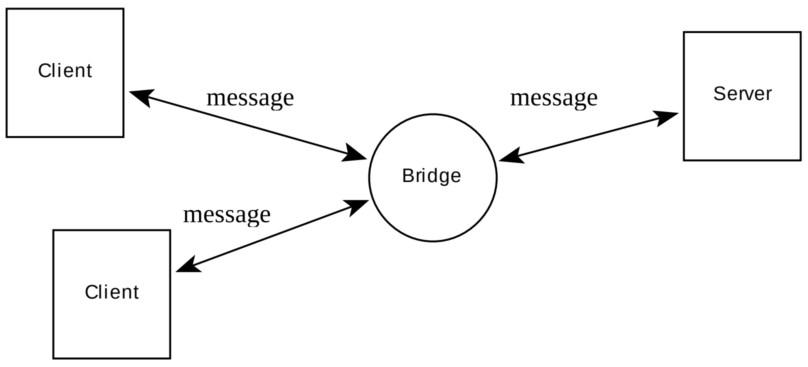


Figure 4.5 Test connection configuration

The test was carried out with several attempts to check the average connection performance. In these ten connection attempts, three of them couldn’t be done due to the normal Bluetooth connection fault between client and bridge. In other seven successful connections, the time needed for the connection was between 3-18 seconds. The sending and receiving of data packages were carried out perfectly with an almost negligible time delay.

Although the initial connection establishment takes a long time, the negligible time delay of data transferring among different nodes is an important factor that demonstrates the viability of interconnetion. Also we found the connection fault is quite frequent during the connection establishment process even if the devices have strong enough signal. To avoid this problem, the connection attempt repetition in the Bridge service design would be necessary to guarantee a satisfactory connection. Further applications also need to be modified similarly.

Chapter 5

Task Migration

5.1 Migration scenarios

After achieving the total environment awareness and the interconnection capability for the devices, we are ready to discuss about task migration in a constant changing environment. First of all, in this project we consider the task migration's benefit for the mobile devices are evident and already demonstrated in several previous studies. Second of all, the typical task migration mainly consists on the transmission of certain data from one mobile device to other device which has the capability/server to solve the task more efficiently, later the result is sent back to the mobile device. Thus we will concentrate our study to two different stages of the connection during the task migration process. Respectively they are:

1. Mobile device is interchanging information continuously with the service, the connection is needed permanently.
2. Mobile device sends data to the service, the server will process the data and the result will be sent back to the mobile device. The connection is not needed permanently.

In a real PeerHood environment like figure 5.1, the connection could be lost in any time due to the random mobility of devices and random network distribution. In this chapter we will try to find how to avoid the connection lose and carry out the task depending different scenarios.

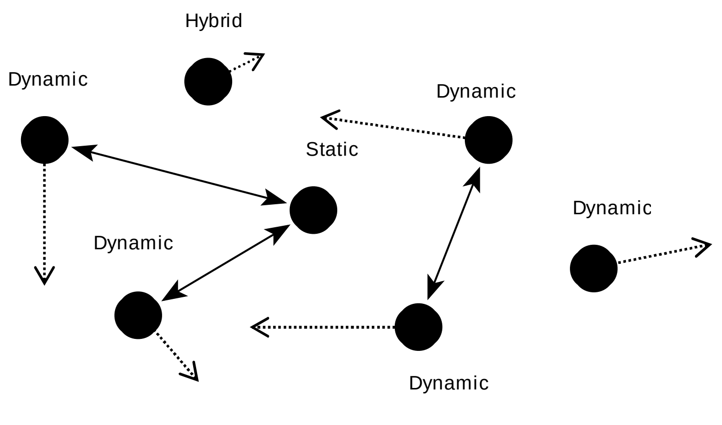


Figure 5.1 Constant changing scenarios

5.2 Soft Handover

Handover process is commonly used in GSM and 3G mobile communication. In figure 5.2 the typical situation is presented. The mobile device is leaving the coverage area of the first base station. After the signal becomes low a second connection is established to the second base station at the same time. Once the hysteresis threshold is overcome the connection will definitely transfer only to the second base station.

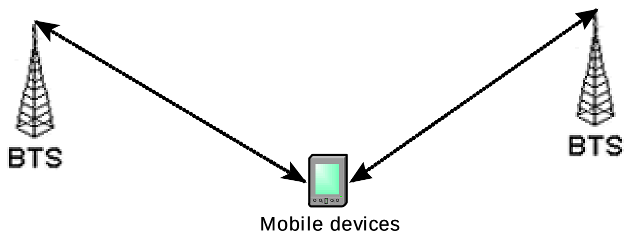


Figure 5.2 Soft Handover

In this work we tried to implement the same Handover functionality to PeerHood. One HandoverThread was created to listen to the link quality and check the available near servers. Whenever the quality gets weak due to the device’s movement, HandoverThread can detect it and will try to continue the service by establishing a second alternative connection way.

However, there exists a fundamental difference between cell phone communication system and PeerHood. In the case of GSM, all the traffics in base station have a common destination that is the MSC (Mobile Switching Centre) which is the controller of the whole network traffic. BTS (Base Station Controller) doesn’t process any data and everything it receives is sent to MSC. In other words, the GSM network consists on many access points (BTS) and a unique server (MSC) which is in charge to interconnect cell phones calls and data transmission. Nevertheless in PeerHood environment there could exist unlimited kind of service inside any kind of device. And even the same service is present in various devices, every service is in principle independent from each other. The change of service location would mean a complete reset of the application that is not desirable in the most of cases [7].

In next figure we have a typical task migration stage to demonstrate the mentioned problem. Normally mobile devices are not able to process high quality images, instead of a long time and high energy consumption processing, optionally they can transfer the images to the near PeerHood static server to carry out the task, and receive the result back. In moment A the mobile device is connected only to the server 1 and the image transmission is started. In moment B the link quality is becoming weak and according to the design, HandoverThread found the nearest device server with a good enough link quality to establish the alternative route. However, after the connection broke with server1. Even we are connected to the same picture analyse service of server2, the whole task migration should start again due to the inexistence of connection between server 1 and 2.

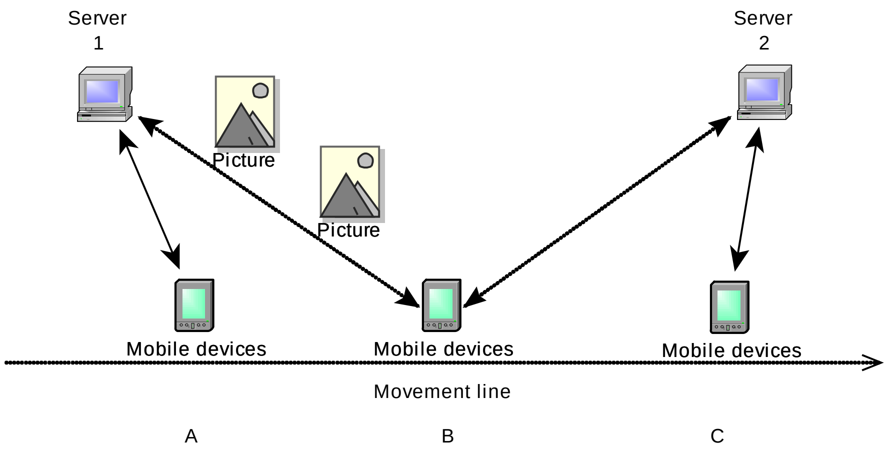


Figure 5.3 Two servers handover

Obviously the service reconnection is necessary when we don’t have any other choices and it’s the only way to carry out the task migration. Although before we would like to comment another way to maintain the same connection to the server: the Routing handover.

5.2.1 Routing Handover

In PeerHood environment whenever the mobile device are moving, device discovery process is constantly detecting the change of neighbourhood. According to the moment and network distribution, many alternative routes will be available to connect two devices. If we consider the example of figure 5.4 where the mobile device is connected to the laptop to do any task, as the mobile device is leaving the effective coverage area, the connection could break in any moment. It's interesting to see once the device is leaving from the laptop, it is approaching to other one that has good connection signal with the laptop. Therefore the same connection could be kept doing the interconnection among these devices. We can summarize this stage in 5 main points.

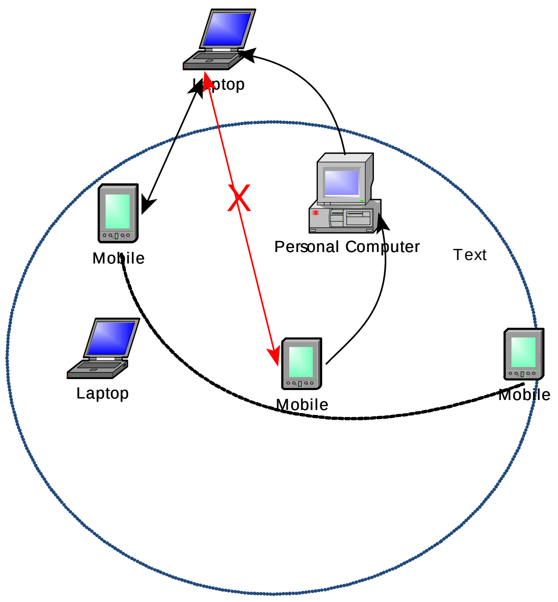


Figure 5.4 Routing Handover

1. The service provider is a direct neighbour inside the initial coverage area as other devices.
2. Many of these devices also consider the service provider as one of their direct neighbour.
3. Whenever the mobile device is leaving from the service provider, it is approaching to other direct neighbours.

4. The same connection could continue using a direct neighbour as bridge node to connect to the service provider.
5. Link monitor is measuring the link strength with neighbours.

One important supposition we did is that we assume the speed of the mobile device is not high enough to change the whole direct neighbourhood environment in few seconds (case Bluetooth). If the device's velocity is too high the device discovery could not update appropriately the neighbourhood. Most of neighbourhood information will be wrong and the handover system won't work.

Thus we can summarize the routing handover in 3 main states. The Activity diagram of the HandoverThread is illustrated in figure 5.5 .

- State 0: HandoverThread Gets DeviceList from Daemon and searches for the actual connection address in each device's neighbourlist. The link quality of each new route is checked and the highest quality route is stored.

- State 1: Monitoring the link quality of the existing connection. We consider if the signal has been too low for 3 times it means the degradation of the connection and we go to the state 2.

- State 2: HandoverThread Create a new bridge connection to the intermediate node with the stored route. Once the connection is confirmed, the application will be notified by the callback ChangeConnection method and the connection will be substituted.

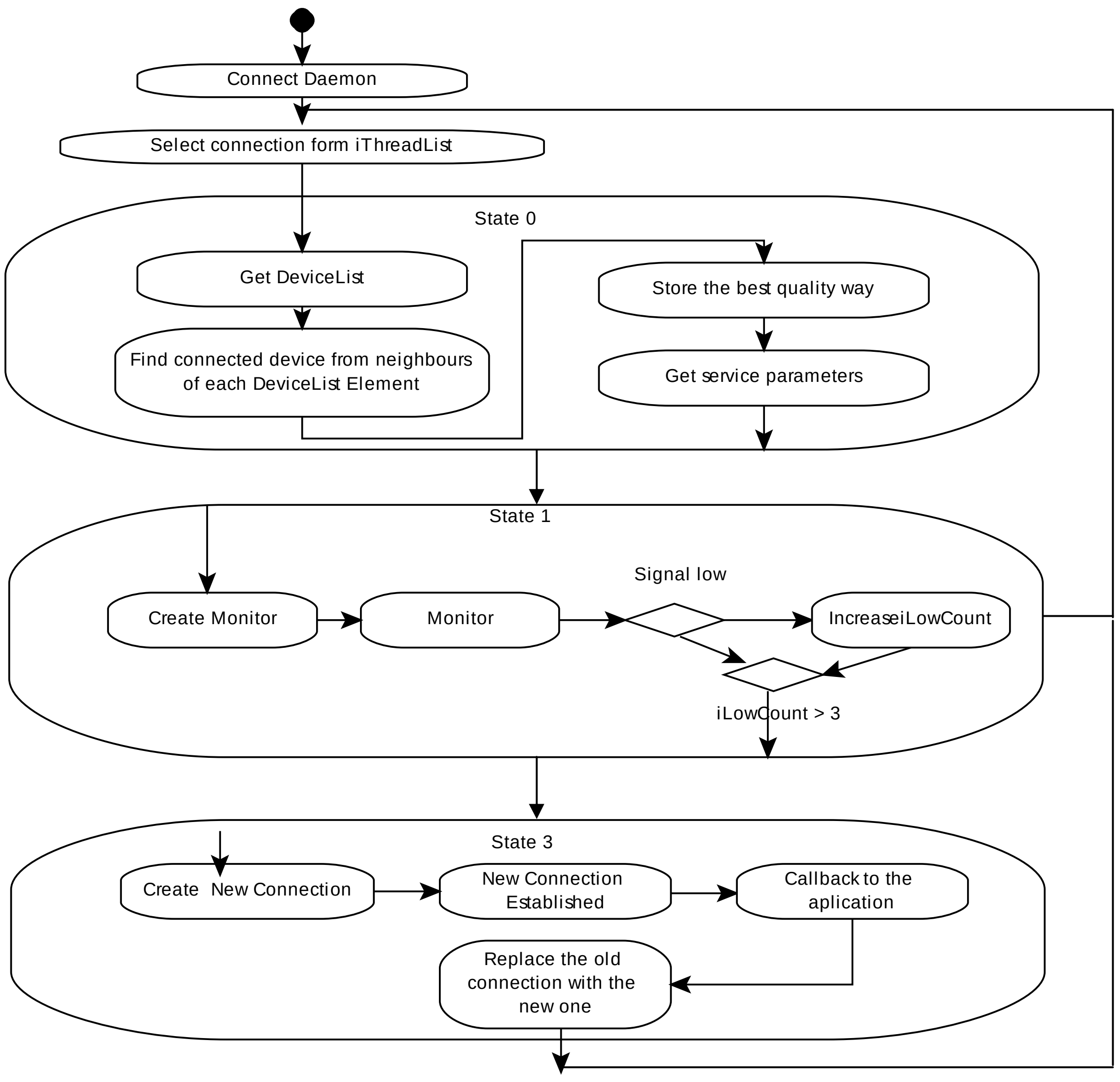

Connect Daemon
Select connection form iThreadList
State 0
Get DeviceList
Find connected device from neighbours of each DeviceList Element
Store the best quality way
Get service parameters
State 1
Create Monitor
Monitor
Signal low
IncreaseiLowCount
iLowCount > 3
State 3
Create New Connection
New Connection Established
Callback to the aplication
Replace the old connection with the new one

Figure 5.5 Routing Handover Diagram

During the implementation testing we proved the interconnection functionality implemented in last chapter allows the HandoverThread can choose freely the alternative route and makes the routing connection a viable solution. Even though there exist implementation challenges to make it really applicable to PeerHood and achieve the expected result. The one we found was Monitoring limitation.

<u>Monitoring limitation</u>

According to the principle of node independence, the second HandoverThread will be created by the bridge to listen the connection established with server. Whenever the server device is leaving from the bridge coverage area, this HandoverThread will try to connect to the next bridge to continue with the connection and so on. This system makes every HandoverThread is responsible only for the connection it is listening to and has the autonomy to decide to change the connection route.

In figure 5.6 we can see a normal stage of the routing handover. The client A is moving away from the connected server A. When the link strength becomes low the connection is interconnected through bridge B and later also through C. This is the ideal performance of routing handover.

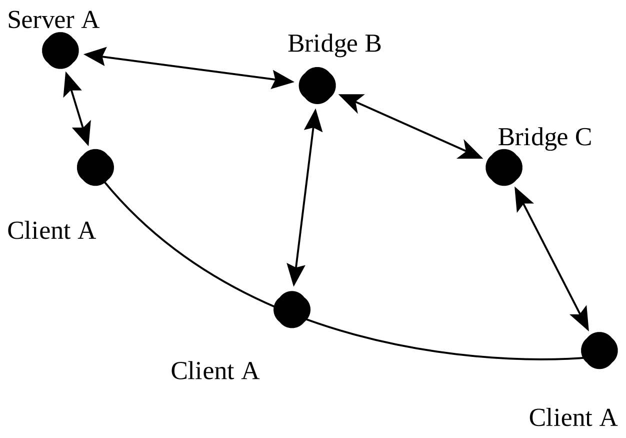


Figure 5.6 Bridge routing stage A

Nevertheless in situation like figure 5.7, the client A is coming back to the initial point but the HandoverThread of Bridge C only considers the possibility to continue the connection from itself. The result is an inefficient connection using unnecessary bridge nodes. We still didn’t find the solution to this implementation problem.

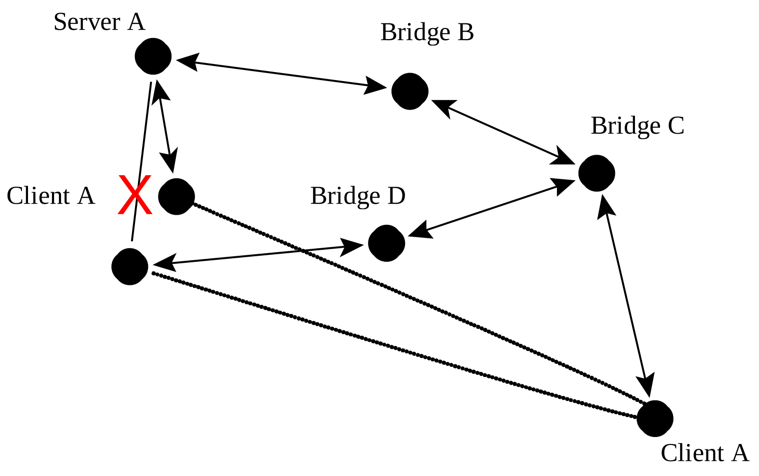


Figure 5.7 Bridge routing stage B

The simulation of Routing handover is done with 3 devices following the distribution of next figure. In first place a client application B connect to server A to print the message “good morning!” 50 times in the server’s screen. Due to the difficulty to distribute the computers with enough coverage separation, we simulate the first connection deterioration subtracting the monitored link quality value artificially by 1 every second. Once this value is smaller than threshold 230, the signallow account increased. And when this account is bigger than three, the HandoverThread will proceed to change the connection to the second route.

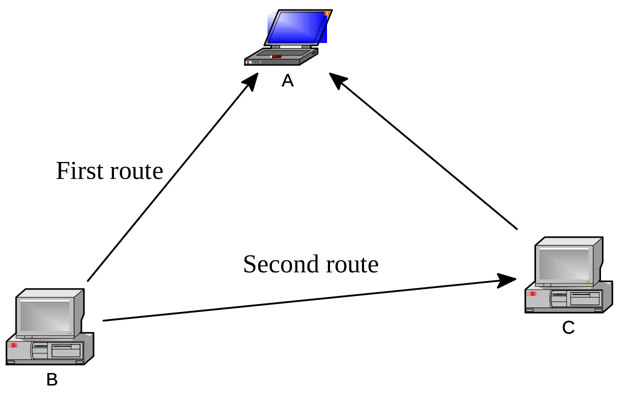


Figure 5.8 Handover simulation stage

The simulation was repeated several times to get an average performance of routing handover. Apart of the connection fault errors produced during the interconnection process, which is commented before in the last chapter, the connection changes were carried out with the same time delay like a normal interconnection process without any problem. After we took the laptop from the office to the corridor during a connection with B and we observed the decrease of Bluetooth link quality parameter is really fast and we can lose the connection in few seconds with a normal walking speed. If we add also the interconnection time that would be from 4 to 15 seconds. More than probably the connection will be lost before we achieve the second route connection establishment. This huge connection establishment in Bluetooth is a serious obstacle for the theoretical functionality of routing handover.

5.2.2 Service Reconnection

Whenever the Routing handover is not possible (no suitable Bridge device around to get to the same connected device) or the routing path is incorrect and after various attempts the HandoverThread couldn’t restablish the connection with the old device. After the connection is definitely broken (HandoverThread is continuously listening to the channel quality). PeerHood will try to connect to another service provider device. As we commented before about the independence of service owner devices, unless some really specific services are based on asymmetric traffic from server, all the information needed for the task migration is required again from the client and the application should be restarted. The process is identical to a completely new connection process but made by HandoverThread.

```
HandoverThread

Find the same device in neighbour's neighbourhood
Devicefound = true

Find the service in neighbourhood
Servicefound = true

If (connection == broken && Servicefound == true)
  If (routing handover attempts > limit || devicefound = false) {

Reconnection option to client
Service Reconnection
}
```

We consider it's preferable to notify to the application user about the reconnection need and let him to give the permission to the service reconnection. Depending on the application and transmission need some times the user would prefer to quit the connection if he has to initialize the connection again from zero.

5.3 Result Routing

In most of the task migration process, once the information of client application is sent to the server the connection is not necessary until the server finishes the task processing. After the client has already sent all the information, it will remain to a sleeping state waiting for the result back. We have observed in this case if the connection breaks the connection is not needed to be repaired immediately due to the unknown data processing time of the server. And any attempt of client to reconnect to the server would be inefficient due to the connection only is needed after the result processing. Thus we consider the optimal would be the server establishes the connection with client after the data processing. An example of picture analyse migration is presented in the in figure 5.9.

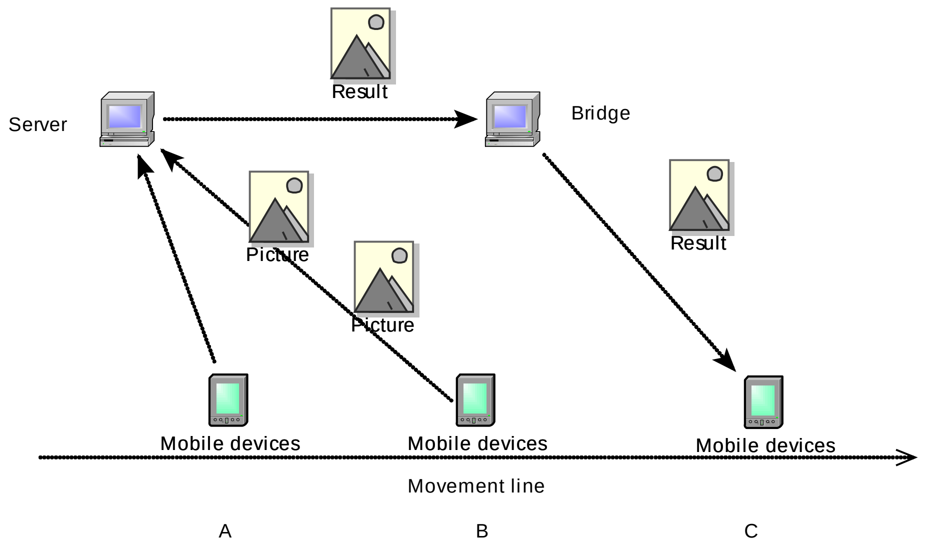


Figure 5.9 Waiting for response

To achieve this configuration in PeerHood, a new boolean variable “sending” is added in iThreadlist. Getsending method is called from the application to change the value to indicate the end of data sending or viceverse. Thus when the link quality becomes low the HandoverThread will be aware about the no need for the reconnection and avoid the routing handover or service reconnection.

A testing server and client application are created to demonstrate the functionality of waiting for response system. The server is simulating an image analyse server which receives a big size photo from any client, the people from the photo will be recognized and names are added in the same picture and sent back to the client. The implementation of the server is similar to other PeerHood application using only the library functions. One connection is allowed at the same time.

The client is simulating a mobile device which will send a picture to analyse in the remote server. First the client will send the size of photo (package numbers) and then each data package. After the data sending it will simulate the device movement disconnecting from the server, and enters to the sleeping state waiting for the image analyse server’s connection and receives the result.

During the implementation we realized once the connection is broken, the server has not enough information to reconnect to the client. Due to the previous design of PeerHood, once a new connection from client is received in server, only connection ID and service port number are received as connection's parameter. To be able to establish a new connection to the client, prototype, device address, service name, device name , Pid and port number also are necessary values that the PeerHood engine can't provide. We consider there are two methods to solve this problem:

1. Clients insert a "client" service to the daemon to provide the connection possibility to servers. This method would increment the number of network service unnecessary and the application will be visible for the whole PeerHood network and make it target of possible attacks. The other inconvenient is the dependence of connection to the device discovery process, even server is aware about the presence of client, it has to wait for the plugins to discover the client device.

2. The mentioned parameter like prototype, Pid number, service name, checksum, device name and port number are sent in the beginning of the connection between client and server. It would be the best option to avoid unnecessary "client" service in the PeerHood network.

In this performance test we have chosen the first option to test the performance due to its relative simplicity. In figure 5.10 is presented the activity diagram of image analyse server.

The performance test was repeated several times with package numbers from very small value to huge size. During the test some connection faults were produced due to the normal Bluetooth limitation. We can summarize the result in following three groups.

1. With a smaller number of data packages the processing time is also smaller and the task could be carried out before the device leaves the coverage area.

2. With a considerable number of data packages the connection is broken during the processing time after the server has already received all picture information. In this case server looks for the device in its neighborhood routing table and tries to send the result back after the task processing.
3. With a huge number of data packages the connection is broken during the data packages transmission. Before the definitive connection loss Handover thread will try to restablish the connection though the neighbor node.

During the experiment we have observed an important event in the third case:

As Bluetooth was the chosen technology for the implementation, the time needed to establish the connection through another bridge node had an average value superior than 10 seconds. Such huge connection time made the mobile device has lost the connection before the alternative connection is done and consequently producing a connection lack affecting the task migration performance. The connection time would be much higher if the jump number of nodes is bigger. Based on this result, we believe the Routing Handover is not suitable for all network technologies but only those have a short connection establishment.

We can also confirm that in the second case, migrated task's result could be sent back from server without any problem. And doesn't need any change inside PeerHood library and engine. This functionality could be added by the application programmers according to the service need and time delay limitation. After we add the total network acknowledge (chapter3) y and interconnection capability (chapter4), the server can easily return the result to the client taking the advantage of the environment with some time delay.

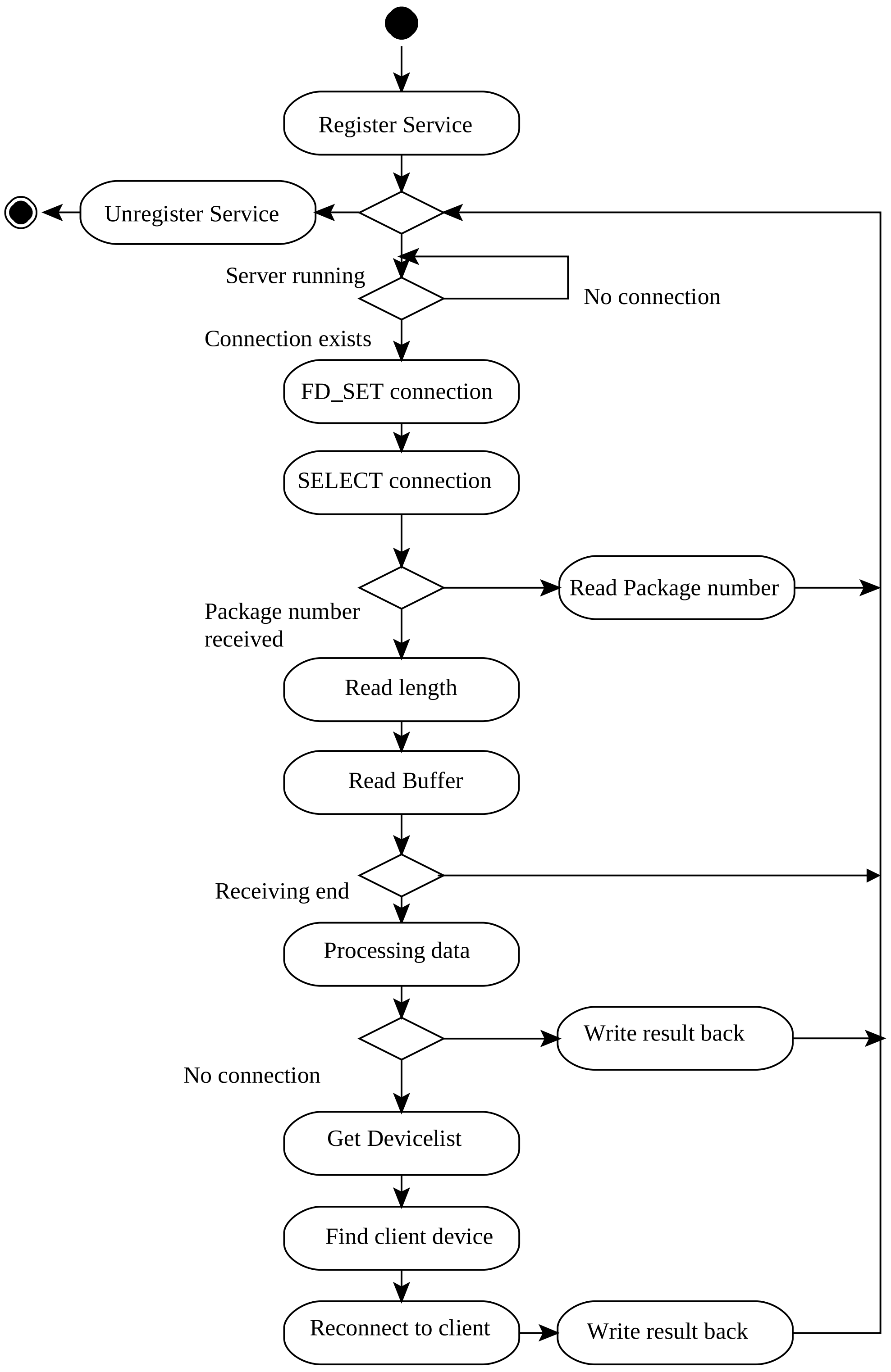


Figure 5.10 Picture analyse Server

Chapter 6
Conclusions

Conclusions & Further work

In this project we have proposed a different design of PeerHood environment and the handover system. However, this work only the first attempt to achieve the mentioned functionality to PeerHood and many lacks of design are needed to improve. Also more real performance test should be carried out in the future to certificate the viability of these ideas.

After working with PeerHood during several months, we want to stand out the importance for PeerHood to achieve the total environment acknowledge and interconnection capability. In this project we did the simple version of the implementations and the results have demonstrated the advantages and viability of these changes.

The most difficult part of this project was the routing handover design. The theoretical routing handover undoubtedly can improve the PeerHood performance even though the unpredictable behaviour of mobile devices, the big connection delay and the monitoring limitation make us doubt of its usefulness in a real environment.

So far there exists the possibility to lose data due to Write function not being aware of the connection loss. Additionally, the implementation of Data Transferring Acknowledge is too costly due to the small size of packet. Thus an efficient Data Buffering is necessary to guarantee the data integrity. About the new parameters of Plugin, link quality parameter was used as a supposition value but in any moment we have doubted the necessity of this value inside PeerHood protocol. The link quality as an indispensable parameter has to be studied in more detail.

Regarding high processing task migration, in the actual telecom market PeerHood has hard competences as cellular network 3G and HSDPA due to the excellent coverage and reasonable data transmission cost. In fact there exist already some servers offering image processing and text translation services through MMS *http://www.tauyou.com/* and direct video analysis through 3G *http://www.t-immersion.com/*. However, the possibility to interoperate between the existing network technologies and incorporation of any others give PeerHood the unique capacity to design a totally flexible network combining different technologies.

6.2 Potential Applications

Coverage Amplification

One of the most interesting features of PeerHood would be the amplification of coverage in areas where normally devices are not able to receive the signal. In figure 6.2 we have a tunnel where mobile phones have not any GPRS signal. One server is in the outside of the tunnel and provided with GPRS antenna. Inside the tunnel we proceed to install several Bluetooth devices making function of connection bridges. Once the mobile phone wants to access to the mobile services it will use a PeerHood application to connect to the server and access to the whole GPRS network.

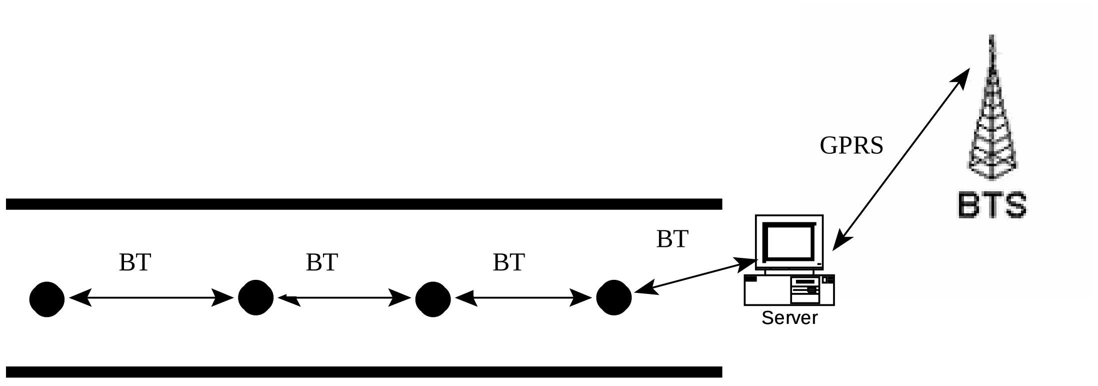


Figure 6.1 Coverage Amplification

The coverage amplification concept is also applicable to unify small LANs to a bigger network. In high mobile devices density places like office building or university, the concentrated distribution of mobile devices means a wide PeerHood network with many dynamic nodes. Many applications can be used in the environment as free Bluetooth calls as social networking, etc.

REFERENCES

[1] Jari Porras, Petri Hiirsalmi and Ari Valtaoja: "Peer-to-peer Communication Approach for a Mobile Environment", 37th Annual Hawaii International Conference on System Sciences, 2004.

[2] Arto Hämäläinen and Jari Porras: “Enhancing Mobile Peer-to-Peer Environment with Neighborhood Information”, Proceedings of the 3rd Workshop on Applications of Wireless Communications, 2005.

[3] R. Gold and C. Mascolo, Use of Context-Awareness in Mobile Peer-to-Peer Networks. Proc. 8 tn IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 01), Bologna, Italy, 2001.

[4] Sidath B Handuruande Samrat Ganguly Sudeept Bhatnagar: “Fast Bluetooth Discovery for Mobile Peer-to-Peer Applications”, Fourth International Conference on Mobile Systems, Applications and Services, Uppsala, Sweden, June 2006.

[5] Tommi Kallonen and Jari Porras, "Use of distributed resources in mobile environment", the 14th IEEE International Conference on Software, Telecommunications and Computer Networks (Softcom 2006), Sept 29 - Oct 1, Split, 2006, ISBN 953-6114-87-9.

[6] Savvas Gitzenis, Nicholas Bambos, "Mobile to Base Task Migration in Wireless Computing," Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), 2004

[7] Oriana Riva, Tamer Nadeem, Cristian Borcea, and Liviu Iftode: “Mobile Services: Context-Aware Service Migration in Ad Hoc Networks” IEEE Transactions on Mobile Computing, Volume 6, Issue 12, Dec. 2007 Page(s):1313 – 1328, Digital Object Identifier 10.1109/TMC.2007.1053

[8] Mads Darø Kristensen, “Enabling Cyber Foraging for Mobile Devices”, the fifth Middleware for Network Eccentric and Mobile Applications Workshop, Magdeburg, Germany, September 2007.

[9] Yu-Tang, Guo Wan-Li Lv, Bin Luo. An improved Resource Discovery Algorithm for Gnutella Networks. Third International Conference on Natural Computation 2007 IEEE

[10] Khaled Nagi, Iman Elghandour and Birgitta Kônig-Ries. “Mobile Agents for Locating Documents in Ad-hoc Networks”. Seventh International Workshop on Agents and Peer-to-Peer Computing (AP2PC08) Estoril, Portugal May 13, 2008

[11] Yasser K.Ali, Hesham N.Elmahdy, Sanaa El Olla Hanfy Ahmed. “Optimizing Mobile Agents Migration Based on Decision Tree Learning”. Proceedings of World Academy of Science, Engineering and Technology (PWASET) Volume 22 July 2007 ISSN 1307-6884

[12] Nikos Migas, William J.Buchanan and Kevin A. Mcartney. “Mobile agents for routing, topology discovery, and automatic network reconfiguration in ad-hoc networks”**.** Engineering of Computer-Based Systems, 2003, Proceeding. 10th IEEE International Conference and Workshop, April.

[13] Romit RoyChoudhury, S.Bandyopadhyay, Krishna Paul. “A Distributed Mechanism for Topology Discovery in Ad Hoc Wireless Networks Using Mobile Agents”. Mobile and Ad Hoc Networking and Computing First Annual Workshop, 2000.

[14] Sanket Nesargi, Ravi Prakash. “MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network”. INFOCOM 2002 Twenty-First Annual joint Conference of the IEEE Computer and Communications Societies. 23-27 June 2002

[15] H. Qi, F. Wang, "Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks," The 13th International Conference on Wireless Communications, vol. 1, pp.147-153. Calgary, Canada, July, 2001

[16] Fang Zhiyuan, Chen Xiaoyun, Tang Yong, Zhang Jingchun, Zhou Yu**. “**Real-Time State Management in Mobile Peer-to-Peer File-Sharing Services”. Service-Oriented Computing and Applications International Conference, 2007. SOCA '07. IEEE.

[17] Gaogang Xe, Zhenyu Li, Jianing Chen, Yifen Wei, Issarny, V, Conte, A. DTCS: “A Dynamic Tree-based Consistency Scheme of Cooperative Caching in Mobile Ad Hoc Networks”.Wireless and Mobile Computing, Networking and Communications, 2007. WiMOB 2007. Third IEEE International Conference.

[18] Jiannong Cao, Yang Zhang, Li Xie, Guohong Cao**. “**Consistency of cooperative caching in mobile peer-to-peer systems over MANET”**.** Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference

[19] http:://en.wikipedia.org/wiki/Gnutella. Wikipedia

[20] http:://en.wikipedia.org/wiki/Handoff Wikipedia

[21] David R.Musser & Atul Saini. “STL Tutotial and Reference Guide: C++ Programming with the Standard Template Library”. ISBN 0-201-63398-1

[22] W.Rchard Stevens, Bill Fenner & Andrew M.Rudoff. “Unix Network Programming: The Sockets Networking API”. Third Edition ISBN 0-13-141155-1