MPSockets: Addressing Mobility in End-to-end Protocols

by

Kun Niu

A dissertation submitted to the University of Dublin,

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

September 2006

DECLARATION

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Kun Niu

September 11, 2006

PERMISSION TO LEND AND/OR COPY

ACKNOWLEDGEMENTS

Many thanks to my supervisor, Stephen Barrett, for his guidance and advice throughout the course of this project. Many thanks to Stefan Weber, who was ever willing to help with any problems, however trivial.

Many thanks to the entire ubiquitous computing class for a fantastic and unforgettable year in Ireland.

Special thanks to my family. To my parents and my boyfriend for their perpetual, and unconditional support. And lastly I thank myself, without whom none of this would have been possible…

Kun Niu

University of Dublin, Trinity College

September 2006

ABSTRACT

Computer networks have in the past been based on fixed infrastructures where network nodes are interconnected by immobile elements such as switches, bridges and routers. The traditional end-to-end communication is based on the assumptions that the availability of both endpoints at the time of connection and keeping a stable connection during data transferring.

However, the exponential increase in the number of mobile network nodes, such as laptops, PDAs and mobile phones, is changing the traditional architecture of networks. Some frequent disconnection issues caused by the mobility of nodes or continuous changes of topology break the requirements of traditional communication model, and are becoming problems faced by traditional end-to-end protocols, such as TCP in mobile networks.

To solve these problems, we developed a new model by introducing the concept of "Meeting Places" (MPs), which are a few known and relatively stable nodes in the network. We made the assumption that a fixed MP responsible for a given area. Once a node moves to this area, it could connect to the MP. By inserting some MP nodes between two endpoints, the end-to-end communication is split into several segments, which enhance the stability and reliability of data transmission due to shortening of path length and reducing the cost of persistent network connection. It is also scaleable through introducing more MP nodes. Moreover, an MP node not only works as a proxy to receive and forward package, but may also have some intelligence as well, such as communication with other MP node.

We started our work by describing some scenarios in the mobile networks, such as communication between both or either mobile endpoints, then through analyzing some requirements for the solution , the architecture of MPSockets and details of its design are presented.

We made our evaluation from three aspects: by comparing with traditional end-to-end communication, such as Java Sockets; by analyzing some application area it could be used; by comparing with some existing application layer and transport layer solutions, such as Mobile IP, Split-TCP, Indirect-TCP.

This dissertation also presents some future work by analyzing some assumptions we have made in the design and implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

The overall goal of this thesis is to develop a new model which addresses the problems faced by traditional end-to-end protocols such as TCP in mobile networks. The first chapter introduces the issues encountered by end-to-end communication in mobile networks and outlines the project objectives and gives an overview of my design. Finally a roadmap of the rest of this document is presented.

1.1 Background

In this section, the characteristics of mobile networks will be discussed first, which was followed by major issues mobile networks encountered. Finally, some requirements for traditional end-to-end communication will be described.

1.1.1 Characteristics of Mobile Networks

The development of mobile communications technology created a new model for the interconnection of electronic devices. There is no need for the communication devices to be attached to a physical location, or to be stationary. Some devices such as PDAs and laptops, they can move to anywhere at any time they want.

Mobility becomes the most prominent feature of mobile networks due to more and more mobile nodes existing. Mobility introduces some problems: mobile nodes' network address changes accordingly, highly dynamic variable network quality of service. Moreover, the continuous change in topology[1].

Disconnection issues[2] become common in mobile networks which may be caused by mobility of nodes or network quality of services. However, some application area often depend heavily on network and probably cease to function during network failures. For example, some exceptions will occur for an accessing node once the accessed node disconnected with the network.

1.1.2 Major Issues Mobile Networks Encountered

In this section, we'd like to analyze the major issues faced by mobile networks from the view of layered protocol stack[3].

- **Inefficiency of network layer protocol:**

Internet Protocol (IP) is original designed for wired networks with fixed nodes. Each node is assigned an IP address which consists of network identifier and host identifier. For a node moves from one subnet to another, the most significant change is IP address. From this view, the traditional IP addressing schema isn't supportive of it. Though the node moved to another subnet and changed its address, the packets will still be delivered to the original address. Therefore, it will cause packets loss or quality of service degrading related issues.

- **Inefficiency of transport layer protocol:**

The transport layer is responsible for establishing the end-to-end connections, data packets delivery, flow control and congestion control as well. There exist simple, unreliable, and connection-less transport layer protocols such as UDP, and reliable, byte-stream-based, and connection-oriented transport layer protocols such as TCP for wired networks. However, these approaches are not suitable for mobile networks due to their dependence on the availability of both nodes and on the stable network connection between them during the period of communication. Once the communication couldn't meet the requirements, some exception will occur. And the congestion control algorithm which is used for handling the congestion in fixed networks is not suitable in a mobile network. It will cause degraded performance due to misunderstand the real cause which lead to the packets loss in mobile network. Every packet loss could be assumed as congestion and the congestion window would be reduced. Therefore, it will lead to a very low throughout in the mobile networks if some disconnection occurred frequently.

- **Inefficiency of application layer protocol:**

HTTP worked as application layer protocols in the wired network, which is also not quite suitable for mobile networks. The major issues are its stateless operation, high overhead due to character encoding, redundant information carried in the HTTP requests, and opening of a new TCP connection with every transaction. It's relatively expensive application protocol.

1.1.3 Traditional end-to-end Communication

The communication between two nodes in wired networks is often based on end-to-end protocols to implement communication models such as Client/Server.

Some requirements are need to be satisfied for the traditional end-to-end communication:

- Availability of both nodes at the time of connection;
- Stable network connection between these two nodes during the data transferring

Take the traditional TCP for example:

TCP provides a connection-oriented, reliable, and byte stream services[4]. The conception of "connection-oriented" means that two end-points using TCP must establish a TCP connection with each other before they can transmit request and reply. There is a flow control mechanism for the receiver to limit the numbers of data could be transmitted by the sender. And due to limited length of datagram, it's necessary to split the data into multiple segments with assigned sequence numbers and reassemble them at the destination. The congestion control mechanism is also used for guaranteeing the reliable transferring data and ensuring that the data won't be transmitted out of order.

It’s a relative complicated process for application developer. Fortunately, Sockets shield the programmer from low level details of networks, such as error detection, packet size, packet retransmission, and more.

A socket is a connection between two nodes in the networks. The traditional Java Sockets working mechanism could be presented as follows[5]:

For the Client:

- Create a new Socket connection with a remoter node by using a constructor.
- Once the connection is established, the local and remote nodes could get input and output stream from the socket and use those streams to send and receive data from remote node.
- When the data delivery is finished, close the socket connection.

For the Server:

- Create a new ServerSocket on a particular port using a constructor.
- The ServerSocket listens for incoming connection attempts on that port by using accept method, which blocks until a client attempt to make a connection, then returns a Socket object connecting the client and server.
- Once the connection is established, could get input and output stream from the socket and use those streams to send and receive data from client.
- The server, the client, or both close the socket connection.
- The server returns to listen for next incoming connection.

1.2 Design Objectives

In the sections above, we discussed the traditional end-to-end communication in wired networks, but it couldn’t perform well in a mobile network according to the characteristics we described in chapter 1.1.1 and some major issues existing in the mobile networks.

Therefore, our design objectives are to develop a new model which addresses the problems faced by traditional end-to-end protocols such as TCP in mobile networks. In the following chapter 2, some related works will be described from application layer to transport layer, I will make a comparison between our solution with them in the chapter 5---evaluation.

1.3 Design Overview

A concept of “Meeting Place” (MP) is introduced in our solution, and we made some assumption that there is always a fixed MP responsible for an area, once a node moves to this area, it could connect to this MP. Our Solution is to Split end-to-end communication into segments by inserting “Meeting Place” nodes (MP). The MP could work as a proxy to accept registry information from nodes or data from endpoint node then forward them to another endpoint node. In some situations, it also communicate with another MP node.

The key advantages of our solution are:

- A few known and relatively stable network nodes can be used to achieve end-to-end communication without the otherwise necessarily persistent quality of network.
- The approach is scaleable through the introduction of more Meeting Place nodes.

1.4 Dissertation Roadmap

This dissertation is composed by seven chapters and provides a linear approach to the design, implementation and evaluation of our design.

Chapter 1 discusses the characteristics of mobile networks and requirements for the traditional end-to-end communication in wired networks, after that we described some major issues existing in the mobile networks from the view of layered protocol stack. Based on these problem descriptions, we provide our design objectives and have a overview of the solution.

Chapter 2 will discuss some existing solutions which used to solve the mobility problems in the mobile networks. We divided these solutions into application layer and transport layer solution, and a global comparison of transport layer solution is also made.

Chapter 3 describes the design details which begin from an overview of the design, Through the scenarios description, we got the requirements for the design. Then we present the architecture of our design, which consists of configuration file design, MPSockets library design and Meeting Place design, which are described in the following sections.

Chapter 4 begins from our implementation goals, then through two application description, we present the implementation of many detailed issues, such as user defined Class and data storage mechanism.

Chapter 5 investigates the results of our work by comparing with traditional Java Sockets communication and including a discussion of several application area that we expect. We will conclude by comparing with some solutions in the related work.

Chapter 6 provides a final overview of the work. We will include some suggestions for possible future work in order to expand our solution and include new features.

Chapter 2: Related Work

In this chapter, we will directly discuss prior work in related areas of solving the mobility problems in mobile networks. We will categorize and individually review the most important network layer and transport layer solutions for mobile networks. We will conclude the chapter with a global feature comparison table.

2.1 Network Layer Solution—Mobile IP

In the traditional wired networks, each computer will be assigned a global unique IP address when it join the network, which is used to identify this particular computer. All data packet will be routed to this computer according to its IP address. This scheme works very well in the wired networks but once it introduced to the mobile networks, problems encountered, because the IP assigned to the mobile node can not be restricted to a region any more.

The first conceivable solutions for above problem will be assign a new IP address to the mobile node when it moves from one sub-network to another one. This naturally solution has the advantage of keeping the consistent with the subnet that the node currently in. But this solution has a fatal weakness. Because all connections from the host computer to another terminal are based on the TCP, therefore, if the IP address changed, all TCP connections will have to be re-established again, that means all current sessions will be terminated. Obviously, it is not a very desirable solution. Another solution would be to keep the IP address unchanged and use some special routing scheme to track down the current location of the mobile node. This solution would be more practical if the number of mobile nodes is small. These solutions are inadequate due to theirs natural weakness, but they can provide certain guidelines for the actual solution – mobile IP.

The essence of the Mobile IP[6] scheme is to keep the old IP address and add a few mechanisms to provide the mobility support. I will briefly introduce how does this scheme work by using a illustration. Before we start looking into the details of this scheme, one thing we should make it clear: what is the essential characteristics for a good solution? Three requirements should be satisfied. First, the compatibility. The solution should be totally based on the already existing wired Internet infrastructure, because it is well-established and economical. Any idea that try to alter the way it is working is not practical. Second, the scalability: the solution should be able to support large number of users, otherwise it is commercially impractical. Third, transparency: the mobility provided by the solution should be completely transparent to the user. The user shouldn't be able to sense the different when working in a wireless domain or in a wired one. Now, lets have a close look the way that the mobile IP solution works[3].

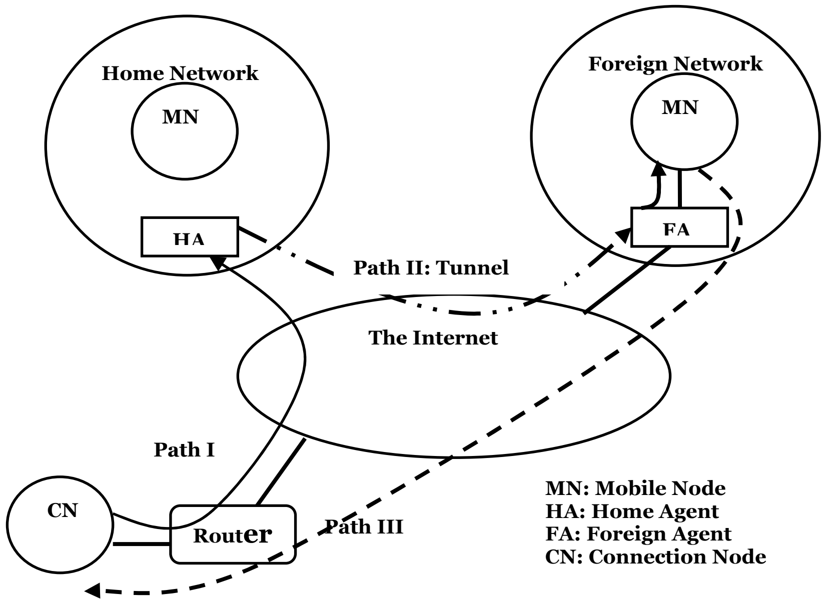


Figure 2-1: Working Mechanism: Mobile IP

In above figure, mobile node (MN) is a mobile terminal or a mobile router. It is the host where the mobility support is provided. The correspondence node (CN) is fixed node or a mobile node which would like to communicate with the MN. The HA in this figure refers to the home agent, which is the "base station node" of the subnet that the MN currently staying in before moving into another domain. Same as the HA, the FA refers to the foreign agent, which is the “base station node” of the subnet that the MN will be staying after the moving. The “based station node” normally is the access point (AP) in the wireless network.

The basic idea about mobile IP scheme is to keep the old IP address unchanged and use a few additional mechanisms to provide mobility support. Actually, in this scheme, a new IP address, the care of address (COA) indeed assigned to the MN, but it is totally transparent to the user, the user doesn’t feel any change at all. There are two types of COA, Foreign agent-based COA and Co-located COA. The first type COA actually is the address of the FA node which the MN currently connected to. By using this COA, the MN can be located through the FA. The second one is a topologically correct IP address. In this case, every MN has two different IP address. When a CN wants to communicate with the MN, it will send the data to the old address, the HA receives the data and forward it to the MN by using its new IP address, this is called tunnelling. The Co-located COA is more widely used in the industry.

Take the above figure as an example, the MN has been assigned a Co-located COA. The CN send data through the Path I, the HA received the data and tunnel it to the MN by path II . In the Co-located COA scenario, the MN will periodically inform HA about its current COA by sending a registration packet directly to the HA. This process is called registration. When the MN wants to send back a reply, there is another problem, because some routers will filter away the packet going out of the network if the source IP address associated with the packets is not the subnet’s IP address. This is called ingress filtering problem. In the mobile IP scheme, this problem get solved by using the idea of reverse tunnelling. The MN will encapsulates its packet using the source network IP address, the COA that it has been assigned, then send the packet to HA, then the HA will de-capsulate the packet and forward it to the CN. This method is not efficient, but it does work in practise.

2.2 Transport Layer Solutions

After spent some time on discussing about the network layer solution – mobile IP, and had a general idea how does it work, now it is the time to take a look at the transport layer solutions[3].

As we all know the mobile network not only has the mobility problem in the network layer, but also has the high transmission error rate and low bandwidth problem, they are the problems exist in the transport layer. Obviously it the job of the transport layer protocol to perform the error recovery and flow control.

The traditional TCP, a connection-oriented transport layer protocol which guarantees the in-order and reliable service for the data transmission, is the classical wired networks transmission protocol. It was originally designed for the wired network, therefore, it is not quite suitable for using at the wireless domain. For example, It assumed that any packet loss caused by congestion, then it will reduce the congestion window. Therefore, the frequent nodes moving and disconnection would lead to a very low throughout in the mobile networks.

Some solutions are proposed to alter the traditional TCP protocol to suit the mobile networks. Since the TCP has dominated the network layer for decades, any idea trying to develop a brand new protocol to replace the TCP would be impractical, hence, the modification of the TCP is the only correct approach of solving this problem.

Since there are numbers of different solution in the transport layer, before we start discussing about each one of them, I would like to create a classification tree to show the relationship between them and the TCP over ad hoc networks. The top-level classification divides the transport layer protocols into two divisions: the TCP over ad hoc networks and the non-TCP base other protocols. Then the TCP over ad hoc can be further classified into Split approach and End-to-End approach depending the solution approaches. Further more it will be each individual solution protocols.

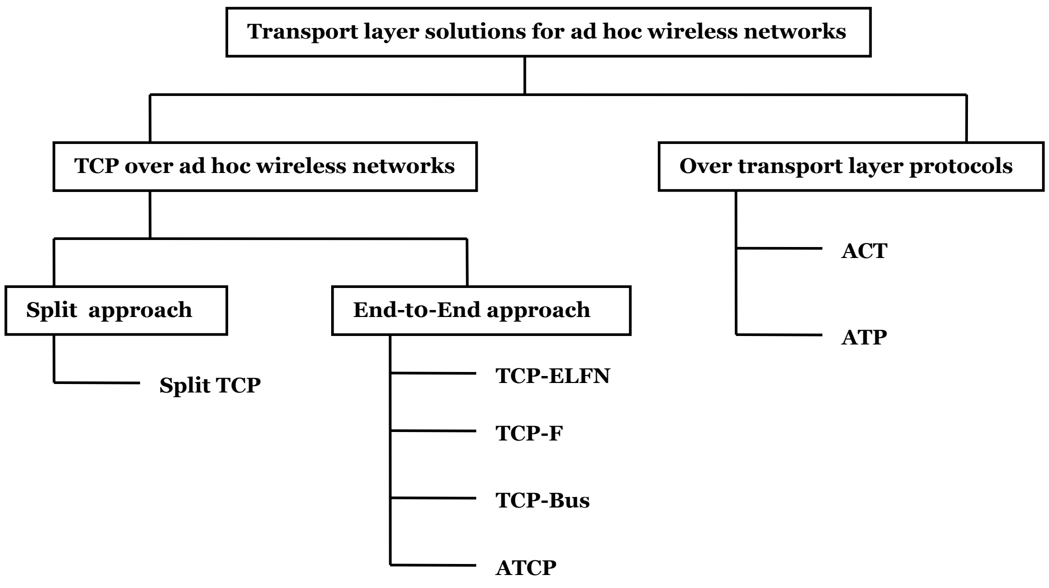

Transport layer solutions for ad hoc wireless networks
TCP over ad hoc wireless networks
Over transport layer protocols
Split approach
End-to-End approach
Split TCP
TCP-ELFN
TCP-F
TCP-Bus
ATCP
ACT
ATP

Figure2-2 :Solutions Classification Tree

Now it's time to have a closer look into every individual solutions. There are different solutions has been developed by far, we will discuss them one by one in the following paragraph.

2.2.1 Indirect-TCP

In order to make the solution clearly understandable, a illustration will be used during the discussion of Indirect-TCP[7].

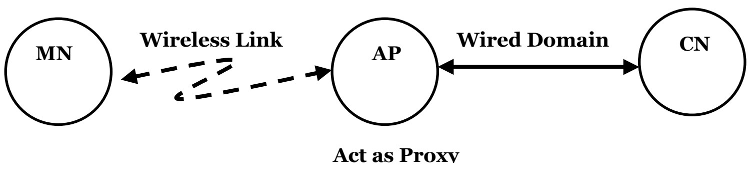


Figure 2-3: Indirect-TCP

In this solution, the TCP connection will be split into two distinct connections, one TCP connection between the MN and the AP and another one between AP and CN. The division point is the access point (AP) which is the boundary of the wireless domain and the wired domain and acting like the proxy for MN. The traditional TCP can be used in the wired domain as normal, and some modified version the TCP will be used in the wireless domain. The basic idea of the indirect TCP is “store and forward”. We all know that the wireless connection is quite unreliable and has high error rate, this will cause a high retransmission rate which will dramatically reduce the performance of the network. In this solution the AP is acting like a proxy for the MN, all packet sent from the CN to the MN will be buffered in the AP for a short moment until it has been received by the MN correctly, if any went wrong during the wireless domain, which is highly possible, the AP will take the packet out from the buffer and retransmit again. In this way the retransmission is performed by the AP, not the CN. The AP will acknowledge to the CN for the packet transmission, the MN will acknowledge the data received to the AP only. This strategy might take a longer time when performing the handoff process, because acknowledgments could be buffered couple of times during the transmission.

2.2.2 Snoop-TCP

The basic idea of the snoop TCP[8] is to buffer the data as close to MN as possible. Because the shorter the transmission path is, the fewer time it takes. In this solution, the base station (BS) just snoop the data packets transmitted in both directions and recognizes the acknowledgments. The BS will buffer the packets transmitted, but it does not acknowledge instead of MN. If one packet has been transmitted successfully, it simply remove this packet from its buffer. If anything went wrong, such as duplicate acknowledgement or no acknowledgement at all after snooping for a quite moment, then it will retransmit the packet from the buffer and discard the duplicate acknowledgement. This will avoid unnecessary retransmission from the CN. On the reverse path, when the packet is sent from the MN to the CN, if the BS detects a packet sequence error according to the acknowledgement sent by the CN, it will send a negative acknowledgement to the MN to indicate that there is a data loss over the wireless domain.

2.2.3 Mobile-TCP

In the wireless network domain, the most common problem that happens quite often is the connection between the MN and the AP could be lost for a short intervals. This typically happens when the MN moves behind or enters a huge building which could filter away the signals. In such case the CN will keep transmitting but eventually times out will occur. If in case of the indirect TCP solution, the data buffered in the AP could grow too large. This could also lead to a data lost. The way to solve this problem is simple, the CN must be informed in such case, therefore, the mobile TCP is designed to handle this situation.

In Mobile-TCP[9], a node in the wired part of the network is established to controls a number of APs. This node is called the supervisory host. When the situation described above happened, this node will inform the CN and prevent the CN from sending more data packet to the MN. The connection could be resumed if the MN can be contacted again after a short interval. When the supervisory host receives a TCP packet, it will forward it to the MN. After it has received a acknowledgement from the MN, it forwards the ACK packet to the CN. Therefore, in case of any thing went wrong on the MN side, such as link error, the supervisory host will stop forwarding acknowledgement to the CN. By using this strategy to avoid the buffer leaking problem.

2.2.4 Split-TCP

The performance of the TCP over ad hoc network could be affected by number of issues, the major one among of them is the increasing path length could degrade the throughput of the network. Because the short connections normally obtain much higher throughput compare to the long connections. This problem can also lead to a unfairness among TCP sessions, as certain session could occupy the transmission channel for long duration which will dramatically reduce the flow of other sessions. Hence, this problem can lead to low overall throughput of the system.

Split-TCP[10] was developed to solve this problem. The basic idea of this solution is to split the objectives of the transport layer into congestion control and end-to-end reliability and it also split a long TCP connections into several of short connections (Segments or Zone) by selecting number of intermediate nodes (Proxy Node). Lets take a close look at how does this solution work by using a illustration

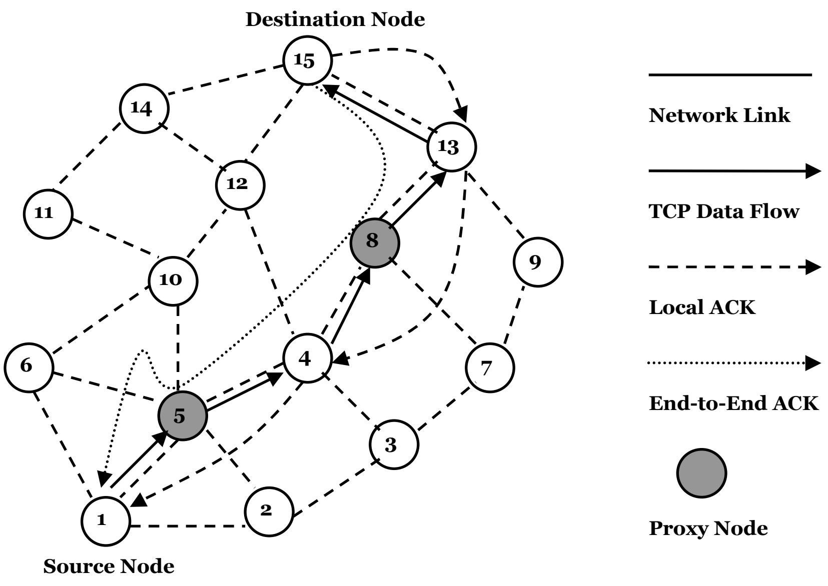


Figure 2-4: Split-TCP

In the above illustration, the entire TCP connection is divided into three zones by 2 intermediate node (Node 4 and 13). The source node is 1 and the destination node is 15. A proxy node receives a TCP packet, it exams the contents of the packet, store it in its local buffer, and sends an acknowledgement (Local ACK) back to the source node, it could be the connection initiator node or just a previous proxy node. Then this proxy node forward this packet to the next proxy node. After the next proxy node received packet, it repeat the same process again, buffer the packet, send back acknowledgement, and forward. Every proxy node clears those buffered packet only after receiving an acknowledgement. This procedure will be repeat again and again until the packet reaches its destination node. The source node clears the buffered packets after receiving the acknowledgement (End-to-end ACK) from the destination node.

2.2.5 A global feature comparison

Now, Lets have at look the different features of different solutions. In order to make the features comparison clear and understandable, I will create a table to show the different features of every solution. The comparison will be made from nine different aspects, they are: Packets switching paths; if there is a retransmitting node participate; if there is a single failure node; handoff latency; Security; End-to-End Semantics; if there is a intermediate node; how is the slow star feature; if the packets buffered at the access point[3].

Feature	Indirect TCP	Snoop TCP	Mobile TCP	Split TCP
Changes in:				
AP	**Yes**	**Yes**	**Yes**	**Yes**
CN	**No**	**No**	**No**	**No**
MN	**Yes**	**Yes**	**Yes**	**Yes**
Retransmitting Node	**AP**	**AP**	**Not Applicable**	**Proxy Node**
Single Point Failure	**Yes (AP)**	**No**	**No**	**Yes (PN)**
Handoff Latency	**Low**	**Low**	**Low**	**High**
Security	**Breach at AP**	**Breach at AP**	**Not Applicable**	**Breach at PN**
End-to-End Semantics	**No**	**Yes**	**Yes**	**No**
Retransmissions by Intermediate Nodes	**Yes**	**Yes**	**No**	**Yes**
Slow Start	**Not Applicable**	**Yes**	**No**	**Not Applicable**
Buffer at AP	**Yes**	**Yes**	**No**	**Yes**

Table 2-1: Solution Features Comparison

Chapter 3: Design

This chapter begins with an overview of the design, and then describes some communication scenarios in mobile networks by using our approach. Through analyzing these scenarios, some requirements for the design will also be described. Then more emphasis will be put on descriptions of the architectures of the design.

3.1 Overview

From the scenarios we described in Chapter2, point-to-point communication is still playing an important role in the mobile networks. However, some issues arise due to mobile network's characteristics, such as the communications between endpoints seems not efficient due to temporary and unannounced loss of network connectivity

To solve such problems in our design, some fixed nodes called "Meeting Place" are inserted in the mobile networks. Some assumptions are made in our "Meeting Place" model, such as once a node (Server/Client) moves to a network area, there is always a fixed MP node ("Meeting Place" node) connected to it in this area. Thus, the point-to-point communication has been split into several segments due to MP nodes are involved. The MP node could work as a proxy to accept registry information from nodes or data from endpoint node then forward them to another endpoint node. In some special situations, it also communicate with another MP node.

Some advantages of our model:

- First, our solution is at the transport level, it breaks the traditional assumption in the point-to-point communication that two endpoints must be connected at the same time. Thus, some higher application in the mobile networks could use it directly without any change.

- Second, our solution uses a few known relatively stable network nodes working as "Meeting Place" to achieve end-to-end communication. For the further study, they could work not only as a proxy to store/forward data, but also could have some filter functions to assure network security.

- Third, our solution splits point-to-point communication into several segments to some extent which will reduce maintenance costs of necessarily persistent quality of networks.

- Fourth, the approach is scaleable through the introduction of more Meeting Place nodes and can accommodate a range of intermediary patterns to achieve various tradeoffs in performance.

3.2 Scenarios with "Meeting Place" model

In this section, we will describe some scenarios using "Meeting Place" model. These scenarios are presented according to the mobility of nodes.

3.2.1 Mobile Client and Fixed Server

In this scenario, the client is mobile while server is fixed. The client sent the request to the server, but when the reply was sent back from server, the client has gone. However, once the client connect to the network, it will get the reply.

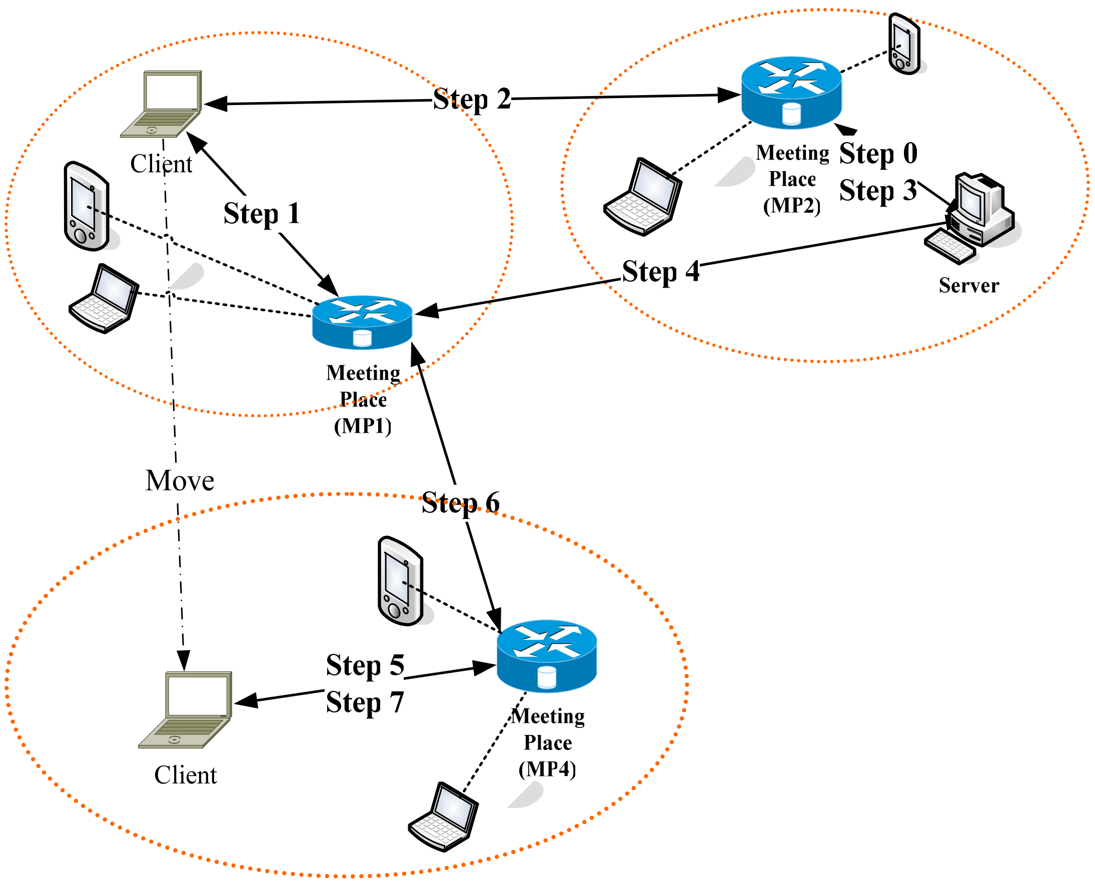


Figure 3-1: Scenario: Mobile Client and Fixed Server

The dialogue between client, server and MP (meeting place) assumes the following form:

Step 0:

0.1 *Server* read some configuration files to find out *MP2* is meeting place in this area.

0.2 *Server* to *MP2*: "Please let me register in and give me a global id".

0.3 *MP2* to *Server*: "Here is your global id ".

Step 1:

1.1 *Client* read some configuration files to find out *MP1* is meeting place in this area.

1.2 *Client* to *MP1*: "Please let me register in and give me a global id".

1.3 *MP1* to *Client*: "Here is your global id".

Step 2:

2.1 *Client* read some configuration files to find out which MP *Server* belongs to . In this scenario, *Server* belongs to *MP2* at this moment.

2.2 *Client* sent request to *MP2*. "Please send my request to *Server*".

Then *Client* moves to another place.

Step 3:

3.1 *MP2*: "I'd better store the request from *Client* until Server/new MP come to collect it".

3.2 *Server* to *MP2*: "I'm coming , is there any request to me?"

3.3 *MP2* to *Server*: "There is a request to you which came from *Client*".

Step 4:

4.1 *Server* processed the request and prepared to send a reply to *Client*.

4.2 *Server* read some configuration files to find out which MP *Client* belongs to. In this scenario, *Client* belongs to *MP1* in the configuration file.

4.3 *Server* to *MP1*: "There is a reply to Client".

4.4 *MP1*: "I'd better store the reply from Server until Client/new MP come to collect it".

Step 5:

5.1 *Client* to *MP4*: "I moved to your area please let me register here".

"Please ask my old MP to forward any reply to me if it has".

5.2 *MP4*: "You can register here, but you still use your original global id".

"I'll ask your old MP to forward your reply to me".

Step 6:

6.1 *MP4* to *MP1*: "Could you please forward the reply to me, I'm the new MP of the Client ?"

6.2 *MP1* to *MP4*: "Ok, here you are".

Step 7: *MP4* to *Client*: "Here is the reply to you."

3.2.2 Fixed Client and Mobile Server

In this scenario, the server is mobile while client is fixed. The client sent a request to the known server, but he didn't know the server is not available at this moment or not come back forever. By using my approach, the request could be delivered to the server and the client could get the reply from the server.

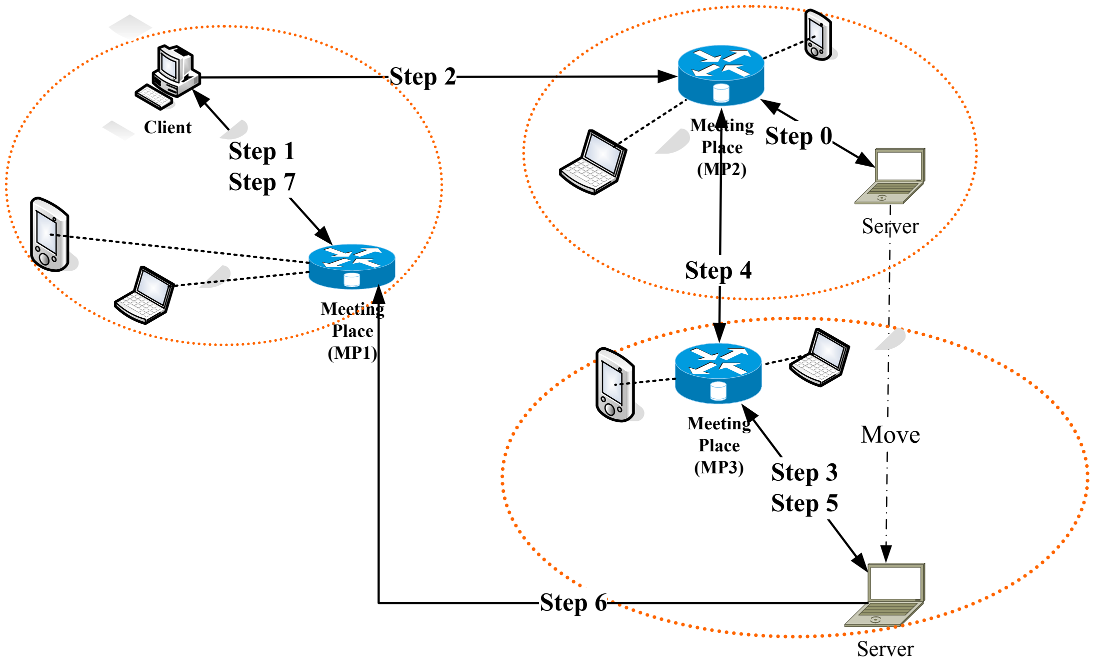


Figure 3-2*: Scenario: Fixed Client and Mobile Server*

The dialogue between client, server and MP (meeting place) assumes the following form:

Step 0:

0.1 *Server* read some configuration files to find out *MP2* is meeting place in this area.

0.2 *Server* to *MP2*: “Please let me register in and give me a global id”.

0.3 *MP2* to *Server*: “Here is your global id ”.

After a while, *Server* moves to another place.

Step 1:

1.1 *Client* read some configuration files to find out *MP1* is meeting place in this area.

1.2 *Client* to *MP1*: “Please let me register in and give me a global id”.

1.3 *MP1* to *Client*: “Here is your global id”.

Step 2:

2.1 *Client* read some configuration files to find out which MP *Server* belongs to . In this scenario, *Server* belongs to *MP2* in the configuration file.

2.2 *Client* sent request to *MP2*. “Please send my request to *Server*”.

2.3 *MP2*: “I’d better store the request from *Client* until Server/new MP come to collect it”.

Step 3:

3.1 *Server* to *MP3*: “I moved to your area please let me register here”.

“Please ask my old MP to forward any request to me if it has”.

3.2 *MP3*: “You can register here, but you still use your original global id”.

“I’ll ask your old MP to forward your request to me”.

Step 4:

4.1 *MP3* to *MP2*: “Could you please forward the reply to me, I’m the new MP of the Server ?”

4.2 *MP4* to *MP3*: “Ok, here you are”.

Step 5:

5.1 *MP3* to *Server*: “This is the request for you which came from *client*”.

5.2 *Server* processed the request and prepared to send a reply to *Client*.

5.3 *Server* read some configuration files to find out which MP *Client* belongs to. In this scenario, *Client* belongs to *MP1* in the configuration file.

Step 6:

6.1 *Server* to *MP1*: “There is the reply to Client”.

Step 7:

7.1 *MP1*: “I’d better store the reply to *Client* until *Client*/new MP comes to collect it”

7.2 *Client* to *MP1*: “I’m coming, is there any reply to me ? ”

7.3 *MP1* to *Client*: “Here is the reply to you.”

3.2.3 Mobile Client and Mobile Server

In this scenario, both client and server are mobile. Assume that client and server transmit instant messages when they are taking on a business travel. Both of them could disconnected at any time. By using my approach, mobile nodes wouldn’t miss any message once it connected to the network.

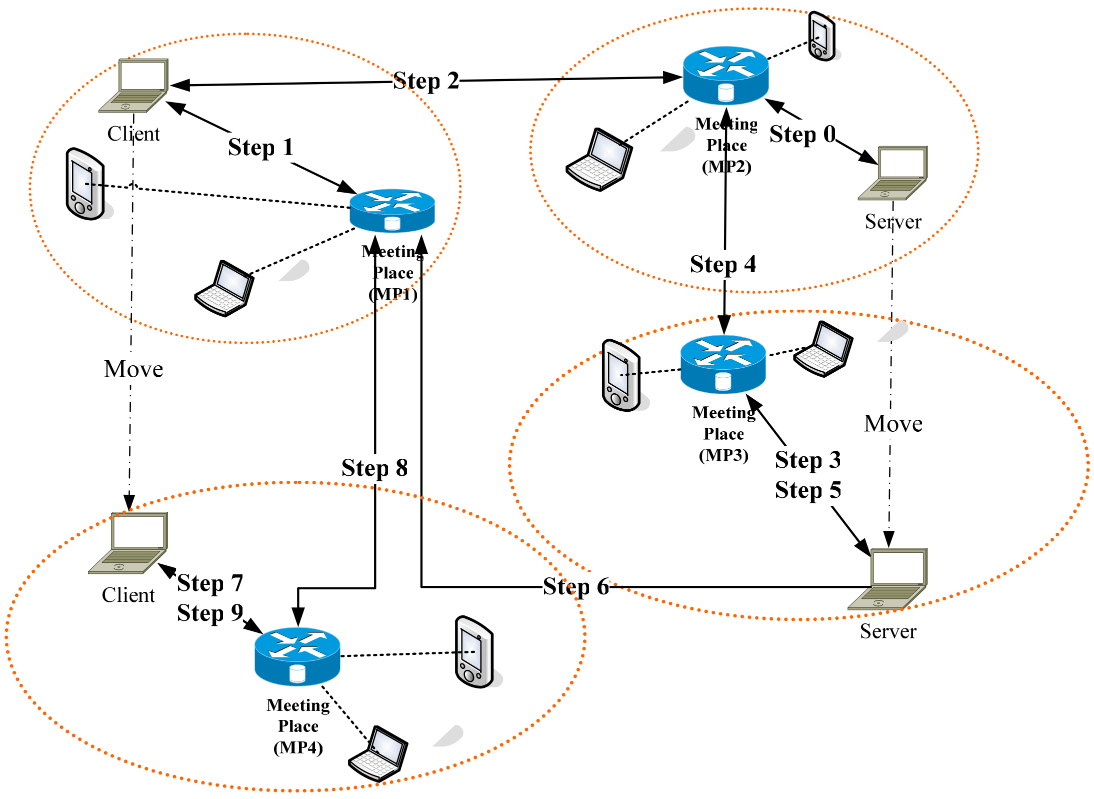


Figure 3-3*: Scenario: Mobile Client and Mobile Server*

The dialogue between client, server and MP (meeting place) assumes the following form:

Step 0:

0.4 *Server* read some configuration files to find out *MP2* is meeting place in this area.

0.5 *Server* to *MP2*: "Please let me register in and give me a global id".

0.6 *MP2* to *Server*: "Here is your global id ".

After a while, *Server* moves to another place.

Step 1:

1.4 *Client* read some configuration files to find out *MP1* is meeting place in this area.

1.5 *Client* to *MP1*: "Please let me register in and give me a global id".

1.6 *MP1* to *Client*: "Here is your global id".

Step 2:

2.4 *Client* read some configuration files to find out which MP *Server* belongs to . In this scenario, *Server* belongs to *MP2* in the configuration file.

2.5 *Client* sent request to *MP2*. "Please send my request to *Server*".

2.6 *MP2*: "I'd better store the request from *Client* until Server/new MP come to collect it".

Step 3:

3.3 Server to *MP3*: "I moved to your area please let me register here".

"Please ask my old MP to forward any request to me if it has".

3.4 MP3: "You can register here, but you still use your original global id".

"I'll ask your old MP to forward your request to me".

Step 4:

4.3 MP3 to *MP2*: "Could you please forward the reply to me, I'm the new MP of the Server ?"

4.4 MP4 to *MP3*: "Ok, here you are".

Step 5:

5.4 MP3 to *Server*: "This is the request for you which came from *client*".

5.5 Server processed the request and prepared to send a reply to *Client*.

5.6 Server read some configuration files to find out which MP *Client* belongs to. In this scenario, *Client* belongs to *MP1* in the configuration file.

Step 6:

6.2 *Server* to *MP1*: "There is the reply to Client".

6.3 *MP1* “I’d better store the reply to *Client* until *Client*/new MP comes to collect it”

Step 7:

7.1 *Client* to *MP4*: “I moved to your area please let me register here”.

“Please ask my old MP to forward any reply to me if it has”.

7.2 *MP4*: “You can register here, but you still use your original global id”.

“I’ll ask your old MP to forward your reply to me”.

Step 8:

8.1 *MP4* to *MP1*: “Could you please forward the reply to me, I’m the new MP of the Client ?”

8.2 *MP1* to *MP4*: “Ok, here you are”.

Step 9:

9.1 *MP4* to *Client*: “Here is the reply to you”.

3.3 Requirements for design

In this section, some requirements will be described through analyzing the scenarios above. And a complete scenario model will be put forward as followed.

3.3.1 Global Identifier for each nodes

The aim of our approach is to make the mobility be transparent to accessing node. For example, the client sent the data to a mobile server and it only knew the original information of the server (hostname or IP address) rather than checking the connection state of server frequently.

However, due to the mobility of node in the mobile networks, the IP address info changed accordingly. If we use the IP address to identify a unique node, it's not reasonable once node moves or connects to another networks. Therefore, a global identifier for each node is required.

Three popular distributed object paradigms are DCOM[11] (Distributed Component Object Model), CORBA[12] (OMG's Common Object Request Broker Architecture) and RMI (Remote Method Invocation) [13]. They use different ways to identify uniqueness for remote server object or for an interface.

- **DCOM**: In the DCOM IDL file, the association will be built between the interface and object class. Each interface and object class will be assigned a GUID (Gloablly Unique Identifier) which is a 128-bit number and is written in hexadecimal form. Commonly, it's written using a four-type word, 3 two-type words, and a six-byte word. For example : `{2C4C0DB3-2A45-13d2-12C5-30A02424D655}`

- **CORBA:** In order to invoke an object on a CORBA server, a client has to acquire a reference to the object. The references can be get through using COS naming service which is specified by OMG. So some stings are used as the identifier.

- **RMI:** it uses objID (Object Identifier) to uniquely identify remote server objects. Each identifier consists of an object number (a long) and an address space identifier (a UID) which is unique with respect to a host. When the identifier is used to uniquely identify an interface, it'll use the interface name and it can be mapped to a URL in the registry.

In our approach, the global identifier (GID) for each node consists of an String (4-bit) and a random number(6-bit) . The global identifier is generated by each Meeting Place it belongs to. The 4-bit String is the id for the Meeting Place. The random number is generated by the MP, which is also responsible for checking uniqueness in its registry record. In this way, we guaranteed that the GID is unique in the networks.

3.3.2 Identifier for Meeting Place

Due to Meeting Place acts as a proxy role, some communication between node and MP or MP and MP is necessary. Considering the scalability of the model, e.g. the numbers of MP could be added dynamically or node will move and change MP frequently. If we use a fixed IP address to identify a MP, it will prevent the model's expansibility and flexibility.

In view of known numbers of MP in our model, we use an identifier which consists of a 2-bit String (i.e. MP) and a 2-bit number. For example, MP01.... MP10. Therefore, if some nodes will connect to a MP, it doesn't need to know the detailed information about MP. And it's also convenient for using a new node to replace the old MP if it is broken.

3.3.3 Association Issues

Some association issues should be considered as follows:

- The association between MP's id and it's detailed information such as hostname, IP address and port.
- The association between nodes (client/server) and MP. A node needs to know which MP is responsible for the area it belongs to. And it also needs to know which MP is responsible for the area correspondent node belongs to.

For handling the association issues, there are some methods, such as lookup services, configuration files and some mapping mechanisms like hash table or map which can hold key/value pairs.

In our approach, some configuration files are used to implement the associations described above. Some lookup service is used as well for the association between hostname and IP address info.

3.3.4 Timeout Issues

TCP is a reliable end-to-end transport layer protocol for the wired networks. You can ensure your read/write operation won't block for more than a fixed number of milliseconds by setting timeout parameters. Commonly timeouts are given in milliseconds, after that, an exception will be thrown. However, this is not suitable for the mobile network. It's common for mobile nodes to disconnect from the network for more than several hours even several weeks. In the scenarios we described above, Client sends the request, then the MP of mobile Server will keep that and forward it to the Server until it's available. Therefore, it's obvious that milliseconds are not enough for mobile communication.

We should provide some methods to make the timeout much longer or avoid the Exception in the mobile networks. If the user set the timeout (e.g. 1 hour), we should thrown out an exception until the timeout expires.

3.3.5 Other issues

For this model, we have some assumptions on it:

- We assumed that the information about MP is known before. That is to say, we know which MP is responsible for an area. Once a mobile node moves to this area, the MP should be responsible for it. The node could know the MP by checking the configuration files.

- We assumed that each node has a MP to responsible for it in an area irrespective of the node is mobile or not. That is to say, there is always a fixed MP which could act as a reception to store the data coming from accessing node.

- We assumed that each node won't concern about correspondent node's status or detailed info about MP. For a fixed node, it works the same way as traditional point-to-point communication. If it moves, the only thing it needs to do is modify the old MP id to the new one.

3.4 Architecture of MPSockets

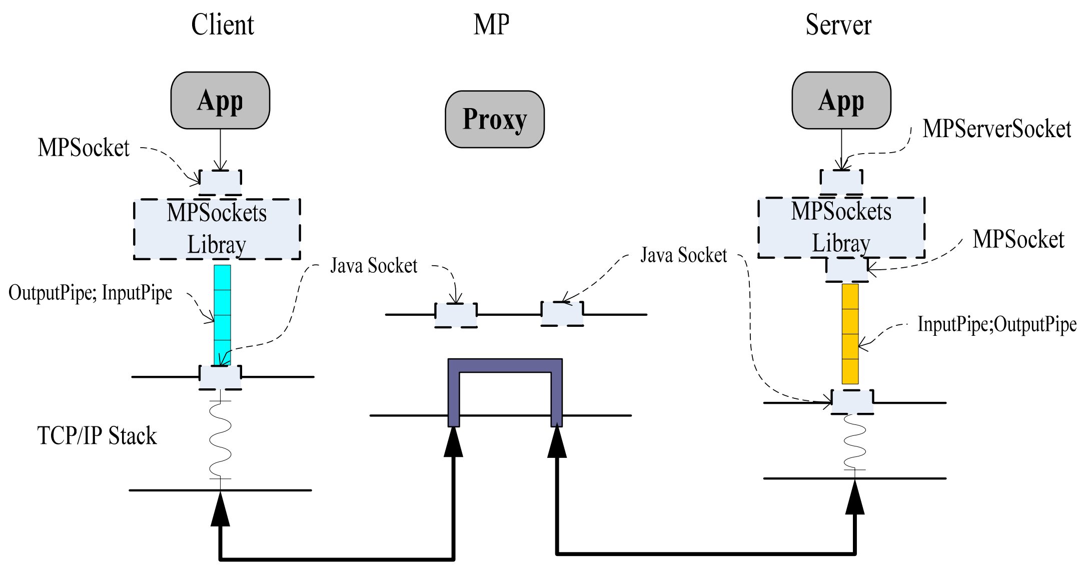


Figure 3-4: Architecture of MPSockets

As shown in the figure above, the MPSockets architecture consists of three parts: a user level proxy process running on fixed Meeting Place nodes; an in-kernel design on the Meeting Place node to provide proxy services; and a MPSockets library that runs under the application on the client/server.

Client/Server application program could call some Classes in the MPSockets to build socket connection, and use the output pipe and input pipe to transmit data. The communication between Client and MP, MP and Server, still using traditional Java Socket.

3.5 Design Configuration Files

According to the assumptions we made above, there are some configuration files are needed in our model:

- **Registry File:** Each end-point node (client/server) has a copy of registry info file. It has some associations between node info (e.g. hostname) and MP id. From checking this file, we can know which MP the end-point node belongs to at the first run.

- **MPinfo File**: Each end-point node and MP nodes have a copy of this file which describes the hostname and port information regarding to each MP id. By using this file, it's easier to maintain MP's information in our model. It also improves our model's scalability and flexibility. If more MP nodes are added, it's much easier to be done by modifying this file.

- **Nodeinfo File:** It's generated by MP nodes after the first end-point node register here. It makes a record of node's global id, hostname/port information and status info. The status will be modified according to data transmitting status, e.g. it's begin/on/complete transmission. By setting this status, it enhanced robustness of the model. Assume that another MP (e.g. MP11) have a talk with this MP(e.g. MP02), and ask MP02 to forward some data to MP11. By checking this status info, MP02 will process the forward request from MP11 using a different way, instead of some unexpected errors happened due to data has been transmitted already. If more end-point nodes registered in this MP, it will append their information in this file as well.

- **Info file:** It's generated and kept by an end-point node at its first run. It includes some information about the node's global id, hostname/port and local MP id. All of these information were got by registered in the local MP at its first run. Once this node moves to another place, the only thing it needs to do is change local MP id to a new one.

3.6 Design MPSockets library

The aim of our design is to solve the problems faced by traditional end-to-end protocols such as TCP in networks with mobile nodes. Moreover, our principle is to minimize the changes of these connection-oriented operations, e.g. establish and close a socket connection, send and receive data.

Therefore, to some extent MPSockets library could be thought as an extension of traditional Socket library. MPSocket Class and MPServerSocket Class are two essential Classes in this library. They replaced traditional Socket and ServerSocket's role from user's view.

In the following sections, we will put the emphasis on the description of the behaviours of these two essential Classes. In addition, due to inserting MP nodes in the mobile networks, point-to-point communication is split into two segments. To make the description much clearer, we'll use some sequence diagrams to present the communication between client and MP, MP to Server individually.

3.6.1 Communication between Client and MP

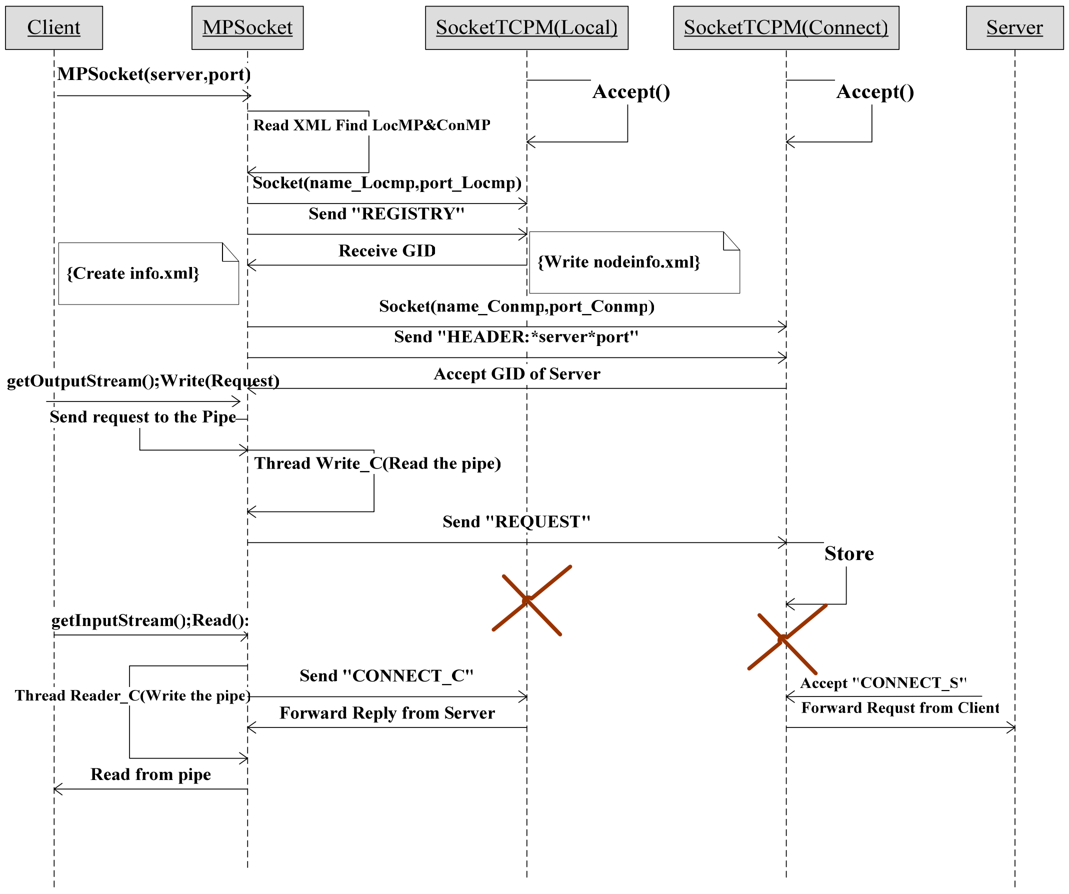


Figure 3-5: *Sequence From Client's View*

From the Client's view, it established connection to a remote server, then sent the request to the server and received the reply from the server.

3.6.1.1 Client Application Design

Client's application programs normally use client sockets in the following fashion:

1. Creates a new socket with a constructor, MPSocket(String host, int port). Then attempts to connect to the remote server.
2. Once the connection is established, client get Output and Input Stream from the socket, then use those streams to send and receive data.

3.6.1.2 MPSocket Class Design

The primary design on MPSocket Class is concentrated on Constructor, getting Input and Output method:

1. **Constructor MPSocket(String host, int port):**

a) Read registry file to find the local MP (LocMP) the client belongs to and the Connect MP (ConMP) the server belongs to.

b) Establish new Socket Connection to LocMP, Send "REGISTRY" to apply for a global id(GID) for itself. Once got the GID, create a info file at local.

c) Establish new Socket Connection to ConMP, Send "HEADER" which includes the info of server. It also get the GID of server from ConMP as well.

d) Read the data from stream which was output by client application program, Send "REQUST" to ConMP.

e) Send "CONNECT_C" to LocMP periodically in order to remind LocMP that it's available maybe after a break or disconnection and expect LocMP forward the reply to it.

2. getOutputStream(),getInputStream()

Modify traditional getOutputStream() method, to make the Output/Input Stream be caught and processed before it is sent to server/client directly.

3.6.2 Communication between MP and Server

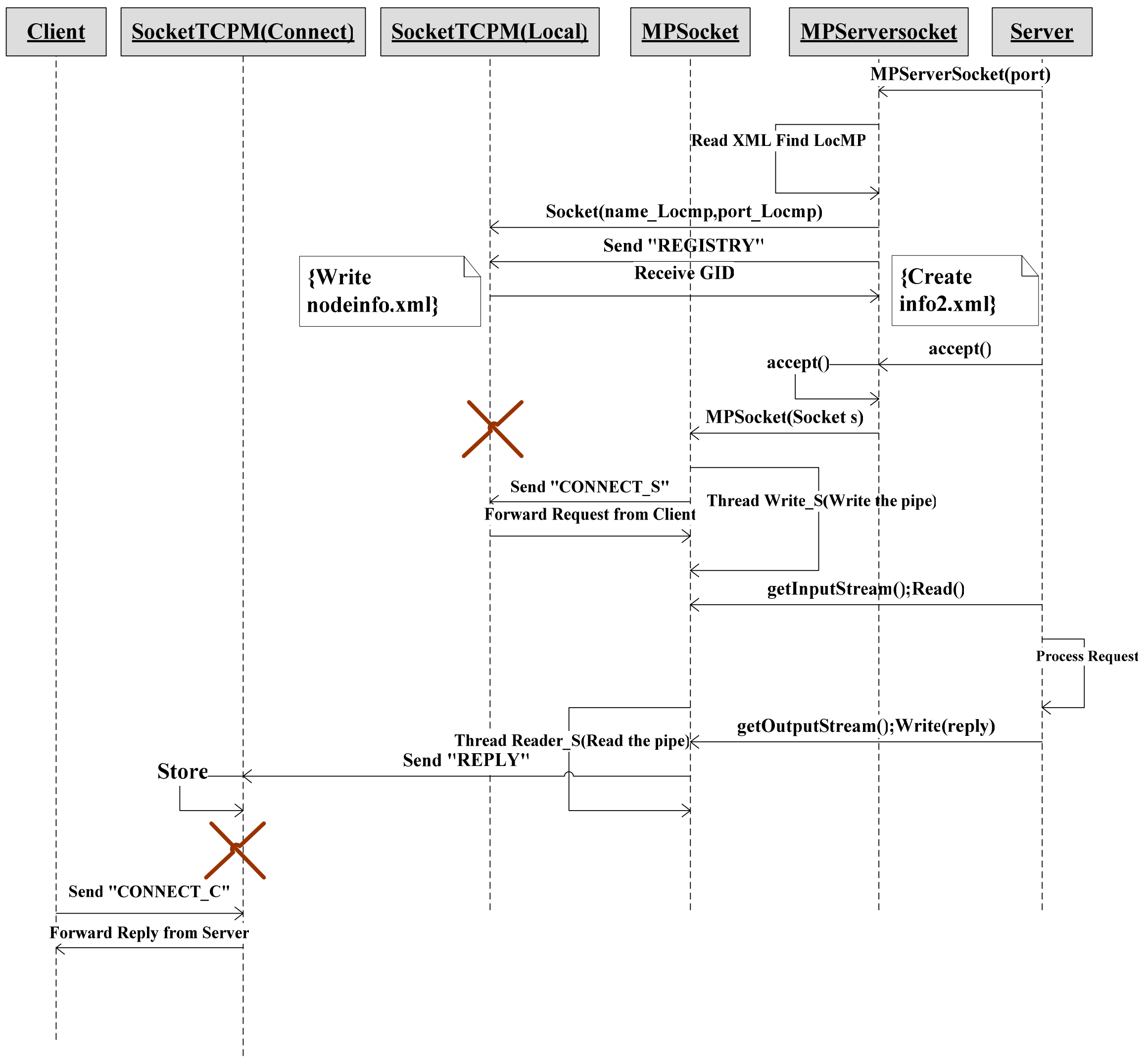


Figure 3-6: *Sequence From Server's View*

From the Server's view, it binds to a specified port, accepts connections from remote client on the bound port. Then it can receive the request from the client and send the reply to it.

3.6.2.1 Server Application Design

The basic life cycle of a Server application programs is in the following fashion:

1. Creates a new ServerSocket on a particular port using a constructor, MPServerSocket(port).

2. The ServerSocket listens for incoming connections attempts on that port using its accept() method.

3. Once the connection is established, accept() returns a Socket object connecting the client and the server.

4. Communicate with Client (receive request and send reply) by getting input and output streams.

3.6.2.2 MPServerSocket Class Design

The primary design on MPServerSocket Class is concentrated on Constructor and accept() method:

1. Constructor MPServerSocket(int port):

a) Read registry file to find the local MP (LocMP) the server belongs to.

b) Establish new Socket Connection to LocMP, Send "REGISTRY" to apply for a global id(GID) for itself. Once got the GID, create a info file at local.

2. accept():

GetOutputstream() and getInputStream() methods use Socket object which is returned by the accept() method. If we want to catch the data before it entered or came out of Socket stream, we have to modify accept() method to make it return MPSocket object.

There is an additional design for constructor MPSocket(Socket s) in MPSocket Class.

3.6.2.3 Supplement of MPSocket Class Design

1. Constructor MPSocket (Socket s):

a) Send “CONNECT_S” to LocMP periodically in order to remind LocMP that it’s available maybe after a break or disconnection and expect LocMP could forward the request to it.

b) Read the data from LocMP and output it into server’s stream which will be collected by server’s application by calling getInputStream().

c) Read the data from stream which was output by server application program by calling getOutputStream(), Send “REPLY” to ConMP.

2. getOutputStream() ,getInputStream()

Design switching mechanism to control the stream coming from client or server.

3.7 Design Meeting Place Architecture

Meeting Place nodes played an important proxy role in the model, It includes application programs design and in-kernel architecture design.

There is a user level proxy program running on these Meeting Place nodes, it works like a server, continues to listen for incoming connections on a specified port.

For in-kernel architecture in MP, it adopts event processing mechanism. In response to different events, it will use different way to handle.

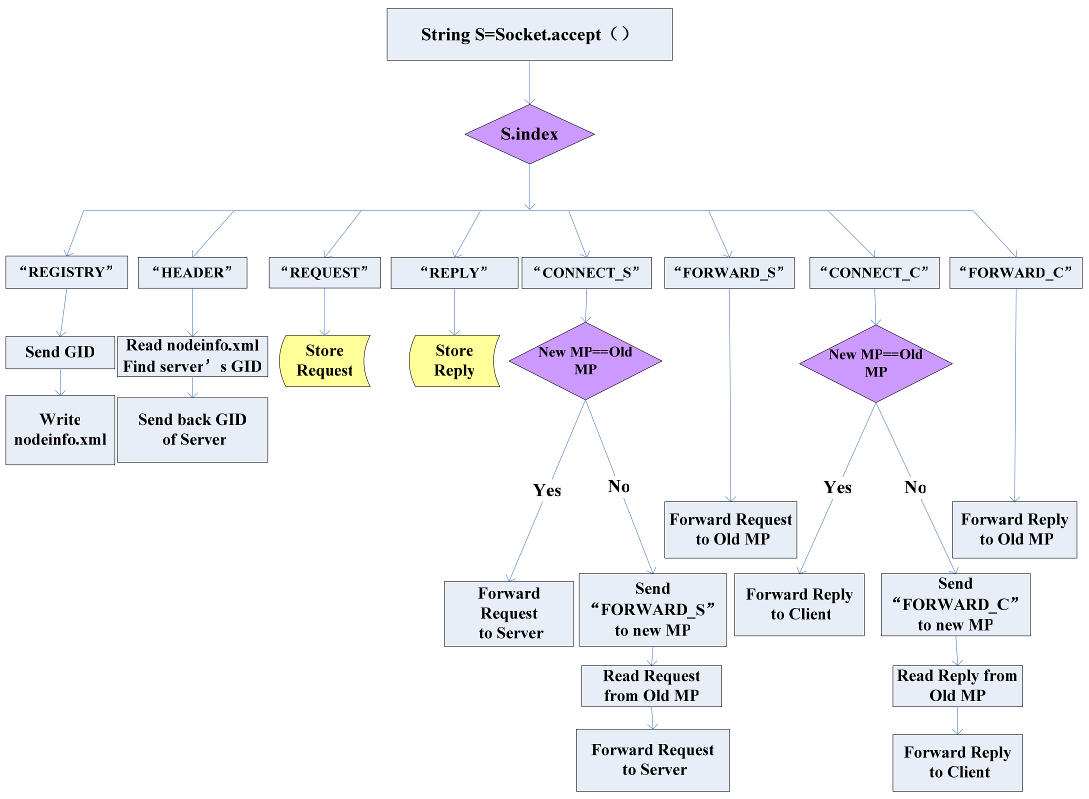


Figure 3-7: *Model of Meeting Place*

The architecture of MP is presented in the figure, Eight different event types are defined:

- **REGISTRY:**

  <Description>**:** for nodes (client/server) apply for registry in this MP.

  <Processing method>: generate a global id here and send it back to nodes.

- **HEADER:**

  <Description>**:** for sending some info of server(e.g. hostname, port) to this MP.

  <Processing method>: establish socket connection with server using these info.

- **REQUEST:**

  <Description>**:** for client send request to this MP.

<Processing method>: store the request until server/new MP come to collect it.

- **REPLY:**

  <Description>**:** for server send reply to this MP.

  <Processing method>: store the reply until client/new MP come to collect it.

- **CONNECT_C**:

  <Description>**:** for client remind this MP that it is available.

  <Processing method>: check if it's the original MP or new MP of client. If it's the original MP, send the reply info to client; if it's the new MP, send "FORWARD_C" to the original MP of client to get the reply. Then forward the reply to client.

- **CONNECT_S**:

  <Description>**:** for server remind this MP that it is available.

  <Processing method>: check if it's the original MP or new MP of server. If it's the original MP, send the request info to server; if it's the new MP, send "FORWARD_S" to the original MP of server to get the request. Then forward the request to server.

- **FORWARD_C:**

  <Description>**:** for remind this MP to forward data.

  <Processing method>:Find the reply in its memory, then send it to the applicant.

- **FORWARD_S:**

  <Description>**:** for remind this MP to forward data.

  <Processing method>:Find the request in its memory, then send it to the applicant.

Chapter 4: Implementation

This chapter describes the development of an instant message application and a web server application that were implemented to verify the design introduced previously. The implementation goals will be presented first. This is followed by a description of the application implementation, then we use a Class Diagram to show the relationship between all of Classes used, which is followed by introducing some Class implementations in detail. Some interface or thread mechanism are also be presented in the following subsections.

4.1 Implementation Goals

The aim of our implementation is to demonstrate that our model is successful in solving point-to-point communication problems in the mobile networks. Our principle is to minimize the changes of these connection-oriented operations, e.g. establish and close a socket connection, send and receive data. Therefore, we built a MPSockets library that allows applications to import and use a set of classes which look similar to the classes in traditional Socket library. For example, to establish a socket connection using MPSocket(host, port).

Another goal of our implementation is to make the mobility be transparent to accessing node as much as possible. For example, the client sent the data to a mobile server and it only need to know the original information of the server (hostname or IP address) rather than checking the connection state of server frequently.

Moreover, we try to minimize the operations for the mobile node before or after it moves. For example, in our design, it only needs to modify the locMP argument to represent it moved to a new area.

4.2 Instant Message Application Implementation

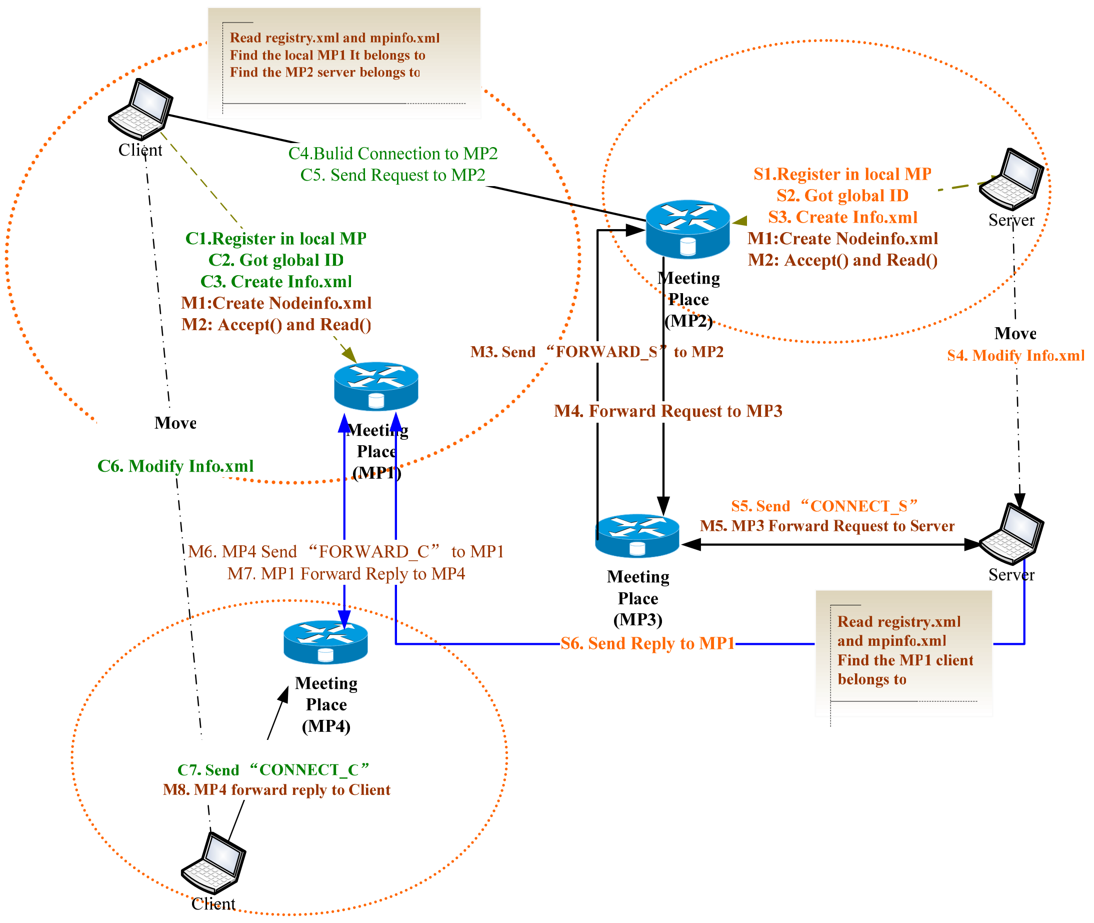


Figure 4-1: *Instant Message Application on MP Model*

This is an integrated implementation description for an Instant Message Application. In this scenario, both of Client and Server are mobile user. The steps are presented by different signs. For example, C1,C2...C5 represents the operation Client has done. S1,S2.... S6 represents the operation on Server. And M1,M2,...M6 represents the operation on Meeting Place.

The process could be described as follows:

Client read the local registry.xml and mpinfo.xml to check the local Meeting Place (MP1) in the area it belongs to. It also find the Meeting Place (MP2) is responsible for the area Server belongs to. Then Client will build Socket connection to Server by using MPSocket(String server, int port). During this process, Client applied for registering in MP1, and MP1 is responsible for generating a new global id for Client. Then Client will create a info.xml to keep this information. MP1 will keep a record for Client in nodeinfo.xml (which is created for the first node registry or appended other nodes register in it) as well.

At the same time or a little earlier, Server will listen to connection request by calling MPServerSocket(port) and accept(). Similar to Client registry operation, Server will check his local Meeting Place (MP2) and register in it.

Still in the process of MPSocket(String server, int port), Client established Java Socket connection with MP2, sent "HEADER" to it. MP2 read the information in the "HEADER" header, then read the nodeinfo.xml to find the global id of Server then send it back to Client.

Then Client using getOutputStream() and some write methods to send the request ("Hello, Server!") to MP2. MP2 accepted it and stored it until some node come to collect it. If the Server is available at this moment, it will send a "CONNECT_S" to MP2, then MP2 will forward the request to it. We described a much complex scenario in the figure above, the Server moved to a new place where MP4 is responsible for it. Server modified his argument-- " locMP " from MP2 to MP3 in info.xml file. Then it will send "CONNECT_S" to MP3, Once MP3 got it, it will keep a record for Server in his nodeinfo.xml file and check the global id in the "CONNECT_S" header. MP3 found Server's old MP is MP2 from the global id. Then it will send "FORWARD_S" to MP2, MP2 will find the request, forward it to MP3. After that MP3 forwarded request to Server. Server will use getInputStream() and some read method to read ("Hello, Server"), then wanted to send the reply ("Hi, I'm server!") to Client.

Server will read the registry.xml and mpinfo.xml to find MP1 is the original MP for Client. Then the reply was sent to MP1 by getOutputStream() and some write methods, which will store the reply and wait for some node come to collect it. However, Client moved to a new area (the operation is same to server moving). MP4 is the new MP for Client, once the Client sent the "CONNECT_C" to it, it will keep a record for Client in his nodeinfo.xml and check the first 4 bits in the global id, which represents the old MP of Client. Then it found the old MP is MP1, then MP4 sent "FORWARD_C" to it. After MP4 got the reply from MP1, it will forward the reply to Client. Client will use getInputStream() and some read methods to read the reply.

If more dialog between Client and Server, they just need to fill their request/reply into some read and write methods without using constructor to establish connection again.

4.3 Web Server Application Implementation

This is an implementation description for an Web Server application. In this scenario, Client would like to request a web page service, but before the Server process the request, Client disconnected the connection with the old Meeting Place, and moved to another place.

When the Server sent back the reply, it will find the old MP Client connected before moving. Then transfer the web page to it, which will store the data until some nodes come to collect it.

Once the Client was available in the network again, i.e. it modified its local info.xml and registered in its new MP, it will send the request “CONNECT_C” to its new MP to ask its new MP forward the reply to it. The new MP will check the Client’s global id and found that itself worked as a new MP. It will have a talk with Client’s old MP. Then it sent “FORWARD_C” to old MP to ask it forward the reply to it. After accept the reply, it will transfer the reply to the Client.

We used this application to verify our solution is suitable for huge volume data transmission. And it will ensure that the data won’t be out of order by dividing them into some segments and checking the sequence number.

4.4 XML documents Format Design

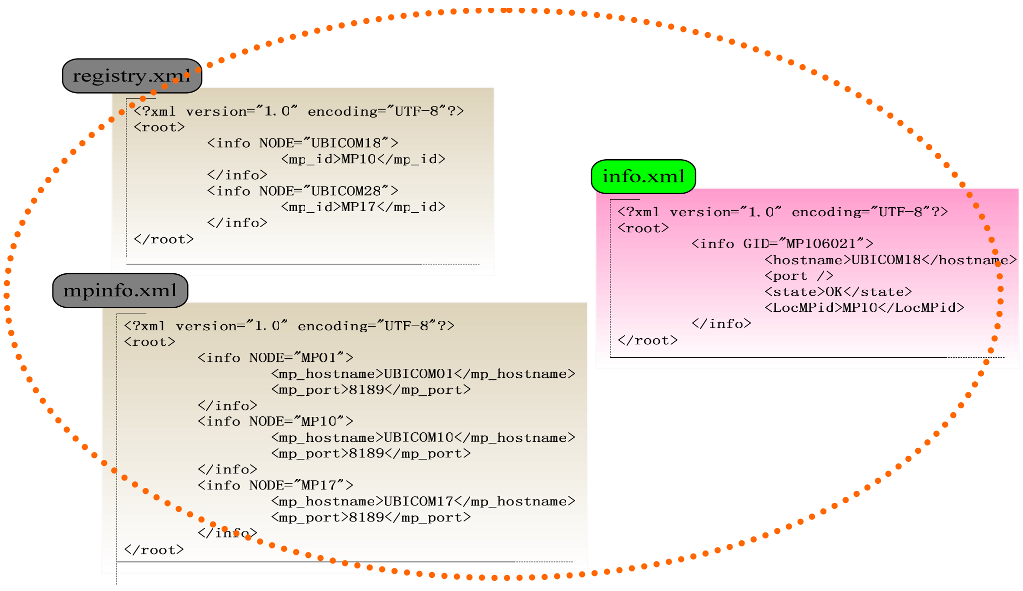
We chose XML documents format as the configuration files in the implementation for some reasons as follows[14]:

- XML documents look similar to HTML documents, and it’s easy to use and implement.

- XML allows you to create your own makeup tags that can accurately define both document structure and the meaning of the data. It separates data from formatting. In short, an XML document is a hierarchical data structure using self-definable tags.

- We can access XML documents easily within Java code by using some API. We used JDOM to read, write and manipulate XML documents in our implementation as it is Java optimized and it behaves like Java.

There is a sample of complete configuration files on client/server node and MP.



registry.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <info NODE="UBICOM18">
                <mp_id>MP10</mp_id>
        </info>
        <info NODE="UBICOM28">
                <mp_id>MP17</mp_id>
        </info>
</root>
```

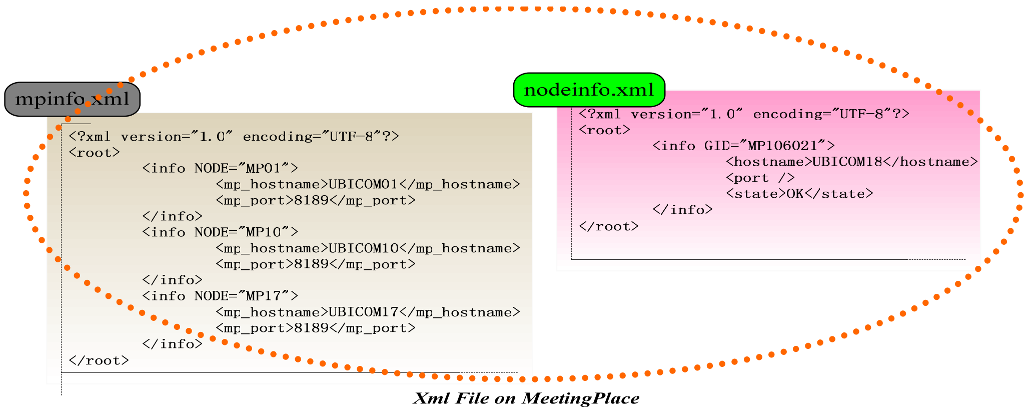
mpinfo.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <info NODE="MP01">
                <mp_hostname>UBICOM01</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
        <info NODE="MP10">
                <mp_hostname>UBICOM10</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
        <info NODE="MP17">
                <mp_hostname>UBICOM17</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
</root>
```

info.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <info GID="MP106021">
                <hostname>UBICOM18</hostname>
                <port />
                <state>OK</state>
                <LocMPid>MP10</LocMPid>
        </info>
</root>
```

Xml File on Each Node (Client or Server)



mpinfo.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <info NODE="MP01">
                <mp_hostname>UBICOM01</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
        <info NODE="MP10">
                <mp_hostname>UBICOM10</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
        <info NODE="MP17">
                <mp_hostname>UBICOM17</mp_hostname>
                <mp_port>8189</mp_port>
        </info>
</root>
```

nodeinfo.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <info GID="MP106021">
                <hostname>UBICOM18</hostname>
                <port />
                <state>OK</state>
        </info>
</root>
```

Xml File on MeetingPlace

Figure 4-2: *An sample of configuration files*

Some configuration files resided at local disk, some are generated later, therefore we use running stages to distinguish these configuration files.

- **Before running:**

For client/server nodes, they have same registry.xml and mpinfo.xml at local disk. Registry.xml describes the association between hostname and original MP id. mpinfo.xml describes the information of MP in detail.

For Meeting Place nodes, they have same mpinfo.xml as client/server has. mpinfo.xml describes the information of MP in detail.

- **On the first run:**

For client/server node, an info.xml file will be created after register in its local MP. This file describes registry info of nodes.

For Meeting Place node, an similar nodeinfo.xml file will be created if it's the first time some node apply for registering here.

- **Running time:**

For fixed node, it won't change anything for the configuration files in his local disk.

For mobile node, the only thing it need to do is modify the value of LocMP in info.xml from old MP id to new MP id.

For Meeting Place node, if a mobile node moved to his area and applied for registering here, it will append/modify registry info of this node in nodeinfo.xml if it exists. Otherwise, it will create a new nodeinfo.xml and add the registry info of this node. But it will use original global id for this mobile node rather than assign a new global id for it.

4.5 Class Relationship Design

In the implementation, we produced some other Classes to store data or operate configure file.

The relationships among these Classes could be described using a class diagram as follows[15]:

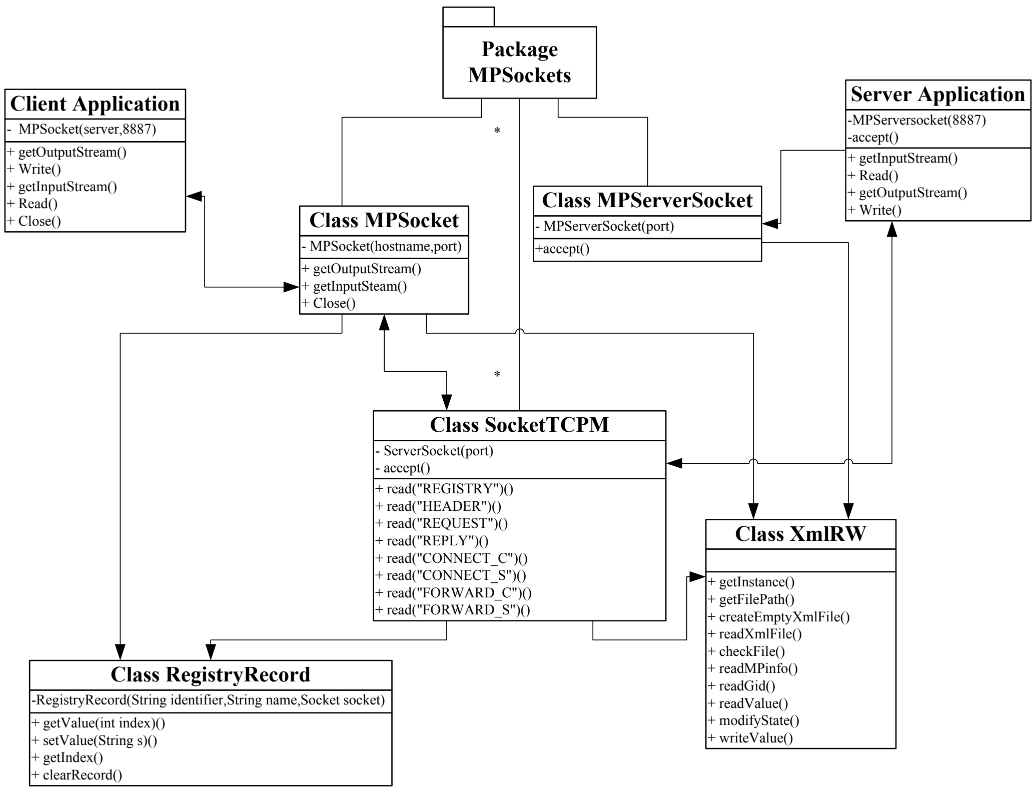


Figure 4-3: *Class Diagram of MPSockets*

Client and server application could use our MPSockets library by import MPSockets package. Then they can communicate in a traditional way, e.g. establish Socket connection, Data transmission and close Socket connection. The only difference is the Class name changed from Socket to MPSocket, ServerSocket to MPServerSocket.

SocketTCPM Class is the core program of Meeting Place. It will give different response to different EVENT request. The EVENT definition and processing methods have been provided in the Chapter 3.

RegistryRecord Class is used for maintain registry information and data transmitted for each node which has registered in the MP.

XmlRW Class is dedicated to process xml file operation, e.g. read, write or modify state.

In the following sections, I will describe these Class Implementation in detail.

4.6 XmlRW Class Implementation

As we mentioned above, we chose XML document format as the figuration file and used JDOM[16] as the API to access and modify configuration files. Therefore, we import some Classes:

```
import org.jdom.Document;
import org.jdom.Element;import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
public Class XmlRW {
        .....
}
```

Some methods are implemented:

```
public Document createEmptyXmlFile();
public boolean CheckFile();
public Document readXmlFile();
public String readMPinfo(String mp_id,String na);
public String findNode(String key);
public String readGid();
public Element findAttr(String attr);
public String readValue(String key,String value);
public String readLocMPID (String id);
public boolean ModifyState(String filepath,String key,String state);
public void WriteValue(String key, String name,String port, String state, String LocMPid);
```

Through these methods, it will be much easier to operate on the XML documents. For example, if we want to create an XML file by using createEmptyXmlFile(), it will check if the file path exists or not, then create a new file.

If we'd like to read the information of a MP, we call readMPinfo(String mp_id, String na), get the name of MP or port of MP by knowing MPid. We also could read the value of some elements by input the attribute values.

Method ModifyState(String filepath,String key, String state) is used to modify element value by inputting key value and state. It was used for each Meeting Place to check the state of data it stored. Some states are set, e.g. when data is arrived, the state is set "D", then once a node sends "CONNECT_C" , "CONNECT_S", "FORWARD_C" or "FORWARD_S", it will check if there is some data for it by using global id in the header. If some were found, it will modify the state to "C", and began to transmit the data to requestor. Once all of the data are transmitted, it will change the state to "OK".

Method WriteValue(String key, String name,String port, String state, String LocMPid) is used to insert some information into nodeinfo.xml and info.xml which reside on Meeting Place nodes and Client/Server nodes individually.

4.7 Data Storage Implementation

Due to our design is not aimed to solve large amount data issues, we didn't choose using some database to store data. In our implementation, we use an array to keep the request/reply data. It's shown as follows:

```
private static RegistryRecord[] reg_array=new RegistryRecord[100];
```

It's necessary to mention the implementation of RegistryRecord Class. Considering there is some associations between data and its destination, for example, we should know where to send a request or a reply. Therefore, we design the RegistryRecord Class which have some elements: gid, host_name, value_array,etc. A Meeting Place node will create an instance of RegistryRecord Class by using its constructor, once some nodes sent "REGISTRY" header to apply for register in it.

The constructor could be described as follows:

```
RegistryRecord(String identifier,String name,Socket socket)
    {
            host_name=name;
            gid=identifier;
            s=socket;
            index=0;
            value_array=new String[100];
    }
```

For example, if a client node wants to register in its local MP, it will input its global id and hostname and current socket as the arguments.

```
RegistryRecord reg=
            new RegistryRecord(MP_id+node_gid,client_hostname,socket_c);

reg_array[index]=reg;
index++;
sum_reg++;
```

Then, it was kept as a record in the MP's memory. And if more nodes register here, the index will increased and sum of registry nodes (sum_reg) will increase as well. But at the registry stage, there is no data in the instance of Class.

On the stage of storing data, i.e. Once MP accepted "REQUST" / "REPLY" header, it will split the global id of Server/ Client in the header, then checking all of the records in its array using the unique global id of Server/Client. Then insert the data into the record by calling setValue method in the RegistryRecord Class.

```
for(int i=0;i<sum_reg;i++)
{
    if(reg_array[i].gid.equals(client_gid))
    {
        reg_array[i].setValue(rep);
    }
}
```

The rep data will be kept in the RegistryRecord Class as a record for this global id.

```
public void setValue(String va)
    {
            if(va!=null)
            {
                    value_array[index]=va;
                    index++;
            }
    }
```

If more data needs to be stored in it, the index will increase. The index number are necessary for the MP to check if some data are transmitted out of order.

On the stage of transmitting data, i.e. Once MP accepted such header:

“CONNECT_C”/“CONNECT_S”/”FORWARD_C”/”FORWARD_S”, it will split the global id of Server/ Client in the header, then checking all of the records in its array using the unique global id of Server/Client. To find the data for this global id, firstly we should call getIndex() to find out how many request/reply would be transmitted. Then invoke getValue(i) method to get each value of it. After find the data, they will be output into a stream one by one. If all of the data were transmitted, we will clear the record in the instance by using clearRecord().

```
for(int i=0;i<sum_reg;i++)
{
    if(reg_array[i].gid.equals(client_gid))
    {
            int num_reply=reg_array[i].getIndex();
            String re=null;
            for(int j=0;j<num_reply;j++)
            {
                    re=reg_array[i].getValue(j);
                    OutputStream ou=socket_c.getOutputStream();
                    ou.write(re.getBytes());
                    ou.flush();
            }
            reg_array[i].clearRecord();
            node_info.ModifyState("D:\\nodeinfo.xml",client_gid,"OK");
```

Considering the number of bytes in a request or reply probably more than 1024, e.g. for the web server application, the webpage as the reply have loads of bytes. Thus, each time read from a stream, at most 1024 bytes could be read out once.

4.8 MPSockets Library

Some detailed design and implementation in MPSockets Library are presented here, which also play an important role in the model.

4.8.1 GID interface

As we talked above, global id (GID) is needed for the implementation of model. To guarantee the uniqueness of the ID, we implement a GID interface. It is responsible for generate a random number, which will be combined with the MPid, then CheckGID(String a) will be used to check if the ID is unique or not.

```
public interface GlobalIdentifier
{
        int getRandomGID();
        boolean CheckGID(String a);
}
```

4.8.2 Pipe Stream Design

getOutputStream() or getInputStream() are used by application program to get Socket Output stream or input Stream, then output some data using write() method or read some data by using read() method.

As we have mentioned in Chapter 3, we have to modify these two methods, to catch the data before it's sent to output stream or read from input stream directly. By this way, we have chance to process the data.

We introduced two pipe streams for Client.

```
ois_c=new PipedInputStream();
oos_c=new PipedOutputStream(ois_c);
ios_c=new PipedOutputStream();
iis_c=new PipedInputStream(ios_c);
```

Take the Client for example, ois_c and oos_c are a pair of pipe stream, oos_c is a new pipe output stream, it will output data to the inputStream ois_c. Similar to it, ios_c and iis_c is another pair of pipe stream for Client.

We introduced another two pairs of pipe streams for Server. Similar to above.

```
ios_s=new PipedOutputStream();
iis_s=new PipedInputStream(ios_s);
ois_s=new PipedInputStream();
oos_s=new PipedOutputStream(ois_s);
```

getOutputStream() are modified as follows:

```
public OutputStream getOutputStream() throws IOException
    {
            if(flag)
            {
                    return oos_c;
            }
            else
            {
                    return oos_s;
            }
    }
```

The flag is used for distinguishing the output pipe stream is for Client or Server. If flag=true, it will return an output pipe stream for client, otherwise for server. Once Client application program calls socket.getOutputStream(), then use write() method to write some data. The data would be transmitted to InputStream ois_c. Once Server application program calls socket.getOutputStream(), then use write() method to write some data. The data would be transmitted to InputStream ois_s.

getInputStream() are modified as follows:

```
public InputStream getInputStream() throws IOException
    {
            if(flag)
            {
                    return iis_c;
            }
            else
            {
                    return iis_s;
            }
    }
```

Similar to getOutputStream, the flag is used to distinguish the input pipe stream is for Client or Server. If flag=true, it will return an input pipe stream for Client, Once Client application program calls getInputStream(), then use read() method to read data. The data will be read from output stream ios_c. Otherwise, it will return an input pipe stream for Server, Once Server application program calls getInputStream(), then use read() method to read data. The data will be read from output stream ios_s.

4.8.3 Threads implementation

Four core threads are used in our implementation, which are mainly used to communication with pipe stream and Java socket stream. There is a unlimited loop in each thread[16], which will be stopped when user close socket connection.The implementation description of each Thread are described as follows:

Thread	Writer_C	Reader_C	Writer_S	Reader_S
Step 1	Read data from pipe stream of Client	Build connection to its local MP, send "CONNECT_C"	Build connection to its local MP, send "CONNECT_S"	Read data from pipe stream of Server
Step 2	Send "REQUEST" header (data is included) to the connected MP which is responsible for the area Server belongs to	Read data from socket stream of its local MP	Read data from socket stream of its local MP	Send "REPLY" header(data is included) to the connected MP which is responsible for the area Client belongs to
Step 3		Send data to pipe stream of Client	Send data to pipe stream of Server	

Table4-1: Threads Implementation Description

Chapter 5: Evaluation

After the introduction of our design and some implementation, we faced the problems about how to evaluate our solution which is on the transport layer. We don't aim at improving some performance results (throughout or latency). Our goal is to handle the disconnection issue in the mobile networks.

Therefore, I'd like to evaluate my project from four aspects: compare with traditional Java Socket communication, analysis application area description and comparison with some solutions in the related works.

5.1 Compare with traditional Java Socket

I'd like to use the experiment result to show the performance from several aspects with comparing to Java Socket.

Once disconnection happened in the connection time or during the data transferring, some exceptions will come out because it broke the requirements of traditional end-to-end communication. Even if the network was recovered or mobile nodes moved to another place and connected to the internet again, still no chance to get the data forwarding.

For our implementation, we caught and handled the connection exception by monitoring the network connection timely or informing available state to MP nodes.

The detailed description could be seen as follows:

	Traditional Java Socket	MPSockets Solution
Disconnect happened on Client/Server	**Exception:** Connection Reset	Network isn't well enough, I'd like to have a sleep
Network connection recovery	**Errors:** The program couldn't work any more	Client/Server will send "CONNECT_C"/"CONNECT_S" header to its local MP, and got the reply/request from its local MP.
Client/Server moved to another area	**Exception:** Connection Reset	Client/Server modified its LocMP argument in the info.xml
Client/Server connect to its new MP	**Errors:** The program couldn't work any more	Client/Server send the "CONNECT_C"/"CONNECT_S" to its new MP, which will send "FORWARD_C"/ "FORWARD_S" to old MP, which will forward reply/request to it. Then it will transfer the reply/request to Client/Server.

Table5-1:Comparison between Java Socket and MPSockets

5.2 Application Area

MPSockets works as the transport layer solution, which could be applied to many potentially application area. In the following area, we will illustrate various different applications in which we think MPSockets can be used.

5.2.1 Instant Message Application

In the former chapter, the implementation description has been made for this application. Here I'd like to extend the application scenarios further. As we know, some popular Instant Message software[18] (such as OICQ, MSN etc.) work well in the wired network. But they couldn't perform well in the mobile networks due to they followed Client/Server structure and traditional end-to-end communication. The disconnection issues badly impacted the quality of services in the network, which will also degrade the communication reliability and quality.

However, some field requires high reliability of data transmission, such as military field. By using MPSockets solution, the mobile nodes could transfer message in secret and ensure that messages won't be out of order.

5.2.2 Web Services Application

We have implemented a web server application, which can be used for some mobile user to access a web page while he's moving. For a overload remote server, especially low bandwidth, it's hard for it to give response immediately for loads of client requests. But some clients couldn't tolerant such a long time waiting, some exceptions will happen. However, some users don't mind the delay for the response. They would like to break the network connection at this moment and bring their laptop to a cafe shop then connect again. For this scenario, our solution couldn't be more suitable because it could solve the short connection broken problems and provide support for nodes' mobility.

We also could extend the web server application to some more complicated web services[19] applications. Some problems faced by network services quality and some latency caused by processing capability or complex database operation could be more tolerant by using our solution. Some time-consuming services could be accepted by mobile user, because they don't need to waste time waiting the response. They could move to their destination location and accept the reply immediately. It would make work and life more efficiency.

5.2.3 Streaming Server Application

Let's imagine some multimedia application which is provided by streaming server[20]. Client could accept some streaming media from a remote streaming server across the Internet. At present, standard RTP (Real-Time Protocol) and RTSP (Real-time Streaming Protocol) [21]are used for the streaming transferring. But we believe that there are still some applications require a high quality and reliability of data transmission. The requirements for keeping a perfect network connection would be extremely strictly. If a short broken happened in the network, the client will miss some data. And if the server is a mobile node, that's totally impossible for guaranteeing the integrity of data transmitted.

By using our solution, a short time network disconnection could be handled, so it won't cause data loss. It also won't affect data transmission no matter server or client is mobile. Once the mobile nodes is available in the network again (maybe in a new location), it would inform its MP to continue the data transferring or receiving. And due to the data storage and sequence control mechanism we designed, the data won't be out of order.

So you can enjoy your movie or music smoothly even the server isn't available for a moment. Or you can continue with your movie when you back home, and the server is not necessary connect to the internet at the same time due to the MP nodes buffered the packets server sent.

5.2.4 Data Collection Application

In the former application description, we put the emphasis on providing solutions for the mobility and network broken problems. In this data collection application, I'd like to describe the data storage is also an advantage of our solution.

A mobile server moved to an area and collected the data from several clients distributed in this area. For the mobile server, once it moved to a new location, it will register in a new MP which is responsible for the area. Similar, each client nodes no matter it's mobile or not, it will be assigned a global id. By inserting a few known and stable MP nodes, the data transmitted from client to server, will be stored in the MP(server's local MP) first. And due to our data storage and sequence control mechanism, the data will be stored for each global id. By this way, once some nodes came to collect the data, it won't be hard to distinguish which server/client this data belongs to and where the data came from and where to go. For a huge volume data transmission, which would be divided into several segments, which will guarantee the sequence of data transferring.

5.3 Comparison with Other Solutions

We have described some popular solutions in the chapter 2—related work, such as Mobile IP and some transport layer solutions. We choose split-tcp and indirect-tcp to make a comparison with our solutions because it will make sense by comparing solutions which have similarity in some aspects. For example, split-TCP and indirect-TCP are also by inserting some intermediate nodes to short the path length between source node and destination node.

	MPSockets	Split-TCP	Indirect-TCP	Mobile IP
layer	On top of transport layer	Transport layer	Transport layer	Network layer
Compatible with TCP	Yes	No	No	Yes
Intermediate nodes	Yes	Yes	Yes	Yes
Number of intermediate nodes for each session	2	By the length of path	1	2
End-to-end semantic	No	No	No	Yes
Name for intermediate node	Meeting Place (MP)	Proxy Node	Access Point (AP)	HA FA
Mobility of intermediate node	Fixed	Mobile	Fixed	Fixed
Function of intermediate node	unified	unified	unified	HA and FA are different
Mobility of Client/Server	Either/Both	Either/Both	Either	Either
Package buffered at intermediate node	Yes	Yes	Yes	No
Inform old proxy	No necessary	“LACK” to confirm each packet	N/A	Always inform HA his new info
Changes in Client/Server	No	Yes	Yes(mobile node)	No

Table 5-2:Comparison between MPsockets and other solutions

We’d like to analyze the features of our solution by comparing with others.

Our solution worked on the top of transport layer which was implemented by modifying traditional Java Socket library and inserting Meeting Place nodes in the model. It’s clear that Mobile IP is a network layer solution for mobility and other solutions are totally transport layer solution. Therefore, our solution and Mobile IP are compatible to TCP while split-TCP and indirect-TCP are not.

As we have mentioned, we choose the solutions which have some similarities in some aspects. For example, they are have the intermediate nodes even they are different meaning. For MPSockets, they are called Meeting Place (MP), number of MP could be scaleable, but for each session, only two MPs which are responsible for their local area individually. However, for the split-TCP, the number of intermediate nodes are not fixed, it all depends on the path length and the packets it received. And the intermediate nodes are also mobile nodes. For indirect-TCP, only one intermediate node, which is called Access Point (AP), it is used to divide the communication between source node and destination node into two parts. To mention that, it focus on handling one mobile node not both in a half wireless and half wired network. For the Mobile IP, it seemed each network also have their own agent, which are called Home Agent (HA) and Foreign Agent (FA). Due to type of care of address mobile node choose, FA may be not used for transmitting data. And there are some difference on the operation of packets for HA and FA.

Due to inserting intermediate nodes in these solutions, most of them provide buffer function in these nodes except mobile IP.

To mention that, in our solution, the mobile nodes don't need to inform their old MP about new MP. But for Mobile IP, the mobile nodes have to inform their HA about their new address frequently. For the split-TCP, some local ACK (LACK) should be sent to the neighbour MP to make sure it got the data transmitted to it. So for the other solutions, the cost for informing mechanism will be increasing due to node's mobility.

Our principle is to minimize changes of connection-oriented operation for user, so there is almost no difference for users to establish connection, close connection and send/receive data. But for the transport layer solution, they changed too much and couldn't guarantee no changes on mobile nodes. And Mobile IP is a network layer solution, which didn't need to change the operation on the nodes as well.

Chapter 6: Conclusions

This chapter makes a conclusion of the thesis, which presents the purpose of my design and how the objectives are implemented. The first section lists the contributions for this area, the final section on future work will provide some suggestions on the further research and development which could be carried out.

6.1 Contribution and Completed Work

At the beginning of the thesis we introduced the problems faced by traditional end-to-end protocols such as TCP in the mobile networks. Some existing solutions are also put forward in the related works. I choose some solutions which have similarities with ours to make a comparison in the evaluation chapter. Combined with the results from comparison with traditional Java Sockets communication and application area analysis, we could make a relative integrated conclusion of our solution.

- We construct a new model by introducing "Meeting Place" concept and design MPsockets library which addresses the mobility problems in traditional end-to-end protocols.

- Minimize changes of connection-oriented operations. e.g. establish and close a socket connection, send and receive data. In the traditional Java Sockets communication, some constructors and methods are used in the Java Sockets library, our solution is to minimize modification existing methods and Classes in the Java Sockets. Moreover, for the design of new MPSockets library, we modified the name based on the traditional one, e.g. from Socket Class to MPSocket Class.

- Make the mobility be transparent to accessing node. Our goal is to decrease the impact of mobility issues in the end-to-end communication. Therefore, the best way is to hide the mobile behaviour of accessed nodes. E.g. A client is enjoying a service from a remote mobile server, it won't aware any difference on the quality of services even if some network disconnection issues happened on the server side.

- Minimize operations for mobile nodes before or after it moves. The only thing to do for the mobile node is change their new MP in a configure file, then a request could be sent to the new MP to ask for registry.

- Using configure file make it much flexible and extensible. In the chapter 3, we discussed some association issues should be solved, such as MPid and MP's detailed information, client/server nodes and MP. In the chapter 4, we also describe why choose XML documents as the configure file, and how to operate it by using JDOM. Moreover, by using configure file we could extend system scale in a simple way, e.g. adding more Meeting Place nodes. It also made the system more flexible by using MP id to represent a MP while the association info of MP were put in a separated file.

- By using a few known and relatively stable network nodes as "Meeting Place" nodes, it reduced cost of persisting network quality. And using fixed nodes is much stable than using mobile nodes (e.g. in split-TCP). Moreover, the MP nodes have the function to store the data and ensure no out-of-order happened for the data transmission.

- Could be used in some application area. From the analysis result of application area in evaluation chapter, MPSockets would be suitable for some fields which have high quality requirements on quality of service of networks or data transmission reliability in mobile networks.

6.2 Future Work

In this section, I'll present the future work by analyzing some assumptions which was made for convenient and reasonable design and implementation.

- **Assumption 1:**

  There are always a few known and stable network nodes as "Meeting Place" nodes.

  Completed Work 1:

  I used XML document files to configure the initial association relationship between client/server nodes and MP nodes. Client/Server application program could know which MP it belongs to or accessing nodes belongs to in the initial registry file.

  Future Work 1:

  To develop some Meeting Place searching mechanism, which could replace this assumption and make the Meeting Place nodes are transparent to client/server nodes. This could be implemented by some routing protocols to find its nearest local MP.

- **Assumption 2:**

  I assumed that this solution is used for solving mobility issues in mobile networks.

  Completed Work 2:

  I used two experiments to verify our solution in mobile networks. I simulated disconnection and nodes mobility by unplug network cable and modifying configuration files to new MP.

  Future Work 2:

  I'd like extend our solution to wireless Ad hoc application area, because there are some similar characteristics between wireless networks and mobile networks. We need to do more experiments to verify if it could be used for the wireless networks directly or some modifications are needed.

- **Assumption 3:**

I assumed that Meeting Place nodes have some proxy functions like receiving data and transferring data. And it also could buffer the data it received.

Completed Work 3:

In my solution, Meeting Place nodes handled different requests from client/server nodes by analyzing headers it accepted, and it also provides data storage and sequence control mechanism to ensure the data integration and order.

Future Work 3:

For the future work, some improvements could be made to Meeting Place nodes, to make them more smart and intelligent.

Chapter 7: References

[1] M.Satyanarayanan. “Fundamental Challenges in Mobile Computing.” *In Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing (PODC'96), Philadelphia, Pennsylvania, USA, May 1996.*

[2] George H. Forman and John Zahorjan. “The Challenges of Mobile Computing.” *IEEE Computer Society Press CA, USA, Volume 27 , Issue 4,April 1994,Pages: 38 - 47*

[3] Murthy C. Siva Ram, Manoj B.S. “Ad Hoc Wireless Networks,” *Prentice Hall Communications Engineering and Emerging Technologies Series,2004,*

[4] Andrew S. Tanenbaum,“Computer Networks,” *fourth edition, Pearson Education International, 2003,pp.532-555.*

[5] Elliotte Rusty Harold. “Java Network Programming,” *Third Edition, O’reilley.*

October 2004,pp.275-285,325-330.

[6] Perkins, C.E. “Mobile networking through Mobile IP,” *Internet Computing, IEEE Volume 2, Issue 1, Jan.-Feb. 1998 Page(s):58 - 69*

[7] A.Bakre and B.R.Badrinath, “I-TCP: Indirect TCP for Mobile Hosts,” *Proceedings of IEEE ICDCS,May 1995,pp.136-143.*

[8] H.Balakrishnan, S. Seshan, and R. Katz, “Improving Rliable Transport and Handoff Performance in Cellular Wireless Networks,” *ACM/Baltzer Wireless Networks Jouranl, , December 1995,vol.1, no. 4, pp. 469-481.*

[9] K. Brown and S. Singh, “M-TCP: TCP for Mobile Cellular Networks,” *ACM Computer Communication Review, , October 1997,vol. 27, no. 5, pp. 19-43.*

[10] Kopparty S., Krishnamurthy S. V., Faloutsos M. and Tripathi S. K. “Split TCP for Mobile Ad Hoc Networks,” *Proceedings of IEEE GLOBECOM 2002, vol. 1, pp. 138-142.*

[11] Yi-Min Wang, Damani. O.P. Woei-Jyh Lee, "Reliability and availability issues in Distributed Component Object Model(DCOM)", *Proceedings of IEEE Community Networking ,1997*

[12] Cesar Munoz, Janusz Zalewski, "Archictecture and Performance of Java-Based Distributed Object Models: CORBA vs RMI", *Real-Time Systems, Volume 21 Issue 1-2, July 2001*

[13] Christian Nester, Michael Philippsen, Bernhard Haumacher, "A more efficient RMI for Java", *Proceedings of the ACM 1999 conference on Java Grande*, June 1999

[14] Brett McLaughlin, "JAVA and XML," *Second Edition, O'Reilly, August 2001*

[15] IBM UML Resource Centre website, Retrieved September 2006.

http://www-306.ibm.com/software/rational/uml/UML

[16] The home of JDOM, http://jdom.org, Retrieved September 2006

[17] Scott Oaks and Henry Wong, "Java Threads", *Second Edition, O'Reilly, January 1999*

[18] Wekipedia website, Retrieved September 2006.

http://en.wikipedia.org/wiki/Instant_messaging

[19] Ethan Cerami, "Web Services Essentials", *First Edition, O'Reilly, February 2002*

[20] Sungjoo Kang, Ku, K.I., Jeong Min Shim, "Performance Evaluation of Software Streaming Server Architecture for Massive Users", *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference, Volume 3, 20-22 Feb. 2006 Page(s):1736 – 1741.*

[21] Yong-Ju Lee, Ok-Gee Min, Hag-Young Kim, " Performance evaluation technique of the RTSP based streaming server", *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on 2005 Page(s):414 - 417 .*