UNIVERSITY COLLEGE LONDON

Department of Computer Science
MSc DCNDS

Reliable Asynchronous Middleware for Mobile Ad Hoc Networks

JMS Implementation for MANETs using Sociable nodes (JIMS)

Supervisor: Dr Cecilia Mascolo

Team members:

Haitian Chen

Liang Chen

Kavitha Gupta

Christos Savvidis

Wai Git Teo

Date: Monday, 6th September 2004

Table of Contents

1. INTRODUCTION 7

2. BACKGROUND 10

2.1 Enterprise Messaging *[23]* 10
2.2 Motivation for providing Message Oriented Middleware for MANETs 11
2.3 Overview of the Java Messaging Service (JMS) *[9]* 14

3. RELIABILITY 18

3.1 Importance of Reliability in MANET 18

4. INTRODUCTION TO JIMS 20

4.1 Assumptions 20
4.2 Challenges of Mobile Computing Environment 20
4.3 Functional Requirements 20
4.4 Non-functional Requirements 21
4.5 Overview of JMS Implementation for MANETs using Sociable nodes (JIMS) 22
4.6 Node Profiles 22
4.7 The Beacon 27

5. JMS IMPLEMENTATION FOR MANET USING SOCIABLE NODES 29

5.1 Introduction 29
5.2 The Publish/Subscribe Model 29
5.3 The Point-to-Point Model 39
5.4 Message Delivery Modes 40

6. SYSTEM EVALUATION 42

6.1 Introduction 42
6.2 OmNet ++ 42
6.3 Mobility Models 43
6.4 Hop Count Vs Buffer Size 44
6.5 The Publish/Subscribe Model 44
6.6 The Point-to-Point Model 54

7. PROTOTYPE 62

7.1 Introduction 62
7.2 Overview 62
7.3 Design 62
7.4 Basic Building Blocks of a JMS Application 68
7.5 The Graphical User Interface (GUI) 71
7.6 Enhancements and Issues 71
7.7 Profiles and Instances 72
7.8 Test Cases 73

8. PROJECT MANAGEMENT 75

8.1 Introduction 75
8.2 Iterative Incremental Development 75
8.3 Project Management Methodology 76

Table of Contents

8.4 Team Structure 79
8.5 Team Communication and Status Monitoring 80
8.6 Risk Management 81
8.7 Project Milestones 81

9. CONCLUSION 84

9.1 Introduction 84
9.2 Related Research 84
9.3 Comparison 85
9.4 Critical Analysis 86
9.5 Future Work 87

10. APPENDIX I – SIMULATION FILES, COMPILATION AND EXECUTION 89

11. BIBLIOGRAPHY 94

Acknowledgements

The group expresses its gratitude to the supervisor Dr. Cecilia Mascolo for her guidance, support and encouragement for this project.

Mr. Mirco Musolesi was invaluable as an external advisor. Indeed the simulation would not be in its current shape if not for his guidance. We would like to thank him for his guidance when we were getting to grips with OMNeT++ and letting us use his mobility models. This enabled us to focus on our task of evaluating the algorithm rather than simulator itself. We are very grateful to him for his time, effort and extensive assistance during the design of the algorithm and analysis of the simulation results.

We would also like to thank Mr. Amit Gupta for his help in editing and formatting this report.

List of Figures

Figure 1 – Enterprise Messaging 10

Figure 2 – Message Oriented Middleware 11

Figure 3 – Publish Subscribe Messaging Model 15

Figure 4 – Point to Point Messaging Model 16

Figure 5 – MANET Application 18

Figure 6 – JMS Implementation of JIMS 22

Figure 7 – Node profiles 23

Figure 8 – Step 1 29

Figure 9 – Step 2 31

Figure 10 – Step 5 34

Figure 11 – Step 5 34

Figure 12 – Step 5 35

Figure 13 – Step 5 36

Figure 14 – Step 5 36

Figure 15 – Message ratios at Different Population Density using Random Waypoint Mobility 46

Figure 16 – Message ratios at Different Population Density using Group Mobility 47

Figure 17 – Message ratios at Different Simulation Time for 32 Nodes 48

Figure 18 – Percent of Message Delivered for 32 Nodes Using Different Buffer Sizes 49

Figure 19 – Latency for 32 Nodes Using Different Buffer Sizes 50

Figure 20 – Distribution of Delivery Ratio (8 Nodes, RWP) 51

Figure 21 – Distribution of Delivery Ratio (16 Nodes, RWP) 52

Figure 22 – Distribution of Delivery Ratio (32 Nodes, RWP) 52

Figure 23 – Distribution of Delivery Ratio (64 Nodes, RWP) 52

Figure 24 – Distribution of Delivery Ratio (8 Nodes, GMM) 53

Figure 25 – Distribution of Delivery Ratio (16 Nodes, GMM) 53

Figure 26 – Distribution of Delivery Ratio (32 Nodes, GMM) 53

Figure 27 – Distribution of Delivery Ratio (64 Nodes, GMM) 54

Figure 28 – Message ratio for different population densities for RWP and GMM 56

Figure 29 – Message ratio for different percentage of receiver nodes for 32 nodes.....57

Figure 30 – Latency for different population densities for RWP and GMM 58

Figure 31 – Latency for different percentage of receivers for 32 nodes.....59

Figure 32 – Message ratio as a function of time and population density for GMM.....60

Figure 33 – Latency distribution for different population densities for GMM.....60

Figure 34 – Message ratio for different speeds for 50% of the population as receivers.....61

Figure 35 – Class Diagram 64

Figure 36 – Building Blocks of JMS Application 69

Figure 37 – Iterative Incremental Development.....75

Figure 38 – Extreme Programming Practices (source – Xprogramming.com) 77

Figure 39 – Risk Management 81

List of Tables

Table 1 – Node table.....30

Table 2 – Simulation Parameters.....44

Table 3 – Characteristics of Random Waypoint and Group Mobility as a Function of Population Density 50

Table 4 – Distribution of Delivery Ratio Using Random Waypoint and Group Mobility.....51

Table 5 – Simulation Parameters.....55

Table 6 – Characteristics of Group Mobility and Random Waypoint Model for different population densities 58

Table 7 – The Random Waypoint performance for two different speeds.....61

Table 8 – Test Cases.....73

1. Introduction

The advent of ubiquitous computing and proliferation of portable computing devices have raised the importance of mobile and wireless networking *[1]*. Mobile working and entertainment are increasingly becoming popular. Current wireless technologies like GSM, have mobile devices as the last hop, with each device connected to the wired network via a base station, wireless access point etc. However, this dependency on fixed infrastructure is expensive, not suitable for some situations (military and emergency situations) and limits the flexibility of mobile devices.

A mobile ad-hoc network (MANET) is a network made up of a collection of mobile, autonomous nodes that communicate using wireless multi-hop links without any fixed infrastructure such as base stations. Only pairs of nodes that lie within each other's transmission radius can communicate directly. However, each node in the network is supposed to act as a router and participate in forwarding packets for other nodes *[2]*.

MANETs are characterised by intermittent connectivity, resource (power, memory and bandwidth) constraints, frequent changes in topology, heterogeneous devices and network interfaces and lack of security *[27]*. Hence, MANETs are inherently problematic. Some of these issues are in common with wired networks, however, on a much larger scale.

The idea of being able to form a network in ad hoc mode, send and receive messages in an infrastructure less environment has a number of applications, from providing state of the art military and emergency services *[28]* to easy and efficient communication in meetings and conferences to potential high-revenue entertainment applications.

MANET applications may demand reliable communication i.e. a level of guarantee that messages transmitted by a source will be received by the destination. Given, the extra-ordinary challenges of MANETs, the approach taken for reliable communication in MANETs should be different from that of traditional wired networks. From our research *[32]*, we concluded that a server less and asynchronous communication paradigm would address the inherent problems within MANETs.

However, designing and implementing applications for such a complex environment is a challenging task given the extra-ordinary challenges posed by the system. Middleware technologies have made application development easy and cost-effective in traditional wired networks. Hence, a middleware for MANETs can potentially have similar benefits.

Application development has been made easy in traditional networks by using middleware technologies like CORBA, COM, JAVA/RMI, Message Oriented Middleware (MOM) etc. *[3], [4]*. However, most of these are based on client-server (centralised) architecture and synchronous communication, which is not suitable for MANETs because of dynamic topology and the fact that disconnection is the norm rather than the exception. In addition, in traditional middleware technologies, it is advantageous to have the application developers and users to have transparency (access, location, failure etc.). However, since MANETs are bandwidth constrained networks and consist of resource-constrained devices (power, memory), it is critical for the middleware and application to be "system-aware" and "context-aware" *[5]* i.e. a device needs to be aware of its own resources and its neighbour's resources

(since they are used as routers) and the available bandwidth to be able to make informed and intelligent decisions.

Hence, our goal is to enable reliable communication in and ease application development for MANETs by implementing a middleware, which delivers messages with high probability (>80%).

MOM supports asynchronous communication and the Java Messaging Service (JMS) is a popular MOM for developing messaging applications in wired networks *[6]*. Hence, implementation of the JMS primitives for MANETs is attractive for three reasons:

- JMS semantics (asynchronous) are suitable for wireless communication since it enables nodes to receive messages even when the receiver is disconnected from the sender (switched off, out of range, low power etc.).
- JMS has been "tried and tested" and is popular in wired networks and because of ease of development could potentially prove to be popular for MANETs as well.
- The same APIs may be used for both wired, nomadic and MANETs. Hence, their integration and application development will become easier.

Given the challenges of MANETs, it is clear that transparency does not necessarily allow the optimisations needed in these kind of environments. Therefore, we have implemented a context-aware middleware by exploiting cross-layering solutions. We have implemented a routing algorithm, JMS Implementation for MANETs using Sociable nodes (JIMS), at the middleware level, which sits underneath the JMS APIs and implements the JMS specification for each API.

We have three main deliverables:

- An algorithm, which closely follows the JMS specification. The algorithm implements both Publish/Subscribe and Point-to-Point models of JMS.
- Implementation of the algorithm and simulation on a discrete event simulator, OmNet++ *[7]*. The algorithm was evaluated to determine its effectiveness on two mobility models, Random Waypoint and sociable-founded Group Mobility Model (GMM) and its reliance on different parameters (queue size, density of nodes, bandwidth, time to live, mobility pattern).
- A prototype, which uses the JMS APIs implemented using the algorithm to demonstrate the practical application of the Publish/Subscribe model of the algorithm.

The algorithm maintains the abstraction and difference between the idea of topics and queues for Publish/Subscribe and Point-to-Point models to enable asynchronous and reliable communication. This idea is at the core of JMS and JIMS.

The simulation results demonstrate remarkable performance even for a challenging mobility model like RWP. Our work is highly significant as there is other product/related research, which has implemented JMS semantics so closely.

The rest of the report is organised as follows:

2. *Background* – This chapter explains the key features of Enterprise Messaging, MOM and detail the motivation for using MOM semantics for MANETs.

3. *Reliability* – This chapter explains the importance of reliability in MANETs

4. *Introduction to JIMS* – This chapter explains the idea of JIMS

5. *JIMS* – This chapter explains the JIMS algorithm in detail

6. *System Evaluation* – This chapter presents, explains and analyses the simulation results

7. *Prototype* – This chapter details the design and implementation of the prototype

8. *Project Management* – This chapter explains the project management and implementation methodologies, team structure and roles and the project milestones

2. Background

In this chapter, we explain the key points regarding Enterprise Messaging, MOM and JMS. Further we also explain in detail the reasons for adapting this technology in MANETs.

2.1 Enterprise Messaging *[23]*

Messaging is a system of **asynchronous** requests, reports, or events that are used by enterprise applications. Messages are sent/published by a sending client (sender/publisher) to a destination (which may not be the endpoint) and retrieved from here (destination) by the receiving client (receiver/subscriber). Messaging technology has proved successful and is very popular in enterprise applications (banking, dissemination of stock information, initiation or termination of various utilities etc). Figure 1 below illustrates this process:

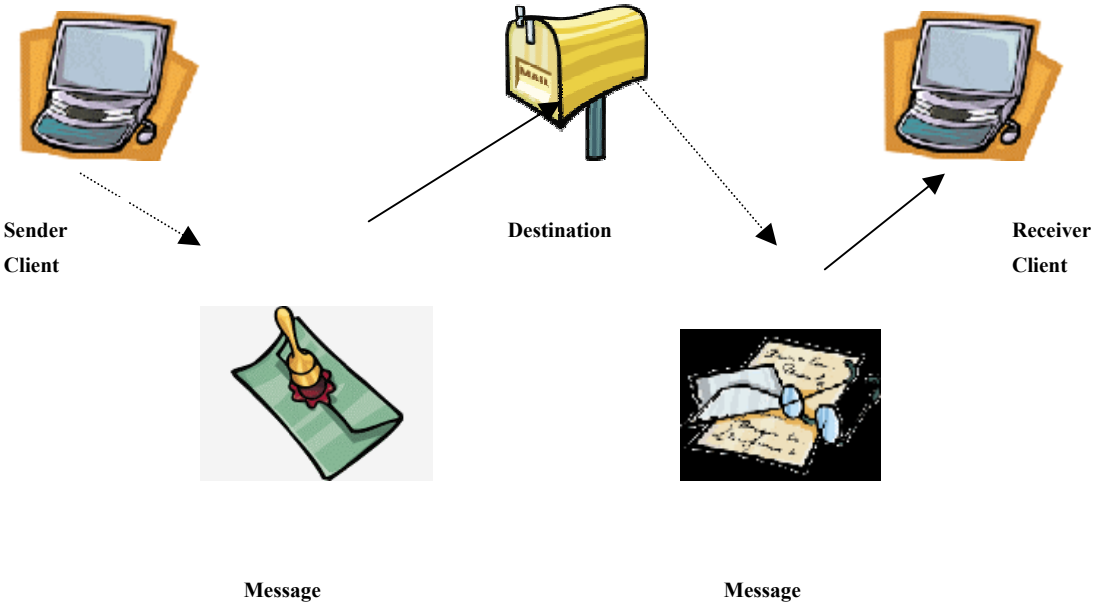

Figure 1 – Enterprise Messaging

For example, suppose a distributed component sends out stock information at regular intervals to remotely located distributed components. The sender client sends (posts) the information with a designated "destination", which is a messaging server. The server then distributes it to the interested remote receiver clients. This system has two advantages:

- The sender client does not need to monitor the current status (connected or disconnected) of the receiver client.
- The sender client does not need to monitor failures while sending information to potentially multiple receiver clients.

Hence, the server decouples the sender and the receiver clients and plays a key role in distributing the messages to all the receivers reliably.

2.2 Motivation for providing Message Oriented Middleware for MANETs

The motivation for deploying and using middleware technologies for application development is primarily to satisfy non-functional requirements such as reliability of message delivery, scalability, openness, heterogeneity, resource sharing and fault tolerance. These requirements are magnified in MANETs. Hence, a middleware for MANETs will prove to be extremely useful for the design and implementation of MANET applications.

MOM acts as an intermediary, and provides a common reliable way for programs to create, send, receive, and read messages in any distributed enterprise system. The notion of a "destination" is central to MOM. Messaging clients send messages to the MOM, which in turn routes these messages to the appropriate receiver. Figure 2 below illustrates this:

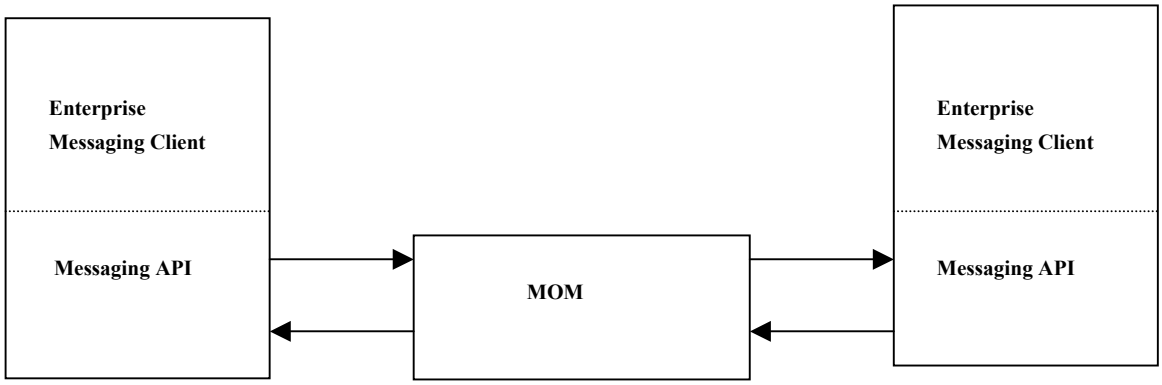

Figure 2 – Message Oriented Middleware

The most common messaging models are:

- Publish/Subcribe Messaging
- Point-to-Point Messaging
- Request-Reply Messaging

Not all providers implement all models of messaging.

The most important features of MOM with respect to MANETs are as follows:

- **Guaranteed Message Delivery** ***[23]***

  The receiver may be disconnected/offline when the message is sent/published. MOM guarantees that messages will be delivered to, as soon as network connection is established with the receiver.

This feature is extremely useful in MANETs, as receiving nodes may be "disconnected" due to a number of reasons *[29]*. They can be out of range of the sender, switched off, may lack sufficient resources etc. In such a scenario, being able to receive messages when conditions are more ideal is very attractive.

- **Asynchronous Communication *[23]***

The sender can continue with other tasks as soon as it sends/publishes the message, without waiting for a response from the receiver.

This is another feature important in MANETs. Since, nodes in a MANET may act as routers *[30]*, the sender can send/publish the message to an intermediate node i.e. destination/message server, which in turn may directly or indirectly (through other intermediate nodes) deliver the message to the final recipient. Once, the message is sent/published the sender client can continue to carry out other tasks uninterrupted without waiting for a response. This is important as the location of the receiver cannot be accurately determined and the sender cannot be stopped waiting for a reply forever. This not only increases the reliability of the system but also helps in hiding the latency experienced in synchronous system.

The advantages of MOM with respect to MANETs are as follows:

- **Reliability**

MOMs support different levels of message priority and corresponding message delivery guarantee. MOMs also support transactions and provide once and only once message delivery guarantee.

This idea is critical to the aim of our project, to enable reliable communication in MANETs. Given the dynamic environment of MANETs *[30]*, it is impossible to guarantee 100% delivery of all messages. If messages can be differentiated according to priority, then the system will endeavour harder to deliver the most important messages efficiently rather than focussing on the less important messages. This reflects real life scenarios and the postal service, where guarantee and efficiency of message delivery depends on its priority.

- **Easy and Flexible of application development**

Developing applications that span multiple operating systems and networking protocols/interfaces is made easier because the MOM forms a layer of abstraction. The application developer uses the APIs provided by the MOM to develop applications that extend transparently across diverse platforms and networks. This increases the flexibility of the application architecture and ease of messaging between various components *[23]*.

This is a very important advantage where MANETs are concerned as devices comprising a MANET may range from a laptop to a mobile phone. Wireless networks may range from GSM to 3G to Bluetooth technologies. Application development for such scenarios can be made much easier if the developer does not need to worry about which device/platform or network technology they are writing the application for.

- **Easy modification of distributed systems**

Since messaging clients operate independently of, and asynchronously from, each other, they can change without recompiling the application itself. Since all communication is through messages, which are routed by the MOM server, new applications may be added to the system without recompiling, re-linking or even stopping and restarting any current applications. This greatly reduces the complexity of distributed system design and implementation *[23]*.

This flexibility and ease very important in MANET applications as the number of nodes and hence applications is dynamic. It should not be necessary to recompile the applications each time the set of nodes in a particular MANET changes.

- **Location Independence**

Since message senders and receivers are decoupled by the "destination", they can be migrated from machine to another even at runtime without affecting the reliability of message delivery. This also enables continuous uninterrupted operation *[23]*.

Given the nature of MANETs, the location of the sender and receiver clients can never be guaranteed. Hence, enabling message delivery irrespective of the location of the senders and receivers makes the system robust and flexible.

- **Scalability**

If several clients generate requests, then the system may be scaled by adding more servers that serve a particular "destination", where the requests are posted *[23]*.

Applications designed for MANETs, should perform well with any number of users (nodes). Hence, scalability is an important requirement. MOMs allow the system to scale while maintaining the reliability of message delivery.

- **Event-driven Systems**

Event driven distributed systems *[8]* are increasingly becoming popular. Since events are essentially a form of messages, MOMs explicitly support the development of event driven distributed applications *[23]*.

MANETs will be required to support most of the traditional applications. Hence, the design and implementation of event-driven distributed applications will be easy through MOMs.

- **Simplicity**

Developers are comfortable with the simple idea of a "destination", where senders send and receivers receive messages from. Many applications for wired networks demonstrate this and hence prove that MOMs are not only popular with businesses but also with developers. Hence, developing similar applications for MANETs will not be perceived as a tough task.

- **Configurable Quality of Service**

  100% reliability is not always required. Sometimes configurable reliability is preferred because of the cost to speed, resilience, latency and availability involved in achieving 100% reliability *[23]*.

  This is useful in MANETs, where resources like memory come at a premium. Messages may be retained or deleted according to priority to free up resources.

- **MOM libraries can be very lightweight**

  This is an important consideration for memory constrained mobile devices *[31]*. Libraries, which considerable amount of memory are not feasible for mobile devices.

2.3 Overview of the Java Messaging Service (JMS) *[9]*

JMS is an API for accessing messaging systems. The JMS specification defines an interface and is vendor neutral but does not itself define or dictate the implementation. The JMS specification provides Java pplications with a common way to create, send, receive and read an enterprise messaging system's messages.

The primary features of JMS are as follows:

- JMS defines two messaging models:
  - Publish/subscribe
  - Point-to-Point

Both these models are defined by separate interfaces so that a provider does not have to support both.

- JMS uses connection factories are to create connections to a specific provider.
- JMS defines the concept of a "topic" for Publish/Subscribe messaging and a "queue" for Point-to-Point messaging as the "destination".
- JMS provides support for distributed transactions.

The two messaging models are described in the following paragraphs.

2.3.1 Publish/Subscribe Messaging [23]

The "destination" in this paradigm is called the **topic**. This domain allows for **asynchronous** message delivery. Publish/Subscribe messaging is used when multiple applications/components need to receive the same messages. Applications/components use this model to notify each other about a particular event. A client publishes an event to a topic; multiple clients are informed of the event by subscribing. There may be multiple publishers and subscribers, publishing and subscribing to the same topic. Figure 3 below illustrates the model:

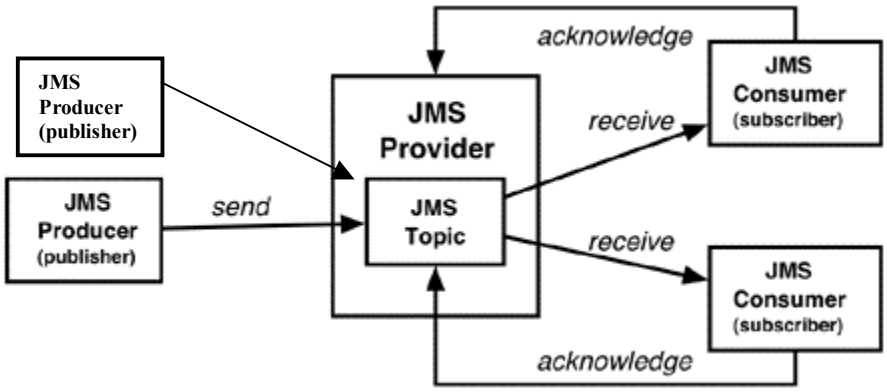

Figure 3 – Publish Subscribe Messaging Model

In the above example, there is one topic, four clients, of which, two are publishers and two are subscribers, which publish and subscribe to the same topic. The interaction steps are as follows:

- The publishers create a message and publish it to the topic.
- The subscribers subscribe to and register an instance of a callback (a message handler invoked by the server) with the topic.
- The messaging system delivers the message to all its subscribers by invoking the respective callbacks.
- Once all the subscribers have been notified, and the appropriate acknowledgements received, the message is removed from the topic.

An example of the use of publish-subscribe model and topics is that used in disseminating share price information on a trade floor. In this case a central entity, such as for example the London Stock Exchange (LSE) would publish the latest share prices onto a topic called, for example, 'share-price'. All the traders on the trade floor would then subscribe to the 'share-price' topic and receive all the latest share prices as they become available. This nicely illustrates the advantages of anonymity and scalability of messaging systems, as the LSE does not need to know who the traders are in order to inform them of the share price, and this setup easily allows for a large number of traders to receive the information simultaneously.

This one-to-many semantics closely matches the one-to-many multicast communication semantics in networks. Hence, it is a concept well understood by both application developers and network specialists. This will make application design and implementation easier for MANETs.

2.3.1.1 *Durable Subscriptions [23]*

When a subscriber requests a durable subscription, the JMS server will "save" in its persistent store any messages intended for the subscriber during periods when the subscriber is inactive. A message is only deleted from the off-line storage when it has been delivered to all durable subscribers or after the expiration time of the message. This is really useful for a MANET, where there may be nodes, which are "disconnected" from the publisher.

2.3.2 *Point-to-Point Messaging [23]*

The destination in this paradigm is called a queue. When one process needs to send a message to one other process, Point-to-Point messaging can be used. However, this may not be a one-way relationship. A messaging system client may only send messages, receive messages or send and receive messages. At the same time, another client may also send and/or receive messages. In the simplest case, one client is the sender and the other is the receiver of the message. The main issue in Point-to-Point messaging is that, even though there may be multiple senders of messages, there is ultimately only a single receiver for each message. This model is illustrated in the following figure:

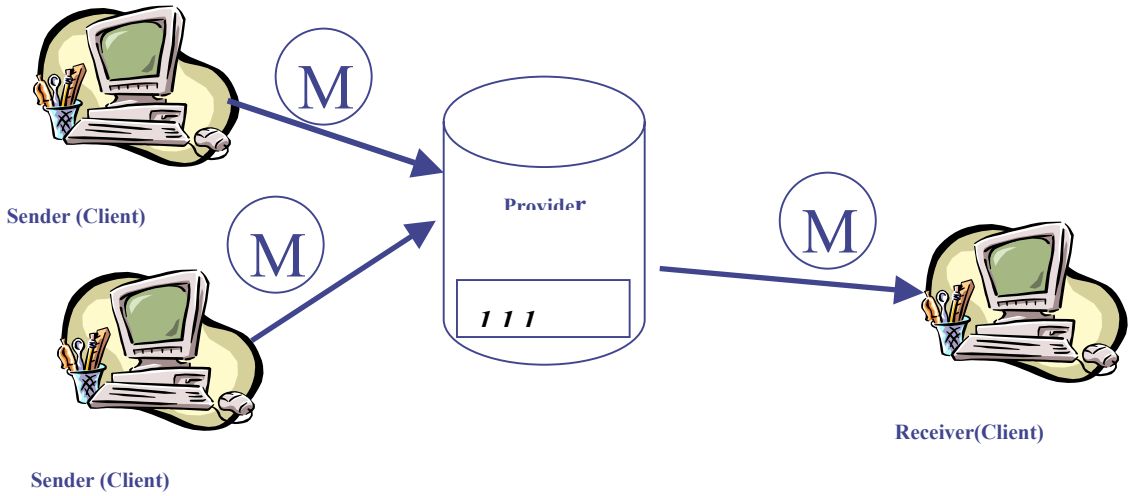

Figure 4 – Point to Point Messaging Model

In the example shown above, there are three clients, two senders and one receiver and a single queue. The steps of interaction is as follows:

- The sender client creates a message and sends it to the queue, where it is held.
- The receiver checks the queue for the message.
- The message is returned to the receiver client.

Taking the trading floor example further, an example demonstrating the Point-to-Point model would be that of processing buy and sell orders from the same traders as mentioned earlier. In this scenario, the LSE could have a number of machines capable of processing buy and sell orders, and have these machines receiving messages from an 'orders' queue. In order for a trader to buy or sell shares, the trader would simply send a message to the 'orders' queue,

which would be received and processed by one of the processing machines for the order to be executed.

Hence, the Publish/Subscribe messaging model is designed for one-to-many messaging operations and Point-to-Point messaging model is designed for one-to-one messaging.

3. Reliability

This chapter explains the importance of reliability in MANETs to emphasise the importance of our work. Figure 5 gives a glimpse of the application of MANETs.

3.1 Importance of Reliability in MANET

The applications of MANETs range from small static networks to large, highly mobile networks. The deployment of MANETs has several advantages:

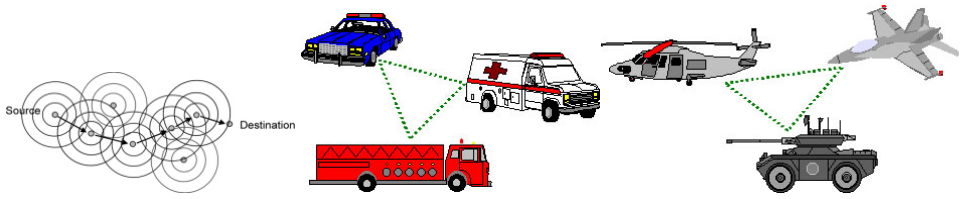

Figure 5 – MANET Application

- Rapid deployment – MANETs may be deployed anytime anywhere and enable instant communication whenever required
- Cheap – MANETs are infrastructure less networks and hence cheap
- Easy deployment – Currently, 802.11b and 802.11g technologies allow devices to form MANETs
- State of the art – As businesses are increasingly relying on mobile working, MANETs can make information available on the finger tips a reality

However, reliability is one of the key issues with MANETs.

Consider the following scenario. Jim is riding the subway and is viewing stock quotes on his PDA. He decides he wants to buy 100 shares of eCommWare Corporation. However, there is no connectivity on the subway to the network and hence the base station. Jenny is sitting close to Jim and is also using her PDA. She's going to get off the subway before Jim and hence get reconnected to the network. In order to carry out the transaction Jim has two options, first, get out of the subway, get reconnected and post the transaction himself or second, form a MANET with Jenny to "connect" with her PDA, pass the message for the transaction to Jenny's PDA so that it can be posted once she's out, which is potentially quicker. If Jim chooses the second option, he has to feel **confident of the success of his transaction** even without current, reliable network coverage.

Consider another example. Building Pleasant Stay is on fire. The fire brigade and rescue team is there and carrying out emergency operations. Each rescue team member carries a mobile device, which is capable of forming a MANET. Hence, all the members are quickly

"connected" to each other. Suddenly, someone realises that there's a child stranded on the second floor. A message is sent by one of the members. If the child is to be saved it is important that this message is delivered "**reliably**" to as many members of the rescue team as possible quickly.

Extending the above example further, a device on the fire brigade can **contact** a device on an ambulance, either directly (one-hop) or via several devices if one is fitted in every vehicle (multi-hop) it passes by. Immediate medical help in this situation may prove to be invaluable and save lives. However, **reliability of message delivery** from the fire brigade to the ambulance is key to the success of this operation.

A similar situation may be sketched for a battlefield. The ease of rapid deployment of independent mobile users without any infrastructure cannot be undermined. However, one-hop communication is not always feasible. In an infrastructure less, wireless, multi-hop scenario, it is important to have a certain **level of confidence** that a message sent would be received.

In addition to the applications listed, there are many other applications for MANETs such as sensor networks, instant conferencing between colleagues who meet outside the office and have mobile devices and Personal Area Network where PDAs, mobile phones and other electronic gadgets communicate using a technology such as Bluetooth. There are several applications of MANETs, however, reliability is one of the most important criteria in order for MANET technology to be successful.

4. Introduction to JIMS

This chapter explains the assumptions we made, challenges posed by MANETs leading to functional and non-functional requirements of the algorithm (JIMS).

4.1 Assumptions

- All hosts use 802.11b or 802.11g protocol and hence can establish an ad hoc connection.
- All hosts must use a common channel in order to establish connection. We propose a single channel to be used, which has a bandwidth of 11Mbps, which is shared between all hosts in the MANET.
- A reliable link layer is available with issues like hidden and exposed terminal are taken care of.
- We assume a secure environment and have not dealt with any security issues (authentication, denial of service etc.)
- All the nodes in the MANET are have enough power to send beacons and messages. Hence, power is not one of our primary concerns while designing the algorithm. However, the algorithm may be easily modified to accommodate power constrained mobile devices.

4.2 Challenges of Mobile Computing Environment

Mobile devices and computing have different features from nodes on wired networks and present a number of challenges, which makes a much more challenging platform as compared to wired, non-mobile devices. The two most important differences are:

- **Limited Connectivity**

  Mobile devices are not necessarily connected to the same network all the time

- **Resource Constraints**

  Mobile devices often have less computational power, available memory, available disk space, limited display capabilities and depend on battery power, introducing constraints not present in wired, non-mobile systems.

4.3 Functional Requirements

The JMS Implementation for MANETs using Sociable nodes (JIMS) algorithm is a routing algorithm and has three primary goals:

- The algorithm should achieve a reasonable message ratio (number of messages received/number of messages sent), >70%, for all types of nodes in all network topologies irrespective of the population density.
- The algorithm should implement the JMS semantics of both Publish/Subscribe and Point-to-Point models.

- The algorithm should be implemented in such a way that it may be used to build a middleware, which implements the JMS APIs.

4.4 Non-functional Requirements

In addition to the functional requirements, the algorithm needs to satisfy several non-functional requirements in order to cope with the extra-ordinary challenges posed by MANETs. These are:

- **Scalability**

  MANETs are "ad hoc" and dynamic. Nodes may enter or leave the network at any time. Hence, it is important for the algorithm to cope with variable number of nodes and ensure reliability with any number of nodes.

- **Heterogeneity**

  As mentioned earlier, heterogeneity (devices, wireless technologies, network interfaces) is integral to MANETs as MANETs is about connecting any two devices using any radio technology. So we may connect a digital watch with our PDA to see the next item on the to-do list or we may choose to connect our PDA to a presenter's laptop to extract the presentations. Hence, the algorithm should be able to maintain reliable message delivery irrespective of the device, wireless technology or network interfaces.

- **Intermittent connectivity and Fault Tolerance**

  Mobile devices may be switched off or may be out of range of the sender. The algorithm should ensure that messages are delivered irrespective of whether the receiver is connected or not currently.

- **Low Resources**

  In a MANET, nodes are not only senders and receivers, they are also relays (intermediate nodes) i.e. they route the message from the sender to the receiver. However, these nodes may not have enough resources (power, memory) to act as relays. The algorithm needs to cope with such circumstances and ensure message delivery.

- **Low Capacity Links**

  MANETs are characterised by low capacity channels. It is a challenge for the algorithm to maintain the survivability of the network in order to ensure reliability of message delivery.

- **Distributed Control**

  Again, MANETs are "ad-hoc" and dynamic. Hence, it is not possible to have a central point of control as this would lead to an inflexible architecture and central point of failure, which can prove to be disastrous in MANETs. The algorithm should enable distribution of control in such a way that the reliability of message delivery is not compromised.

4.5 Overview of JMS Implementation for MANETs using Sociable nodes (JIMS)

The details of the JIMS algorithm/protocol are explained in the next few sections. The JIMS algorithm carries out message routing, when JMS APIs are called. The application is developed as it would be for traditional networks and uses the standard JMS APIs. As APIs are called in the application, step/s of the protocol related to the particular API is executed. The following diagram illustrates the flow:

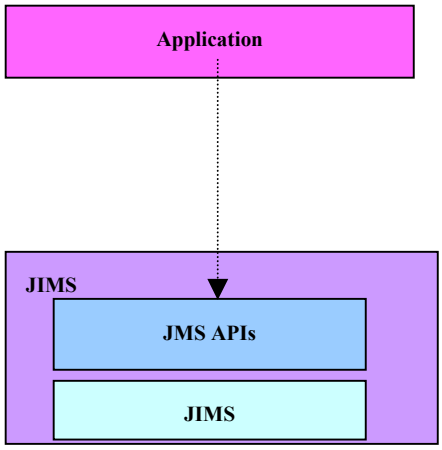

Figure 6 – JMS Implementation of JIMS

From Figure 6, we see that the application calls the JMS APIs, which triggers the execution of the JIMS protocol. The outer (mauve box) box engulfing the JMS APIs indicates that there are parts of the algorithm, which are not triggered by the application. For example, relaying of messages between intermediate nodes. This does not have to be specified by the application. The application only "publishes" a message, the mechanisms used to deliver the message to the destination are transparent to the user and the developer of the application.

JIMS preserves the idea of topics and queues as in JMS. Whilst it is unacceptable to have a centralised, single physical topic/queue for a MANET, we have ensured a single logical topic for each topic published/subscribed and a single logical queue for each Point-to-Point message.

4.6 Node Profiles

Mobile devices have a range of battery power, memory and disc space, with devices like a digital watch and a mobile phone having less power and memory than a laptop. It is also possible in a MANET for certain nodes to "meet" more nodes compared to other nodes. These nodes are more "**sociable**" than others. Hence, we have designed profiles for devices, which reflect their available resources and sociability value. These profiles are aligned to JMS "publishers" and "subscribers".

In traditional middleware technologies designed for wired networks, most of the context and system information (location, bandwidth etc) is hidden from both the developer and user of the application. This is because wired and nomadic networks have much higher system resources and bandwidth as compared to MANETs. Also, disconnections are rare and far apart. However, in order to address the extra-ordinary challenges posed by MANETs, it is important that the middleware be context and system aware in order to make informed decisions like whether or not to send to particular node and whether or not to receive messages from other nodes and adapt to the fast changing environment.

Therefore, to make best use of context information like available resources within a node, ascertain the capabilities of the neighbours of the node, we have taken three parameters into consideration – power, memory and sociability index. For each of these parameters we have established a threshold – power threshold, memory threshold and sociability threshold. Each of these parameters contribute to determine the node's profile.

Hence, each node is "aware" of its resources, determines its profile based on resource information and indicates this in its beacon. In all the profiles we assume that the device has enough power to send beacons and messages. Each node publishes or subscribes to one or multiple topics. The three profiles are explained in the next few paragraphs.

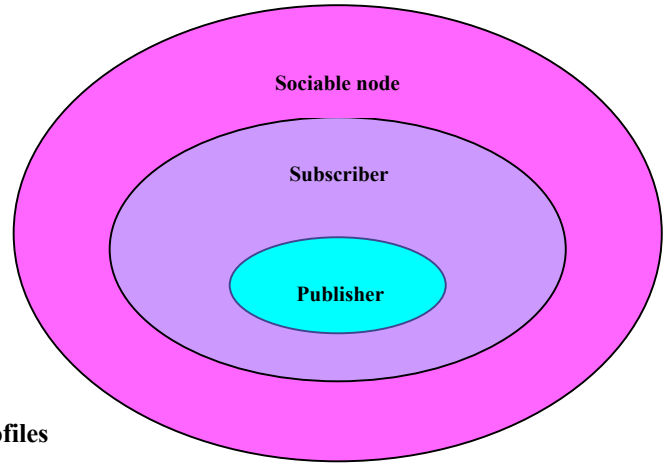

Figure 7 – Node profiles

4.6.1 Publisher

A publisher node has the least resources and sociability value. A node is a publisher if:

- Available power > power threshold

A publisher is capable of:

- Sending/Receiving beacon
- Creating and sending/publishing a message of a particular topic to a subscriber of the same topic or a sociable node
- Relaying saved messages in the queue on receiving meta-data from a subscriber or sociable node.

Note that publishers are not capable of receiving any messages.

4.6.2 Subscriber

A subscriber has more resources than a publisher but still not enough sociability value. A node is a subscriber if:

- Available power > power threshold
- Available memory > memory threshold

A subscriber is capable of:

- Sending/Receiving beacon
- Creating and sending/publishing a message of a particular topic to a subscriber of the same topic or a sociable node
- Creating a descriptor and sending meta-data
- Receiving messages of the topic **it is interested in** from any other node. For example, if a subscriber node is interested in "stock quotes", then it can receive messages from:
  - A publisher publishing stock quotes
  - Another subscriber carrying stock quotes
  - Sociable node carrying stock quotes

  Messages may be received in response to a beacon or meta-data.
- Relaying saved messages in the queue on receiving meta-data either from a subscriber or sociable node.

Hence, a subscriber in addition to publishing can also receive messages of one or more topics **it is interested in**. Therefore, a subscriber node is not only a subscriber but also a **publisher** (can send messages of a particular topic) and a **proxy publisher** (relay messages to other subscribers interested in the same topic).

4.6.3 Sociable node

A sociable node has resources and also meets the sociability criteria. A node is sociable if:

- Available power > power threshold
- Available memory > memory threshold
- Sociability index > sociability threshold

A sociable node is capable of:

- Sending/Receiving beacon

- Creating and sending/publishing a message of a particular topic to a subscriber of the same topic or a sociable node
- Creating a descriptor and sending meta-data
- Receiving messages of **any topic from any node**. Messages may be received in response to a beacon or meta-data.
- Relaying saved messages in the queue on receiving meta-data either from a subscriber or sociable node

Hence, a sociable node is capable of runs more services in addition to those run by the subscriber.

The idea of a sociable node is central to our algorithm and is unique in that we not only leverage the resources but also the mobility pattern of the node. Since a sociable node "meets" a lot of nodes and has high resources, it can be used to carry any topic as it can potentially meet subscribers of any topic and hence can relay messages of the topic that the subscriber is interested in.

A sociable node could be thought of as a mobile server. It is a **publisher** (can send messages of a particular topic), **proxy publisher** (relay messages to subscribers), **proxy subscriber** (saves messages of any topic received from any publisher) and a **subscriber** of a particular topic/s. Hence, it can serve any publisher and potentially any subscriber (depending on whether or not it is carrying messages of its interest). Note that a sociable node can degrade to a subscriber, which in turn can degrade to a publisher depending on the available resources and sociability value.

In order for the idea of sociable nodes to be successful, it was critical that a node should not only be aware of its resources but also be able to measure its sociability value. One solution was to administratively define sociable nodes. The other solution was to determine a node's sociability based on historical information about how many nodes it has met at a given time since the start. The latter was more difficult but would allow flexibility i.e. a sociable node does not necessarily remain a sociable node and can change its profile according to its resources and sociability value. Given the dynamic nature of MANETs, it is important that the design of the algorithm allows changes as and when required. Hence, we chose the second alternative. In addition to taking historical information into consideration, we had to ensure that the formula is not computationally intensive, as some devices do not have high computing power. We found that the predecessor to the final TCP/IP retransmit time out formula suited our requirement. The formula is as follows:

$S_i = (alpha)S_i + (1-(alpha))D$

Where:

S_i = Sociability index
Alpha = constant
D = number of **different** nodes a node meets in the current time unit.

To explain D further, each node maintains information about the nodes it meets for two consecutive time units, previous and current. It is possible that although a node has many neighbours, the neighbours always remain the same. Hence, this node remains in the same network "cloud". Such a node will not prove useful for relaying messages, which need to be sent from one cloud to another. Hence, it is not only important to determine the number but also identity of nodes that a node meets in different time units. Hence, while determining the sociability index, we take only those neighbours into account, which we did not meet in the previous time unit. This is a good indicator, which enables us to determine if a node would be able to relay messages and thus prove effective in increasing the delivery ratio. This has two advantages:

- **Effective use of bandwidth**

  Bandwidth is consumed each time a message is sent. We make prudent use of bandwidth and send messages only to those nodes, which are truly sociable and will prove effective in increasing the delivery ratio.

- **Intelligent use of resources**

  Consider the following scenario. If a node is interested in "stocks" but receives "sports" messages, in which it is not interested, then most of the memory (disc space) is full with sports messages with no place left for stocks. This could potentially decrease the delivery ratio due to lack of disc space. We could argue that these messages could potentially prove useful for node's neighbours. However, within a cloud a subscriber could potentially receive a particular message even if one other subscriber gets "infected" with the message (as the message hops from one subscriber to another), since it is just a few hops away. Since disc space is at a premium in these devices, we concluded that its use should be proportional to potential benefits. Hence, we decided that nodes could carry messages in which it is not interested in only if it has the potential to be part of different clouds and thus infect each of these clouds with the message.

The value chosen for the constant alpha was critical, as it plays the primary role in reflecting a node's true sociability value. D represents the current or the momentary sociability of the node. However, we are interested in the overall sociability. Hence, history information, which is given by (alpha)Si is more important than the current situation. If alpha has a small value then (alpha)Si will be a small value and (1-alpha)D will be a relatively larger value, which means we would give more importance to the current situation rather than historical information. Hence, we decided to have alpha = 0.6, which gives more importance to historical information, however, not completely biased towards it.

Each node has reserved disc space for messages. This space is divided into two. One part is for topic (Publish/Subscribe) messages and the other for queue (Point-to-Point) messages. However, although there is more than one physical topic/queue (disc space), there is only one logical topic/queue (disc space) per topic (i.e. sports etc.). Since all the nodes can potentially carry all the topics in the system, a subscriber can, as mentioned earlier, receive messages in three ways:

- Directly from a publisher, publishing the topic of its interest.
- Another subscriber
- Sociable node carrying messages of the topic it is interested in

This novel way of handling topics/queues has three important advantages:

- It enables **distribution of control**. No one node is responsible for ensuring that a particular message is received by all subscribers in the case of Publish/Subscribe topic messages and by one particular receiver in the case of Point-to-Point queue messages.
- It **increases the probability of message delivery** and hence delivery ratio. Since subscribers/receivers can receive messages from all the three kind of nodes, the chance that all subscribers receive all the messages increases.
- It also conforms to the JMS specification in two ways:
  - The **message is published/sent to a topic/queue and then pulled** by the subscriber/receiver.
  - Messages can be received **asynchronously** by subscribers/receivers, which are disconnected with the sender/publisher when the message is sent/published. This caters to both non-durable (subscriber receiving message directly from a publisher) and durable subscriptions (subscriber receiving messages from other subscribers or sociable nodes). However, unlike JMS, by default all subscriptions are durable. This is important, as the probability of being able to deliver the message synchronously to all the subscribers is very low.

Since, a particular node, given time, can attain any of the three profiles, it potentially carries messages (both Publish/Subscribe and Point-to-Point) that it is not interested in and those in which it is interested are potentially of interest to other nodes. Hence, any kind of node should be responsible enough to determine and relay messages that it has and another node does not.

4.7 The Beacon

Beacons are sent out at regular intervals by mobile devices either to connect to a new or remain connected to an existing base station. The beacon is a basic requirement of a mobile device, which indicates its presence in an area and carries vital information such as equipment identity number etc.

Keeping bandwidth constraints in a MANET in mind, we decided to minimise bandwidth consumption by control messages. Hence, we decided to leverage the power of the beacon to our advantage. We make our **middleware context aware, without increasing the number of control messages**. In addition to the existing information, the beacon would also carry profile and topic information. In this way we achieve two things:

- Each node can easily keep its neighbours informed of its presence.

- The profile indicates the resource and sociability value. Hence, the “beacon receiver” can make informed decision about whether or not to forward messages to the “beacon sender”.

5. JMS Implementation for MANET using Sociable nodes

5.1 Introduction

The JIMS algorithm is explained step by step in detail, with the benefits and conformance to JMS specification of each step. We will first explain the Publish/subscribe model and then the Point-to-Point model. Each of the steps will be explained with the aid of the example.

5.2 The Publish/Subscribe Model

We consider a simple MANET consisting of 10 nodes, some are publishers, some subscribers and others sociable nodes. Node id of each node is indicated inside the circle. The following are the five steps of the Publish/ Subscribe model.

5.2.1 Step 1

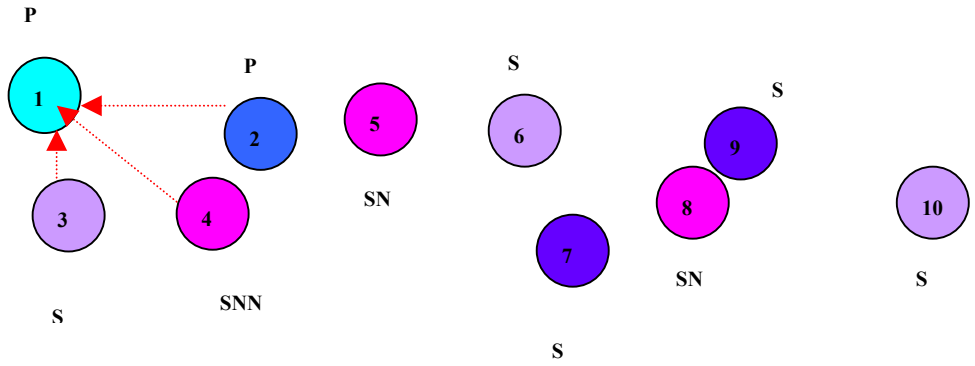

Figure 8 – Step 1

Nodes involved

Node id	*Profile*	*Topic*
1	publisher	sports
2	publisher	weather
3	subscriber	sports
4	sociable node	

Nodes 2, 3 and 4 send beacons, which are received by node 1. The beacon contains the following fields:

Node Id – The ID of the node

Node Type – The profile of the node

Topic Id – The Id of the topic name that the node is interested in as a subscriber or the topic name of the messages that it is publishing if it is a publisher

Each time a node receives a beacon it retrieves information from it and updates its table:

Table 1 – Node table

Unique Node ID	Time Indicator	Topic ID	Sociability Value	Node Type
2	01	W		100
3	01	S		110
4	01			111

Such a table is maintained by every node. This table helps the node to be context aware as it contains information about all the neighbouring nodes. Each row in the table is dedicated to one node with a unique node id. The time indicator (two bits) indicates whether that node was met in the current time unit (right bit) or in the previous time unit (left bit). "0" indicates "not met" and "1" indicates "met". This enables the node to calculate its sociability value as explained in section 4.6.3. Only the nodes met in the current time unit but not met in the previous time unit are taken into consideration while calculating the sociability value. The topic id indicates the topic in which the publisher/subscriber is interested. This information is necessary in order to decide to which nodes the message may be published. The fourth column indicates the Sociability Value of the node. This value is used to find the most sociable neighbour, in case of point-to-point messages. The Node type indicates the type of the beacon sender:

100 – Publisher

110 –Subscriber

111-Sociable node

This along with the topic information helps the node to decide which nodes the message should be sent to.

Once a node updates its table after receiving a beacon, it calculates its sociability value and hence determines its own profile based on this latest information and indicates this in its beacon.

5.2.2 Step 2

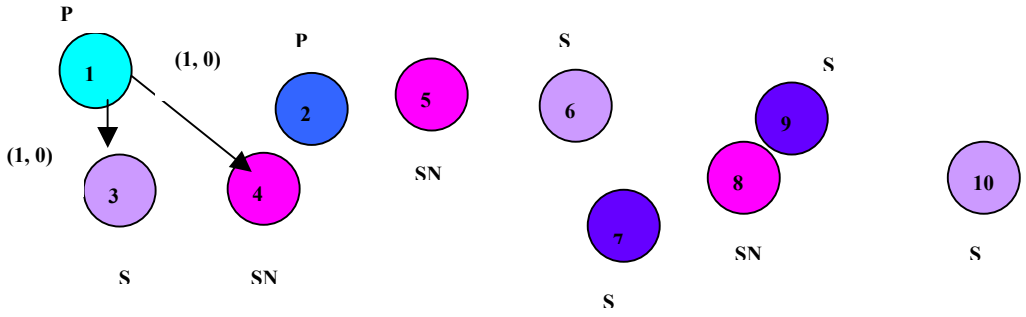

Figure 9 – Step 2

Nodes involved

Node id	*Profile*	*Topic*
1	publisher	sports
3	subscriber	sports
4	sociable node	

Node 1 has now updated its table and is now ready to publish messages. Consider at this point that it has something to publish. It "looks up" its table, finds node 3, which is a subscriber interested in the topic it is publishing (sports) and a node 4, which is a sociable node and node 2, which is a publisher. It publishes the message to nodes 3 and 4 only as node 2 being a publisher, which indicates it does not have enough memory to receive messages. Note that if a subscriber of another topic (other than sports), for example, node 7 was in proximity then node 1 would not have published the message to it, thus making intelligent use of resources and mobility pattern of node 7.

The data message contains the following fields:

- **Node Id**

  The Id of the sending node

- **Sequence Number**

  The sequence number of the message

- **Time to live**

  The *time* to live of the message specifies the lifetime of message in a node's topic/queue. This is explained in more detail in Step 3.

- **Topic Id**

Topic Id of the topic name of the message

- **Priority**

  Priority of the message

- **Hop count**

  The hop count indicates the number of hops a message can survive. It is decreased each time the message is passed on to another node. However, the difference lies in the fact that multiple nodes may be carrying the same message with different hop counts. For example, if a publisher publishes a message with hop count 5, its hop count reduces to 4 and to 3 when it is relayed again. However, in the meantime, the publisher may re-publish the same message to another subscriber/sociable node and the hop count of the same message will be 4 in this node. Whenever the hop count of a copy of a message becomes zero it is deleted from the topic/queue.

- **Payload**

  The actual message. JMS supports four kinds of message formats, text, object, stream and map messages. However, since our focus is on designing a reliable messaging algorithm for MANETs based on the basic primitives of JMS, we decided to handle only simple text messages. Hence, the payload is always plain text. However, please note that the same semantics may be applicable for all other kinds of messages.

Both time to live and hop count fields are to ensure that a message is not floating around in the network forever to avoid congestion and make room for new messages, which may otherwise be dropped.

Since the system may have several are publishing nodes, sequence numbers of the messages in the system is not unique. Each node keeps track only of its own sequence numbers. Hence, each message is identified by two fields – node id and sequence number. For example, the message published by node 1 in this scenario is (1,0) indication message published by node 1 and sequence number 0. The reason for this will become apparent in the next few steps.

5.2.3 Step 3

Research shows *[10]*, that in a MANET, where nodes act as routers or store and forward switches, the probability of message delivery increases as the number of nodes that are "infected" with the message increases. In our case, a publisher may meet many more subscribers and sociable nodes during the lifetime of the message. Hence, it may prove useful for the publisher to re-publish (relay) the messages as long as possible to maximise the "infection". Therefore, once the message is published the first time, the publisher buffers the message until its *time-to-live* period expires. Intermediate nodes also buffer the messages until either the time to live expires or hop count reduces to zero, whichever happens first. A node will indicate the *remaining time to live* each time it publishes/ relays the message. In our example, message (1, 0) is buffered by node 1 once it is published to nodes 3, 4. If a message is published with time to live = 20s at t =10 and is relayed again at t = 15, the remaining time to live = 15s.

Research *[11]* also shows that the longer the lifetime of the message, the greater the probability of its delivery. However, this means holding memory infinitely and not making way for new messages. This is an important factor to take into consideration with memory-constrained devices. Hence, each message has a finite lifetime and is deleted from the publisher's topic buffer once this period expires.

5.2.4 Step 4

Nodes 3 and 4 receive message (1, 0) from node 1. In keeping with the JMS specification of topics, when the message is received it is saved in the topic buffer, for use by the node itself or by other subscriber nodes. Hence, the topic buffer can either be in the node itself (unlike traditional networks) or can be found in other nodes. However, logically there is only one topic buffer for each topic. Just like in traditional networks, each subscriber "knows" where it can "look" for messages.

Queue Management

When a message is received, it should ideally be saved in the topic/queue. If the has available space, then there is no issue. However, the buffer may be full. In this scenario, the node needs to decide whether to discard the message or save it. The former option is easy and would involve no computation. However, it would violate the JMS specification, which treats each message according to its priority. Hence, in keeping with the JMS specification and to give each message a fair chance to be delivered, we have designed an intelligent and efficient queue management system.

If the buffer is full when a message is received, the message state of the newly arrived message is determined. If the message state of the newly arrived message is lower than the message with the minimum message state in the buffer, then it is discarded. However, if the message state of the newly arrived message is higher than the message state of the message with the minimum message state in the queue, then the message with the minimum message state is removed and replaced by the newly arrived message. The message state of a message is determined as follows:

*Message State = alpha*priority + (1-alpha)*(hop count or time to live / priority range)*

Where:

Alpha = constant

Priority = message priority

Hop count = hop count of the message

Priority range = number of priority levels

Each message has a priority and time to live/hop count as explained in step 2. Note that we have taken both these parameters into consideration while calculating the message state. While we need to keep the message priority in mind when saving or discarding the message, it is important to avoid unfairness to lower priority messages. The hop count of the message gives us an indication of the "spread" of the message. Since the hop count is decremented

each time the message is passed on, the lower the hop count, the larger the spread. Hence, it is "fairly" safe to discard messages with a low hop count, as we can be certain that it has infected a fair number of nodes, which can help to spread the message around further. However, if the message priority is high, then the system should endeavour harder (according to JMS specification) to deliver the message. Hence, we decided to take both these parameters into consideration when deciding whether or not to save/discard this message.

5.2.5 Step 5

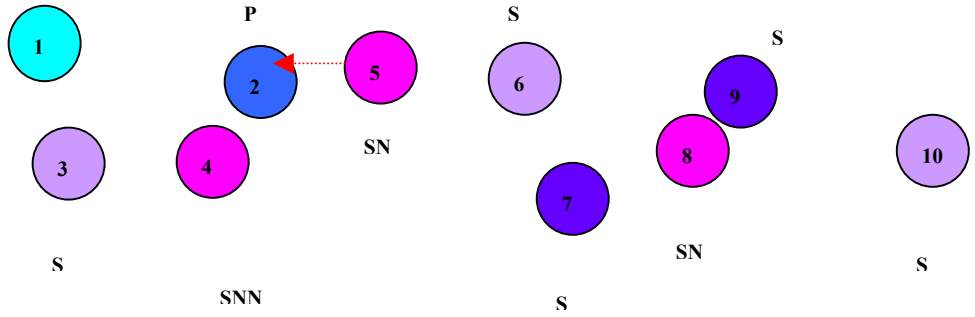

Figure 10 – Step 5

Nodes involved

Node id	*Profile*	*Topic*
5	sociable node	
2	publisher	weather

Consider node 5, which is a sociable node sends a beacon, which is received by node 2, which is a publisher. Node 2 updates its table. At this point a message becomes available to be published. Hence, message (2,0) is published to both nodes 2 and 5 as illustrated in the figure below:

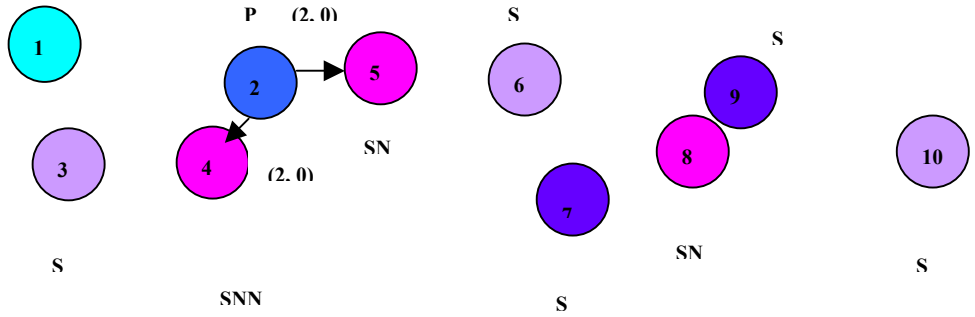

Figure 11 – Step 5

The nodes move, and the topology changes as shown below:

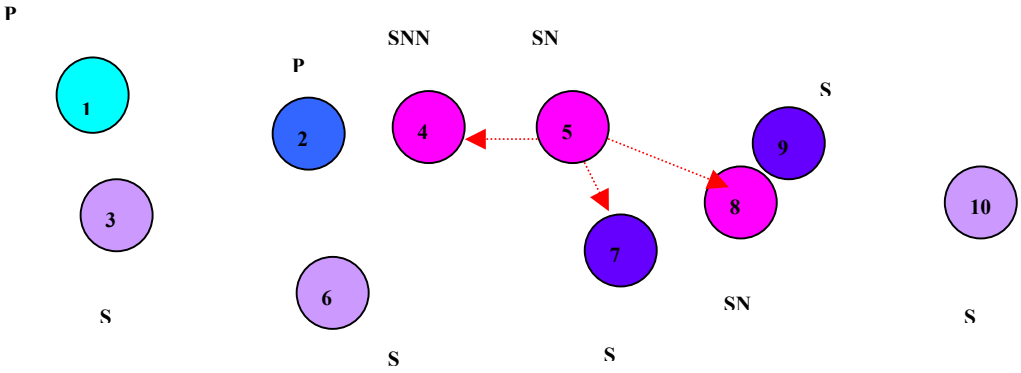

Figure 12 – Step 5

Nodes involved

Node id	*Profile*	*Topic*
4	sociable node	
5	sociable node	
7	subscriber	weather
8	sociable node	

Nodes 4,7 and 8 receive a beacon from node 5. Nodes 4,7 and 8 determine from the beacon that node 5 is a sociable node. Subscriber nodes are aware that a sociable node can carry messages of any topic. Hence, when a subscriber node like node 7 meets a sociable node like node 5, it should, according to the JMS specification, "look up" the topic queue of the sociable node and pull messages from it.

Anti Entropy

However, the subscriber may already have some of the messages that the sociable node is carrying. Hence, it may not need all the messages. Therefore, we can make intelligent use of bandwidth by transmitting only those messages, which are required and not received by node 7 *[12]*. The anti-entropy mechanism is designed to achieve this. When the subscriber, node 7, receives a beacon from sociable node, node 5, it responds by sending meta-data. The meta-data indicates which messages it has and which it does not. The sociable node in turn responds by determining which messages it (sociable node) has, which are required and not received by the subscriber and transmits those messages. This is illustrated in Figure 13:

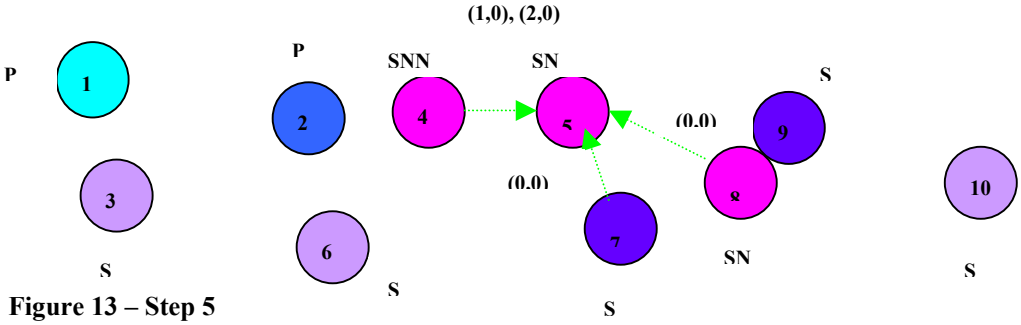

Figure 13 – Step 5

In our simple example, we have assumed that node 7 has not received any messages till now. Hence, it has to indicate this in its meta-data. On receiving meta-data from node 7, node 5 determines that it has one message (2, 0) that node 7 requires (topic = weather) and does not have. Once it determines this, it sends a copy of the message from its topic buffer. Figure 14 illustrates this:

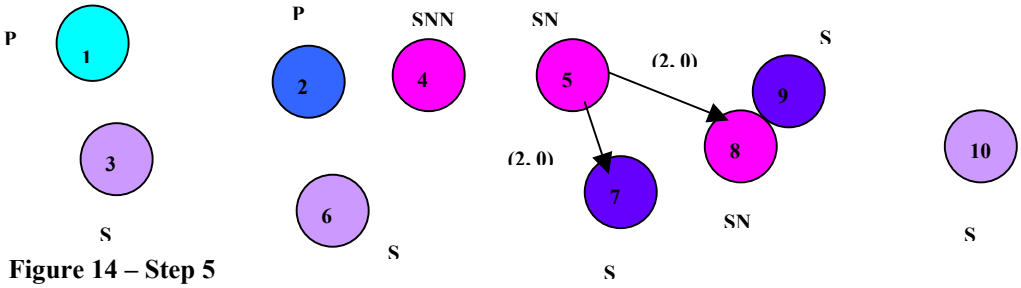

Figure 14 – Step 5

The meta-data contains the following fields:

Node Id – The Id of the node

Topic Id – The topic Id of the descriptor

Descriptor – This indicates the messages present in the buffer

The descriptor of the meta-data is the most important field. This field indicates, which messages a node has/does not have. The length of the descriptor is variable. The length of the descriptor is determined by:

Length of descriptor = number of topics * maximum node id message received from * range of sequence number of messages

We use the (node id, sequence number) pair to describe a message. A "block", of size equal to range sequence number of messages is reserved for each node a message is received from and for all nodes in between up to the maximum node id a message is received from. Such a block is constructed for each topic that the node carries messages. A received message is indicated by a "1" and a missing message is indicated by a "0". For example, if there is a single topic, sequence number of messages is 0-6, range of sequence number of messages = 7 and if a node has messages (1,0), (2,1), (4,2),

Length of descriptor =1 4*7 = 28*

Descriptor =

1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***

In our example, descriptor of node 7 will be:

0	***0***	***0***	***0***	***0***	***0***	***0***

When sociable node receives meta-data from a subscriber or sociable node, it needs to determine if:

- it has any message
- the message is of interest to the meta-data sender
- the meta-data sender already has the message that it has

Hence, it forms its own descriptor and computes (A.~B) (A *and* B *compliment*) for each message to decide which message needs to be sent. Where, A is the descriptor of the sociable node and B is the received descriptor. For example, if a sociable node has messages (1, 0), (2, 0), (3, 1), (4, 2),

*Length of descriptor = 4*7 = 28*

Descriptor =

1	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***

If this sociable node receives the meta-data described earlier the computation performed will be:

1	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***
1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Hence, the sociable determines that messages (2, 0) and (3, 1) satisfy all the conditions stated earlier and hence have to be sent to the meta-data sender. Condition 2 needs to be satisfied only if the meta-data sender is a subscriber i.e. the message topic should be the same as the topic in which the subscriber is interested. This further helps in making effective use of bandwidth and memory as we avoid sending just “any” message; instead we send messages, which are of interest to the subscriber.

In our example, Figure 14, descriptor of node 5 will be:

0	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***

Node 5 will determine that message (2, 0) satisfies all three conditions and hence send a copy of the message to node 7. Further, meta-data from node 4 indicates it already has message (2, 0). However, meta-data from node 8 indicates that its buffer is empty. Hence message (2,0) is sent to node 8 as well.

Either a subscriber or a sociable node may send meta-data to a publisher, subscriber or sociable node. Since a publisher is not capable of receiving messages owing to its memory constraints, it does not send meta-data. Since, it is the subscriber's responsibility to send meta-data, to get the messages it is missing, on receiving a beacon from another subscriber or sociable node, we argue that this is the "pull" mechanism as stated in the JMS specification. In effect, the subscriber sends meta-data to "check" and "pull" messages from another node's buffer (topic/queue).

Note that when a subscriber sends meta-data it indicates only the messages of the topic/s it is interested in rather than all the messages it has in its buffer. The subscriber can receive only messages of the topic/s it is interested in. Hence, it does not try to pull messages of other topics. This further reduces the size of the descriptor and in turn the size of the meta-data. On the other hand, a sociable node indicates all the messages in its buffer in the descriptor since it acts as a relay.

Sending meta-data instead of directly sending messages irrespective of their utility value will cause new messages to be dropped as a result of reduction in bandwidth. The network will get congested and will not survive for a long time. It can be argued that, at times, as demonstrated the descriptor can be very long as compared to the number of messages in the queue. However, each message is represented using only one bit, which is minimal compared to the size of one "unnecessary" message.

The subscriber/receiver receives the message from publisher/subscriber or sociable node in response to its beacon or meta-data. Therefore, we argue that although the message is sent to the subscriber, it is in fact a "pull" action since it is the subscriber's "responsibility" to send out these signals in order to receive messages.

This brings us to the end of the Publish/Subscribe model. In reality most of these steps may be taking place simultaneously in this model. However, we have tried to explain the basic functionality of the algorithm using this example.

Note that we have implemented and achieved reliable Publish/Subscribe (multicast in other words) messaging in MANETs using minimum control messages and state information. This helps in increasing the survivability of bandwidth constrained networks and enables devices with low computation power and memory to be part of the network. Unlike other *[10]*, *[13]* implementations, our algorithm implements and achieves reliability of message delivery by making intelligent use of resources and mobility patterns of individual nodes rather than using bandwidth expensive mechanisms like sending acknowledgements. Reliability is increased by increasing the "infection" of the network by the message rather than by adopting a complex mechanism of keeping time outs for acknowledgements and retransmitting the messages after the time out expires. It is interesting to note that continuous exchange of meta-data will mean that given time, eventually a node will have all the messages. We appreciate

the differences between traditional wired networks and MANETs and have designed novel ways to address the challenges posed by these complex environments rather than porting the same mechanisms that work well in wired networks. Simulation results show that given time 100% reliability may be achieved, i.e. 100% of the subscribers may receive 100% of the published messages. However, there is a fair trade off between latency and degree of reliability that can be achieved. Hence, delay insensitive applications, which demand a high level of reliability can benefit most from this protocol.

5.3 The Point-to-Point Model

According to the JMS specification, point-to-point messages may be sent by multiple senders but is ultimately received by one and only one receiver. However, the set of potential receivers may contain more than one receiver i.e. multiple receivers listening for messages at the same queue. It is crucial in e-commerce applications like the trading example mentioned in 5.3, that transactions are executed exactly once.

In our algorithm, more than one copy of each message is maintained in order to enable and ensure maximum delivery ratio by increasing the spread of the message. This suits publish/subscribe messages where there are multiple receivers of each message. However, in case of point-to-point messages, we have to ensure that one and only one receiver receives each message. Maintaining multiple copies would potentially increase the delivery ratio, but could also lead in violating the JMS specification by delivering one message to more than one receiver. This is because it is difficult to instantly inform every node of the delivery of the message so that all other copies may be deleted. Therefore, we decided to maintain only one copy of every point-to-point message in the network.

The Point-to-Point model is to be programmed as follows:

- The sender looks up a queue name.
- Connects to the queue using the queue name.
- The receiver also connects to the same queue using the same queue name. This is analogous to subscribing to a topic in the Publish/Subscribe model.
- The receiver listens for messages.

Most of the steps for delivering point-to-point messages remain the same as publish/subscribe. The difference between the Publish/Subscribe model and the Point-to-Point model are as follows:

- Instead of a topic buffer, the messages are delivered to a queue buffer.
- Point-to-Point messages are not saved in the sender's queue buffer once it is sent. This implies that the only copy of the message is either with receiver or a sociable node once it is sent. However, if the sender does not come in contact with a receiver or a sociable node within the time to live period, the message is discarded. This reflects reality, as certain messages like a piece of news or more importantly a stock quote loses its value if it is received after a certain period of time.

- A receiver may either receive a point-to-point message directly from the sender or from a sociable node. Unlike the Publish/Subscribe model, where a subscriber may receive a message from another subscriber, in the Point-to-Point model, one receiver cannot receive a message from another receiver. This is because the message needs to be processed just once.

- A receiver in the proximity of a sender is a favourable, however, potentially rare situation. Therefore, it is important that we adopt other means of delivering the message. Using the sociable node as a relay was the best solution. However, since we had to maintain only one copy of the message in the network, we had to increase the chances of message delivery as much as possible within this constraint. Hence, if a receiver is not in proximity, then the sender browses its table of nodes, compares the sociability value of all the sociable nodes in proximity and sends the message to the most sociable node. Further, a sociable node (1) could meet another sociable node (2) before it meets a/the receiver. In such a case, (1) may either relay the message to (2) or keep it with itself and relay it only when it meets a/the receiver. Again the decision should be based on which of the two options will increase the chances of message delivery. Therefore, (1) will relay the message to (2) if the sociability value of (2) is greater than (1), else (1) retains the message with itself. As soon as the message is relayed either to a subscriber or to another sociable node, it is deleted from the queue of the relaying node, thus ensuring only one copy of every point-to-point message.

- The queue buffer size and hop count are two important parameters, which affect the probability of delivery of a message. The former is always at a premium and hence some kind of queue management strategy has to be adopted. Hence, we decided to keep the queue management strategy the same as it is for publish/subscribe messages. However, hop count of a message is under our control i.e. we can decide its value. Hence, depending on the application, we can decide the hop count, i.e. a large hop count for a very important message or a small hop count for a less important one, one that can be discarded before delivery.

Apart from the differences stated above, everything remains the same as the Publish/subscribe model.

5.4 Message Delivery Modes

JMS supports two different modes of message delivery, *persistent* and *non-persistent*. Persistent messages should be delivered *once-and-only-once*. This means that the message should not be lost for any reason and should *never* be delivered more than once. Non-persistent messages should be delivered *at-most-once*. This means that the message may not be delivered eventually but it is *never* delivered more than once.

Given the nature of MANETs, handling persistent messages and guaranteeing once-and-only-once reliability is next to impossible. Hence, we concentrated on non-persistent messages and guarantee at-most-once reliability for point-to-point messages. However, future enhancements may be made in order to handle persistent messages. Nodes may somehow be informed of each other's past neighbours and a node carrying a point-to-point message to be

relayed may relay it to a node, which is most likely to meet the receiver regardless of its sociability value. Such historical information [14] has proved helpful in making decisions in MANETs. However, the fact that this means additional maintenance of state information cannot be ignored. This is an important factor to be taken into consideration when we are dealing with resource constrained environments and devices.

Please note that we have not implemented transaction semantics due to time constraints. However, the algorithm can be easily extended to accommodate transactions. The publisher can indicate a transaction by a control message, messages will need to be acknowledged and delivered within a time out period. The semantics to deliver messages in a transaction may be the same as that for point-to-point messages.

6. System Evaluation

6.1 Introduction

This chapter seeks to explain the evaluation methods and results of the algorithm. We have implemented and evaluated the algorithm using a discrete event simulator, OmNet++. Our primary goal is to achieve reliable message delivery in a MANET. Hence, we have tried to measure the probability of message delivery by varying and studying the affect of different parameters like population density, buffer size etc. We have presented an objective analysis of the algorithm and where appropriate tried to compare the results with other solutions. We begin by evaluating the Publish/Subscribe model and then proceed to the Point-to-Point model. Please not that the focus is more on the Publish/Subscribe model, as we believe that the semantics of the Point-to-Point model are a subset of the Publish/Subscribe model. Hence, the Point-to-Point model should follow the same trend as the Publish/Subscribe model.

6.2 OmNet ++

OmNet ++ is a discrete event simulator, which facilitates object oriented programming in C++. This is useful, as we wanted the code to be reusable for the prototype. This also makes it easy to model a network topology as it provides a high-level language compiler (NED) as the description language.

We used C++ to describe the active components of the model as concurrent processes where some code relies on the simulation class library provided. We used the GUI (Tkenv) for to visualize the movement of nodes for both the mobility models. This enabled us to observe the order of packet exchange and debug our program pictorially, which proved to be more efficient that the command line debugger. Once the code was thoroughly tested, we proceeded to collect simulation results using the command line interface (Cmdenv) to increase the simulation speed. OmNet++++ provides many primitives, including APIs, which enable easy collection of data and transfer to files with just few lines.

We used the seedtool program provided to select good seeds, specifically to generate random location for nodes. For example, suppose we need 8 seeds to determine the location of 8 nodes:

C:\OMNETPP\UTILS> seedtool g 1 10000000 8

(10000000 here means 10,000,000 values apart)

The program outputs 8 numbers that can be used as random number seeds:

1768507984

33648008

1082809519

703931312

1856610745

784675296

426676692

1100642647

6.3 Mobility Models

One of the main challenges in defining realistic mobility models for MANETs is that there is no realistic data available on ad hoc structure and mobility. One can only hope for a reasonable approximation based on current research and reasonable approximation. *[15]* gives a detailed evaluation of various mobility models. Our algorithm does not target any particular mobility model. Hence, we decided to evaluate our algorithm on two distinct mobility models, Random Way Point and Social-founded Group Mobility Model. Hence, this would enable us to make an objective analysis and evaluation of the algorithm. For each evaluation parameter we have compared the performance of the algorithm in the two mobility models.

6.3.1 Random Way Point (RWP)

The Random Way Point mobility model is designed to reflect entities in nature, which move about in completely unpredictable ways. Since MANETs enable communication and exchange of information anytime, anywhere, it is important that we take all possibilities into consideration. For example, we would like to share news with a fellow passenger in the bus/tube. Our choice of bus/tube is random depending on where we want to go.

RWP includes pause times between changes in direction and/or speed *[16]*. A node stays at one location for a period of time (i.e. a pause time) then chooses a new random destination in the simulation area and at a uniform distributed speed. It then travels towards the newly chosen destination at selected speed. Upon arrival, it will pause again before proceeding to the next destination [CampBoleDavi].

6.3.2 Group Mobility Model (GMM)

In the real world, mobile devices are usually carried by humans, hence, their movement is dictated by the human decisions and socialisation behaviour. Sociability founded mobility model *[17]* was developed to model the behaviour of individuals moving in groups and between groups which is most likely deployed in scenarios of disaster relief teams and military troops. Interestingly, most MANET applications are targeted for such scenarios. In this mobility model, social relationships with respect to colocation is implemented using a weighted graphs by defining the weights associated with each edge of the network to evaluate the strength of interaction between individuals. Social relationships are then used among individuals to define groups of hosts that move together in the simulated scenarios. At the same time, individuals will move within the clouds, which are defined by the geographic group to which they are associated. However they will decide to either to move between groups or to leave the group structure and to move completely independently. A node belonging to a group moves inside the corresponding group area towards a randomly generated goal. Moreover, clouds also move towards randomly chosen goals in a simulated space. A threshold is defined to determine so called "aloneness factor" whether a node will

remain within its group, to move to another group, or to escape outside all groups. If the factor is greater than threshold, a new goal is chosen outside the areas of any group and vice versa.

The *propagation model* used is free-space, which means that there is always one line-of-sight path between the transmitter and the recipient, and the communication range is represented as a circle around the transmitter. If the receiver is within the circle, it receives all the packets, otherwise it loses all of them *[26]*. Further analysis of the propagation model is outside the scope of this project. The antenna is omni-directional which means that it radiates or receives equally well in all directions

6.4 Hop Count Vs Buffer Size

We have given priority to those parameters that we think would have the highest impact on the system. Since the simulation area is restricted to 1000m x 1000m, ideally, it should only take a maximum of seven hops for a node at one end of the diagonal to relay the message to a recipient at the other end of the diagonal. Hence, we argue that the hop count of the message has less impact on the message ratio as compared to the buffer size because a non-persistent message can be discarded if the buffer of the next hop node is full. This means that a message may be dropped by its subscriber/receiver if its buffer is full. Therefore, non-persistent messages have unlimited hop count, however, the buffer size of each node is limited.

6.5 The Publish/Subscribe Model

The following sections explain the design, results and analyses of the Publish/Subscribe model.

6.5.1 Simulation Design

Table 2 indicates the simulation parameters.

Table 2 – Simulation Parameters

Mobility Pattern	**Group Mobility**	**Random Waypoint**
Number of hosts	8/16/32/64	8/16/32/64
Simulation area	1Km x 1Km	1Km x 1Km
Propagation model	free space	free space
Antenna type	omnidirectional	omnidirectional
Transmission range (radius)	200m	200m
Number of groups	5	-
Group area	200m x 200m	-
Node speed	1-3 m/s (randomly generated)	1 m/s
Group speed	1-2 m/s (randomly generated)	-
Number of replicates	15 (for each scenario)	15 (for each scenario)

Each replicate was set to run for 3600 seconds (1 hour) in order to obtain statistically reasonable results. Each parameter (population density, buffer size, latency etc) is evaluated for two kinds of messages, persistent and non-persistent. Persistent messages have unlimited hop count and time to

live and infinite buffer space. Non-persistent messages have unlimited hop count and time-to-live is set at 1200 seconds. Buffer size of each node is limited to 15 messages.

Memory constraint of the machine on which the simulation is run restricts the number of nodes and hence messages in the network. Hence, the real world cannot be completely simulated. However, we have endeavoured to set the parameters to closely reflect the real world. Hence, we evaluate the performance of the algorithm for various parameters, each parameter is analysed for 8, 16, 32 and 64 nodes, two types of messages and two mobility models. Since, results vary slightly for small increase in the population density, we have chosen a larger interval scale, 2^3 (8), 2^4 (16), 2^5 (32) and 2^6 (64).

According to the algorithm, all the nodes are capable of publishing. However, for the sake of uniformity and simplicity, for each run, we have two publishers send at most 45 messages, which are published every two seconds. Message priorities are generated randomly and are ranging from 1-5.

In the Publish/Subscribe model, there are multiple subscribers per topic. Hence, there are multiple messages per subscriber. Hence, we need to measure the percentage of subscribers that receive each message and the percentage of messages received by each subscriber. This will enable us to get a balanced view of the system both in terms of subscribers per message and messages per subscriber.

Hence, we define *message ratio* as the average percentage of messages received by each subscriber. We also define *delivery ratio* as the percentage of subscribers that receive each message. We evaluate how these values are affected by various parameters.

6.5.2 Data Analysis

First we evaluate the performance of the algorithm in terms of message ratio. Figure 15 shows the results in RWP and Figure 16 shows the results in GMM. Following are our observations:

- The message ratio increases as the population density increases in both the mobility models. Message delivery relies on the spread of the message i.e. the greater the number of nodes infected with the message, the greater the chances of delivery. As the population density increases, the number of nodes coming in contact with each other increases. Hence, the number of nodes getting infected with a particular message increases, which results in a higher message ratio.
- As expected the message ratio is higher for persistent messages. It is remarkable that even for a challenging mobility model like random way point (RWP), in which the movement of nodes is completely unpredictable, 80% of the messages are delivered under ideal conditions. In the group mobility model (GMM), almost 100% of the messages are delivered. We also see that percentage of messages delivered is higher in GMM (60%) than in RWP (>40%). Overall, the algorithm performs better with respect to GMM. As mentioned earlier, message delivery is largely dependent on the spread of the message. In the JIMS algorithm, sociable nodes are used to infect "clouds" of nodes with messages and once a cloud is infected, "epidemic" effect takes over. In GMM,

groups of nodes are analogous to clouds. Also, there are nodes, which move between clouds. These nodes, in the JIMS algorithm are the sociable nodes. Once a sociable node with a message infects a cloud, the epidemic diffusion of the message to the entire cloud becomes easy, as many of the subscribers may be just one-hop away from each other. However, in RWP, node mobility is more random compared to GMM. There are no clouds. Hence, message delivery largely depends on sociable nodes, which are limited in number. Therefore message ratio in RWP is lower than in GMM.

- The dramatic rise of message delivered from 16 to 32 nodes in GMM, which eventually flattens out at 64 nodes implies that 32 nodes population is an optimum density to achieve a high delivery ratio. This indicates that in >32 nodes population density, the epidemic effect plays a predominant role in spreading the message to a large number of nodes in a short span of time.

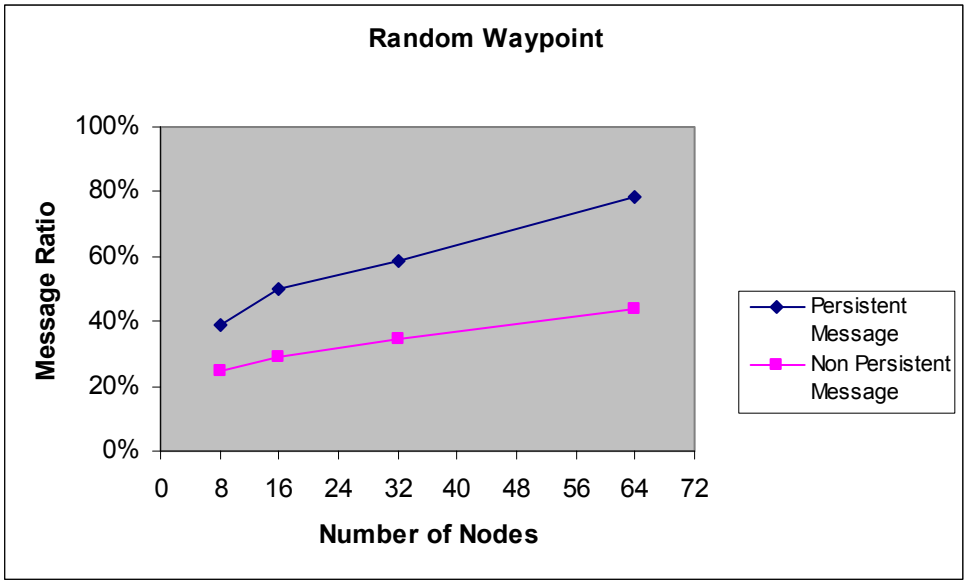

Figure 15 – Message ratios at Different Population Density using Random Waypoint Mobility

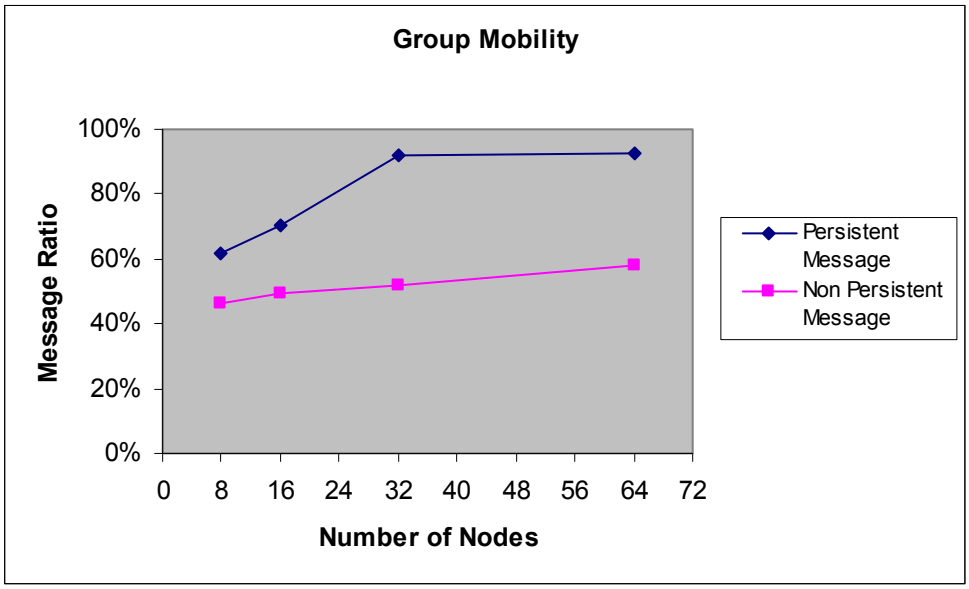

Figure 16 – Message ratios at Different Population Density using Group Mobility

It is also interesting to note that given time message delivery for RWP increases dramatically. Hence there is a fair trade off between reliability and performance. This is also proved in *[33]*. Reliability i.e. probability of message delivery increases with time, as the probability of subscribers coming in contact with a node carrying the message increases. Hence, applications that are delay insensitive but demand a high level of reliability can benefit most. This reflects the challenges for designing applications and protocols for a complex network environment such as a MANET identified in *[32]*. Also, in *[18]*, it is proved that a mobility model can increase throughput per source-destination of ad hoc wireless network i.e. delivery ratio and message ratio. Delay tolerance of applications can be exploited since mobility improves delivery ratio by allowing packets to be relayed from one node to another in 1 hop (higher probability). This fact is clearly reflected in the results shown in Figure 17:

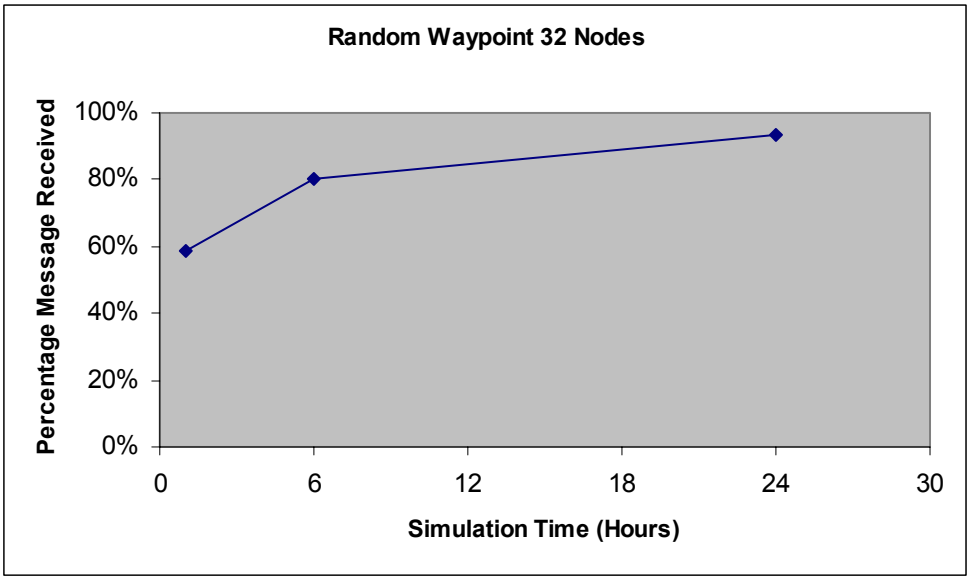

Figure 17 – Message ratios at Different Simulation Time for 32 Nodes

Next we go on to evaluate the affect of various buffer sizes of each node on the message ratio. Figure 18 shows how the percentage of messages delivered varies with the buffer size. From our previous simulations we concluded that population density of 32 nodes resulted in dramatic changes. Hence, we decided to evaluate the behaviour of the algorithm with 32 nodes. The simulation is carried out for non-persistent messages. Following are our observations:

- As before the algorithm performs much better with GMM as compared to RWP, maximum messages delivered for GMM is close to 100% while for RWP it is just over 40%. It is interesting to note that for GMM, a minimum of 40% of the messages are delivered even for a small buffer size of 10. A small increase in the buffer size i.e. increase of 10 units increases the message ratio by almost 20%. Further, irrespective of the time-to-live, almost 100% of the messages are delivered with an increased buffer size. Again, this behaviour is due to the fact that both sociable nodes and the epidemic effect aid message spread in GMM, whereas in RWP, it is mainly the sociable nodes doing the job. This means that in GMM, a high percentage of messages may be delivered even with a short lifetime, which implies high message ratio within a short span of time. Hence, we can utilise the buffer space better and make room for more messages, hence avoid congestion and increase the survivability of the network.
- RWP achieves a minimum of 20% and a maximum of 40%+. However, we argue that due to the mobility pattern of the nodes in this mobility model, message ratio is more a function of time rather than buffer size (as shown in Figure 17) as we do not see significant improvement from buffer size 10 to buffer size 50.

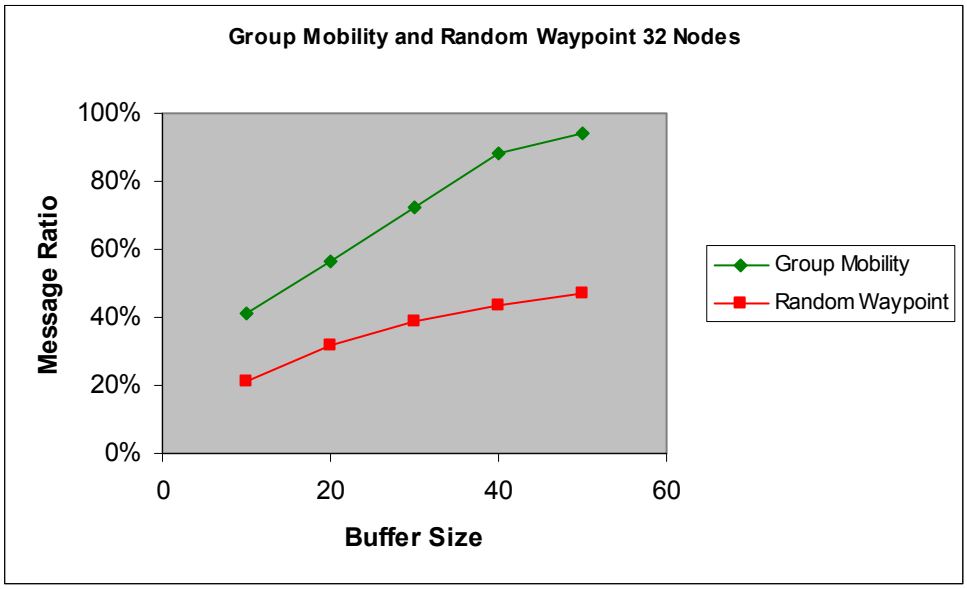

Figure 18 – Percent of Message Delivered for 32 Nodes Using Different Buffer Sizes

Figure 19 shows how latency i.e. the time interval between the time the publisher publishes a message and the time the message is received by the subscriber, varies with population density. Please note that the average latency of each message is taken into account for each message i.e. sum of latencies for each message/ number of subscribers who receive the message. Following are our observations:

- Latency follows an inverse trend with message ratio with respect to population density, i.e. latency decreases as population density increases.
- Following the earlier pattern, message delivery latency is lower in GMM as compared to RWP. As mentioned earlier, message delivery depends on sociable nodes and the epidemic effect. Intuitively, epidemic effect is faster if nodes are in close contact. In GMM clouds are like connected graphs, which implies there exists a path between publisher/proxy-publisher and subscribers, which makes message delivery easier and faster. However, in RWP there is no clear path between publishers and subscribers as they move about randomly. Hence, they are not "connected". This leads to increase in latency.
- As in Figure 16, for message ratio, 32 nodes seems to be the optimal population density for GMM as the latency remains the same for 64 nodes. This implies that the network is well-connected with 32 nodes.
- However, RWP still presents room for improvement at 32 nodes, as we see a further decrease in latency for 64 nodes. This indicates that the epidemic effect becomes stronger with 64 node population density, which results in reduction of latency.

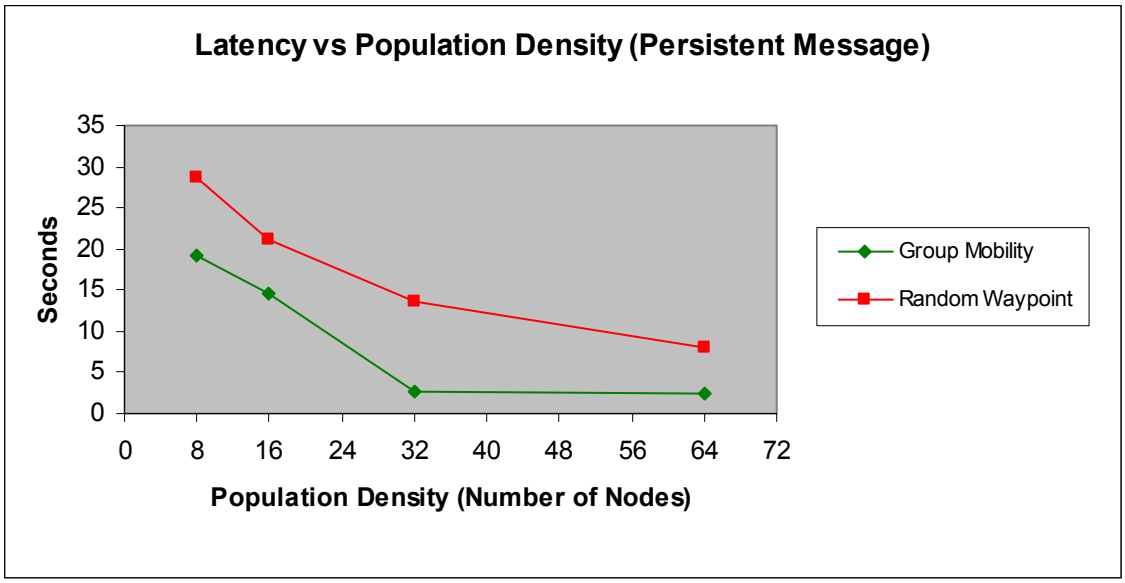

Figure 19 – Latency for 32 Nodes Using Different Buffer Sizes

It is important to note that there is always a trade off between resource consumption and latency. High population density is imperative, if an application requires that messages be delivered to all the subscribers in a short period of time. However, this means that the mobile devices will consume more bandwidth, power and memory. For example sending beacon, meta-data and data happen more frequently. This may lead to a congested network and hence, lower survivability of the network. Hence it is critical to decide which, long survivability of the network or performance is more vital for an application.

Please note that latency values may not exactly reflect reality. Since we are using a discrete event simulator, delays like, transmission time, propagation delay and processing time are not taken into consideration. Also, multiple packets, meta-data, actual data messages etc. are sent at the same time, which in reality is not possible. However, we speculate that the trend will more or less remain the same even if all these parameters were introduced.

Table 3 gives a clearer picture of the trend in Average Message ratio i.e. the average percentage of messages delivered and Average Latency. It is clear in both the mobility models as the population density increases, Average Message ratio increases and Average Latency decreases.

Table 3 – Characteristics of Random Waypoint and Group Mobility as a Function of Population Density

	Random Waypoint				Group Mobility			
Host Number	8	16	32	64	8	16	32	64
Avg Message ratio	39.11 % ± 0.04	50.30 % ± 0.02	58.39 % ± 0.02	78.22 % ± 0.01	61.72 % ± 0.04	70.54 % ± 0.03	91.70 % ± 0.00	92.70 % ± 0.00

Avg Latency (s)	28.59 ± 3.26	21.20 ± 1.30	14.74 ± 0.72	7.92 ± 0.23	19.12 ± 3.25	14.49 ± 1.36	2.64 ± 0.10	2.50 ± 0.04

Figure 20 to Figure 27 show the distribution of delivery ratio. Figure 20 to Figure 23 show the results for different population densities for RWP and Figure 24 to Figure 27 show the results for GMM. We see that just as the average message ratio, average delivery ratio also increases with the increase in population density. Table 4 shows the exact figures:

Table 4 – Distribution of Delivery Ratio Using Random Waypoint and Group Mobility

	Random Waypoint				**Group Mobility**			
Host Number	8	16	32	64	8	16	32	64
Avg Delivery Ratio	54.61% ± 0.78	55.14% ± 0.56	58.88% ± 0.29	78.15% ± 0.40	57.30% ± 0.66	58.35% ± 0.45	78.59% ± 0.59	86.77% ± 0.50

We see that there is a marked improvement in the average delivery ratio in both the mobility models as the population density increases. Again, we notice a big jump from 16 to 32 nodes in GMM, however, in RWP the jump is at 64 nodes.

Figure 20 and Figure 21 indicate a high number of messages are received by a low percentage of subscribers (10%). However, the number of messages received by 60% of the subscribers in Figure 20 is almost the same as the number of messages received by 10% of the subscribers. It is interesting to see that in Figure 21, with a small increase in population density, the number of messages received by 100% of the subscribers is the same as the number of messages received by 10% of the subscribers. Further Figure 22 shows a more even distribution for 32 nodes, with 100% of subscribers receiving the highest number of messages. Again, we see a remarkable jump for 64 nodes in Figure 23, with a bias trend towards 100% of subscribers receiving a very high number of messages.

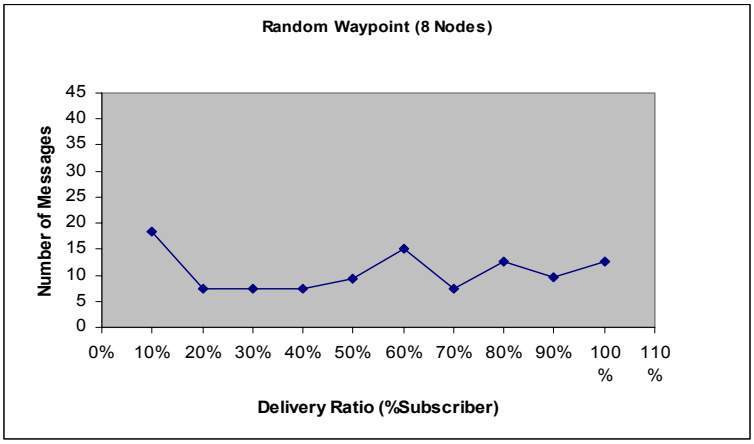

Figure 20 – Distribution of Delivery Ratio (8 Nodes, RWP)

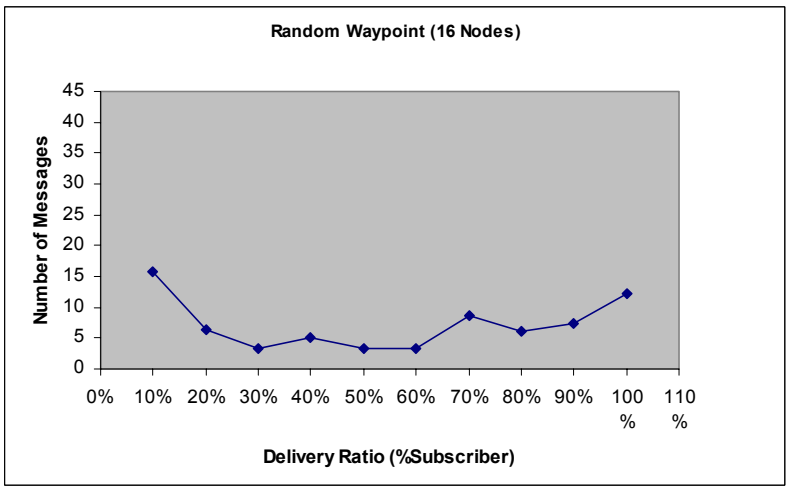

Figure 21 – Distribution of Delivery Ratio (16 Nodes, RWP)

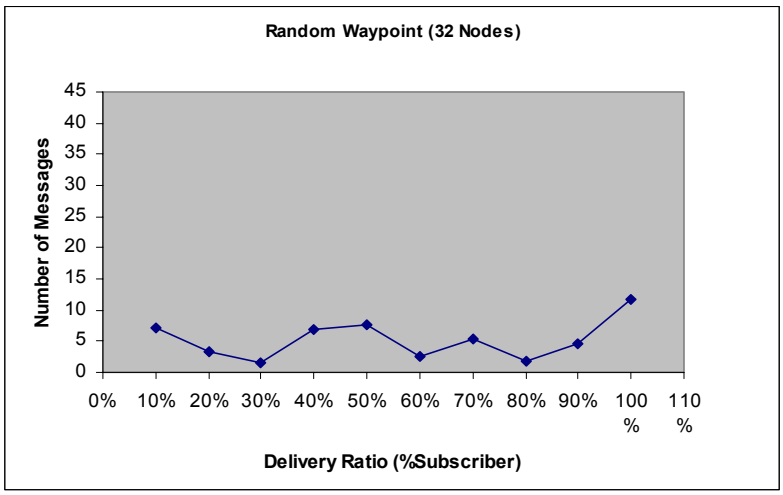

Figure 22 – Distribution of Delivery Ratio (32 Nodes, RWP)

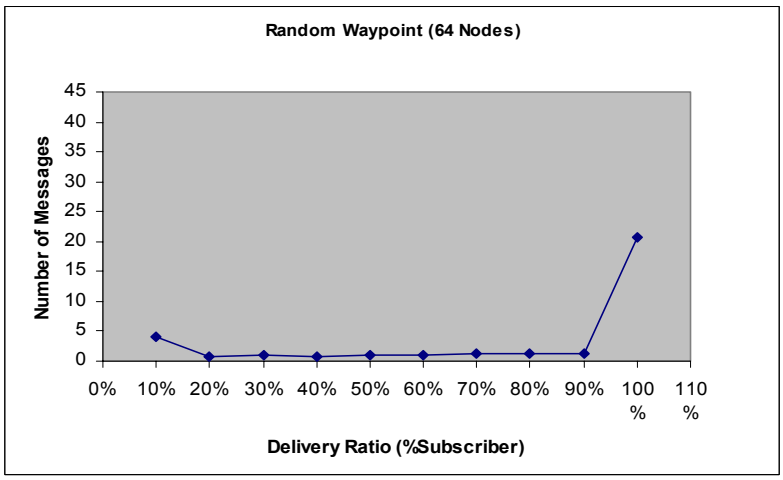

Figure 23 – Distribution of Delivery Ratio (64 Nodes, RWP)

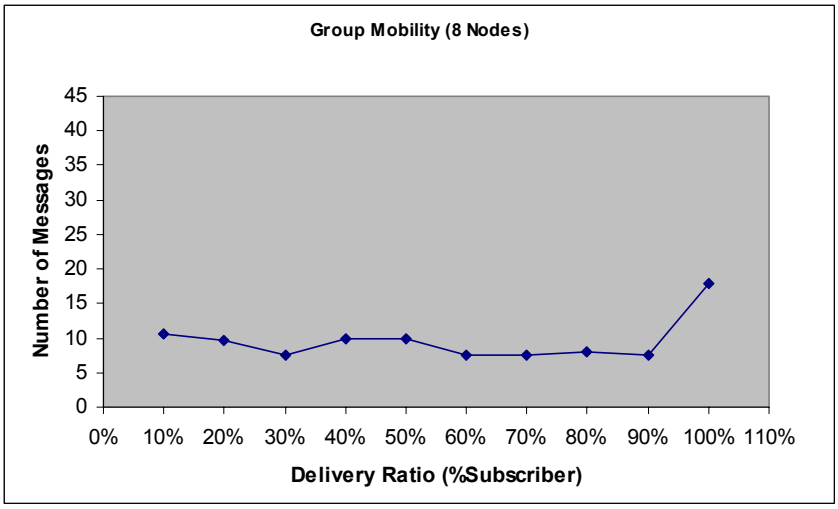

Figure 24 – Distribution of Delivery Ratio (8 Nodes, GMM)

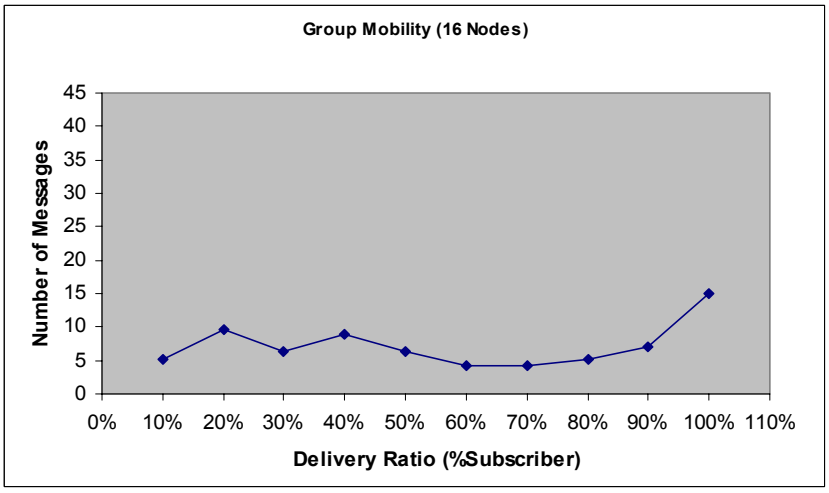

Figure 25 – Distribution of Delivery Ratio (16 Nodes, GMM)

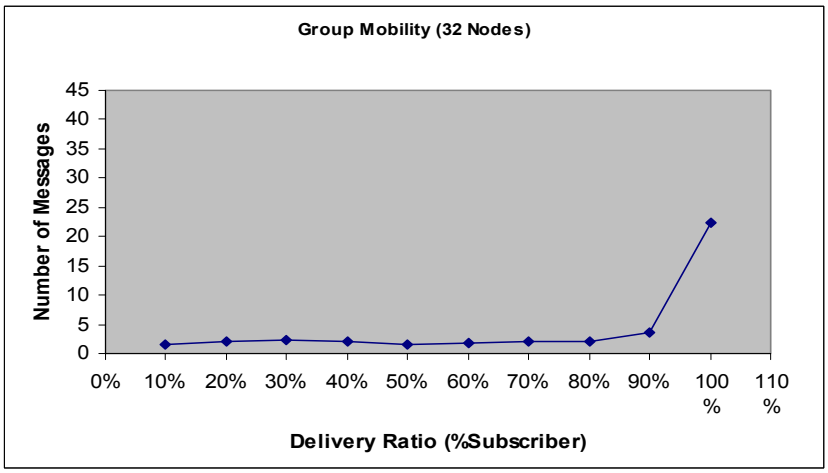

Figure 26 – Distribution of Delivery Ratio (32 Nodes, GMM)

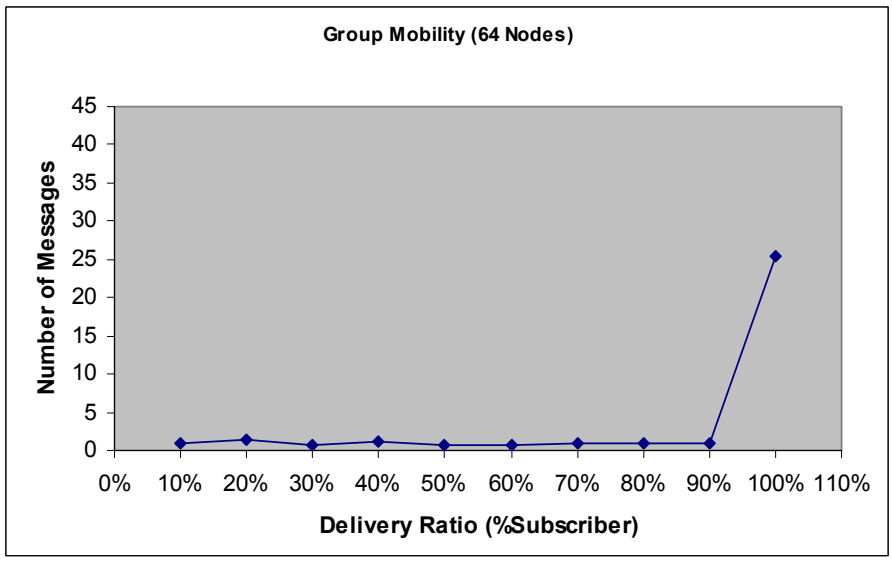

Figure 27 – Distribution of Delivery Ratio (64 Nodes, GMM)

In GMM, we see that the distribution is more uniform for 8 and 16 nodes (Figure 24 and Figure 25), with a bias towards 100% of the subscribers receiving the highest number of messages. Again, as before, there is remarkable change for 32 nodes, where 100% of the subscribers receive a very high number of messages. This implies that, if one subscriber receives a message, it is passed on to all the other subscribers. As a result, very few messages are not received by 100% of the subscribers (about 2 messages received by 10%-90% of the subscribers).

We conclude that message ratio and delivery ratio is highly dependent on population density to increase the spread of a message. This also implies that radio range plays an enormous role. We can increase the spread of a message by infecting more nodes and this can be done faster if the radio range is greater. The greater the radio range the fewer the number of hops required to relay the message to the subscribers.

It is very clear that greater population density increases the epidemic effect and as a result both message ratio and delivery ratio increases within a short span of time. However, this effect is not solely responsible, the fact that sociable nodes are chosen based on their resources and mobility pattern to relay messages plays a major role as well. This effect plays a predominant role in GMM. If individual clouds were not infected with messages by sociable nodes, the epidemic effect alone would not have been sufficient to increase the message ratio and delivery ratio at such a high rate.

6.6 The Point-to-Point Model

The following sections explain the design, results and analyses of the Point-to-Point model.

6.6.1 Simulation Design

Table 5 shows the simulation parameters:

Table 5 – Simulation Parameters

Mobility Pattern	Group Mobility	Random Waypoint
Number of hosts	8/16/32/64	8/16/32/64
Simulation area	1Km x 1Km	1Km x 1Km
Propagation model	free space	free space
Antenna type	omnidirectional	omnidirectional
Transmission range (radius)	200m	200m
Number of groups	5	-
Group area	200m x 200m	-
Node speed	1-3 m/s (randomly generated)	1 m/s
Group speed	1-2 m/s (randomly generated)	-
Number of replicates	30 (for each scenario)	30 (for each scenario)

Each replicate was set to run for 7200 seconds (2 hours) in order to obtain statistically reasonable results. Each parameter (population density, buffer size, latency and speed) is evaluated for persistent messages have unlimited hop count and time to live. For each run, we have two senders, which send at most 44 messages.

6.6.2 Data Analysis

In the Point-to-Point model, there is only one consumer of each message i.e. queue receiver. Hence, we redefine message ratio as the percentage of messages delivered i.e. the ratio of number of messages received to the total number of messages sent.

We begin by evaluating the algorithm in terms of message ratio with increasing population density for both the mobility models. Figure 28 shows the results. Following are our observations:

- The message ratio increases as the population density increases in both the mobility models. In the point-to-point model, message delivery through sociable nodes predominant. This is because there is only one copy per message in the system. Hence, the epidemic effect does not play a part. Message ratio can increase either if the sender comes into direct contact with a receiver more frequently or there are more sociable nodes. We have maintained the percentage of receivers (50%) to be the same in all the population densities. However, the number of messages sent (44) is the same for all the population densities. Hence, the number of receivers for the same number of messages is higher. As the population density increases, the network becomes better connected

and as a result both the probability of a sender/sociable node coming into contact of a receiver/sociable node and the number of sociable nodes increases (the average number of nodes a node meets is greater in dense population). Hence, the increase in message ratio.

- It is interesting to note that unlike publish/subscribe message ratio is only slightly higher in GMM compared to RWP. Since, as stated in 1, the epidemic effect is far less than in publish/subscribe, both the mobility models rely heavily on sociable nodes. In GMM, nodes move in "clouds". Hence, the probability that a sender comes into direct contact with a receiver or a sociable node in the same cloud is higher. This is reflected as the 6-10% increase in message ratio in GMM as compared to RWP.

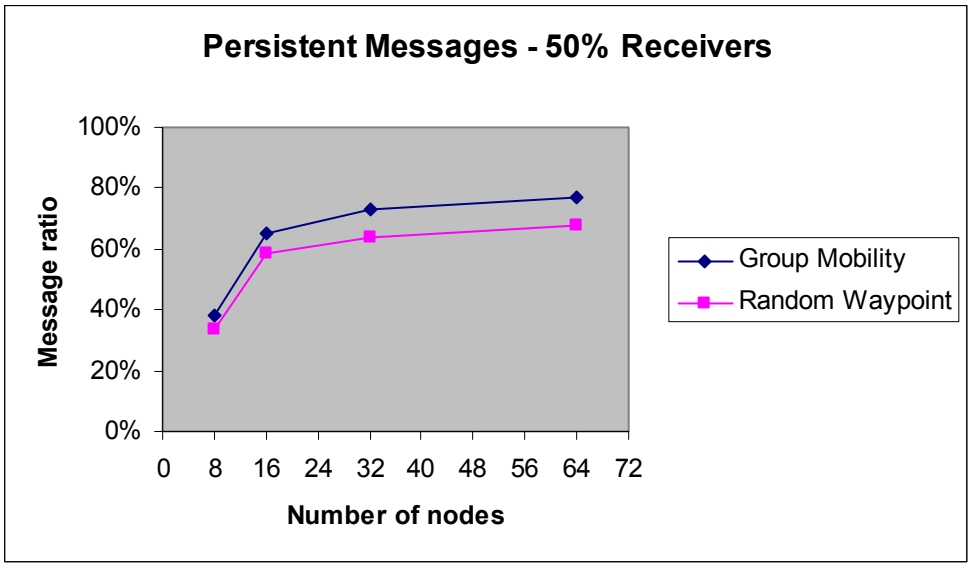

Figure 28 – Message ratio for different population densities for RWP and GMM

Figure 29 shows how message ratio varies as the percentage of receiver nodes increases. The results show the same trend as Figure 28 and follow the same reasoning. As expected, message ratio increases as the percentage of receiver nodes increases. This is because the reliance on sociable nodes for message delivery reduces, as senders meet receivers and send messages directly.

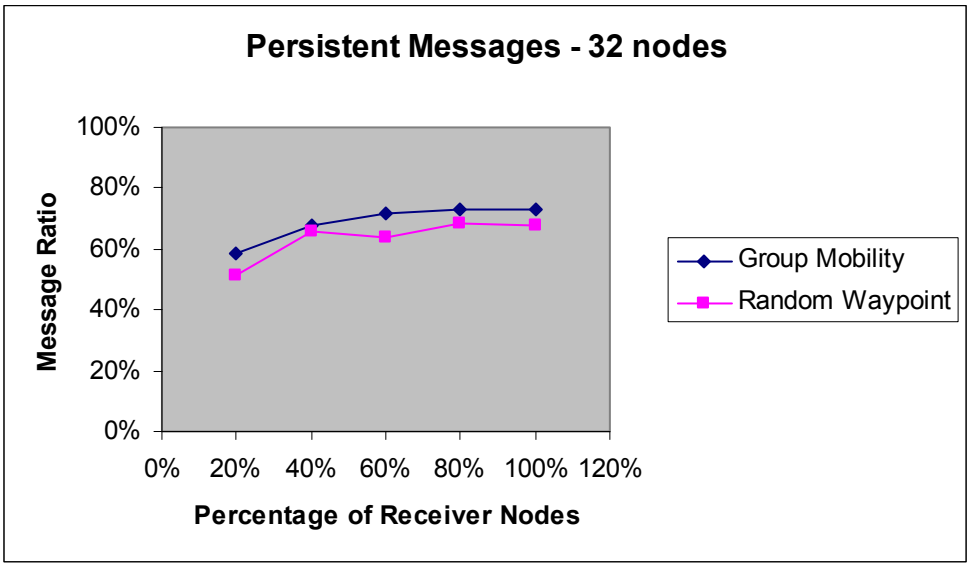

Figure 29 – Message ratio for different percentage of receiver nodes for 32 nodes

Figure 30 shows how latency i.e. the time interval between the time the sender sends a message and the time the message a receiver receives the message, varies with population density. Following are our observations:

- Latency decreases with increase in population density. Latency depends on how often senders meet receivers or sociable nodes. It also depends on how often sociable nodes carrying messages meet receivers. We have maintained the percentage of receivers (50%) to be the same in all the population densities. However, the number of messages sent (44) is the same for all the population densities. Hence, the number of receivers for the same number of messages is higher. Hence, in a dense scenario sender/sociable node meeting a receiver/sociable node faster, which leads to the decrease in latency. Also, as population density increases, the network becomes better connected. Hence, the probability of having a "path" between the sender and receiver increases, which results in reduction in latency.

- GMM has a lower latency as compared to RWP. GMM has clouds, which are well connected. Hence, if a sender/sociable node and a receiver/sociable node are in the same cloud, most often than not they will come into direct contact with each other far quicker than in RWP, where nodes move about more randomly.

- As seen in publish/subscribe, 32 nodes seems to be the optimal population density for GMM, which forms a well connected network. Hence, latency reduces to less than 50s and remains almost stable for 64 nodes.

- As shown in Table 6, the difference in latency between GMM and RWP is almost 120s upto 32 nodes. However, as in publish/subscribe, for 64 nodes, the difference is reduced to 30 seconds. Hence, we conclude that 64 nodes is the optimal population density for RWP, which forms a well-connected network.

- As expected, latency follows an inverse trend with message ratio.

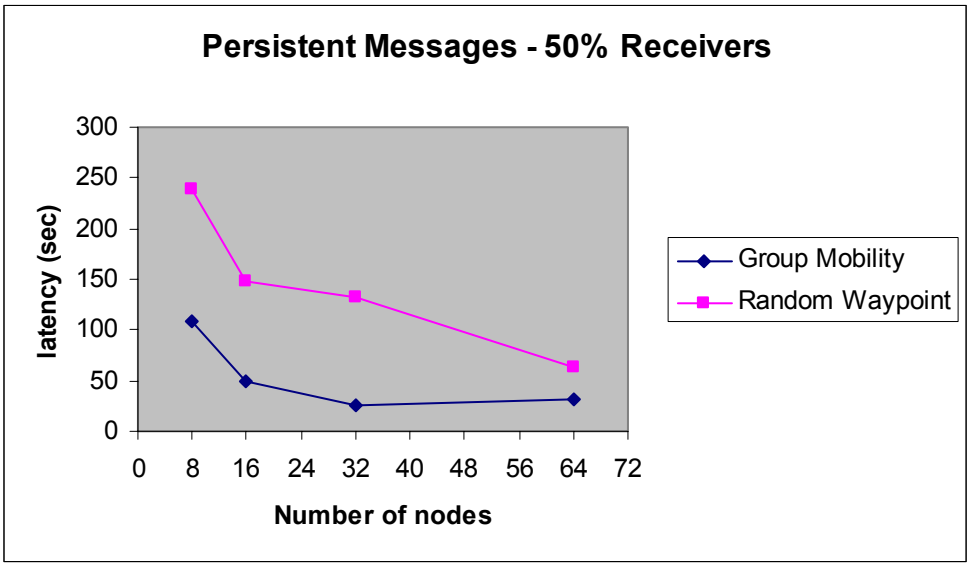

Figure 30 – Latency for different population densities for RWP and GMM

Table 6 – Characteristics of Group Mobility and Random Waypoint Model for different population densities

	Random Waypoint				Group Mobility			
Host Number	8	16	32	64	8	16	32	64
Avg Message ratio	33.54% ± 0.13	58.23% ± 0.10	63.68% ± 0.07	68.07% ± 0.07	38.00% ± 0.13	65.09% ± 0.12	72.91% ± 0.03	76.92% ± 0.03
Avg Latency (s)	238.45 ± 26.83	147.93 ± 17.90	132.55 ± 16.05	62.97 ± 6.30	109.41 ± 36.69	49.60 ± 14.39	25.94 ± 1.82	32.03 ± 4.29

Figure 31 shows how latency varies with percentage of receiving nodes for the two mobility models. Following are our observations:

- Difference in latency between GMM and RWP (~100s) becomes very obvious. This, as mentioned earlier can be attributed to the clouds in GMM.
- In GMM, latency remains stable at ~20s when for percentage of receiver nodes >= 40%. This implies that for a given population density, a minimum level of latency can be expected. In the absence of the epidemic effect, number of sociable nodes for a particular population density remains the same. Hence, latency cannot be decreased any further.

- In RWP, however, increase in percentage of receiver nodes means increased probability of a path between the sender and a receiver. Hence, latency decreases. However, in the absence of the epidemic effect, it still remains quite high at a minimum of ~120s.

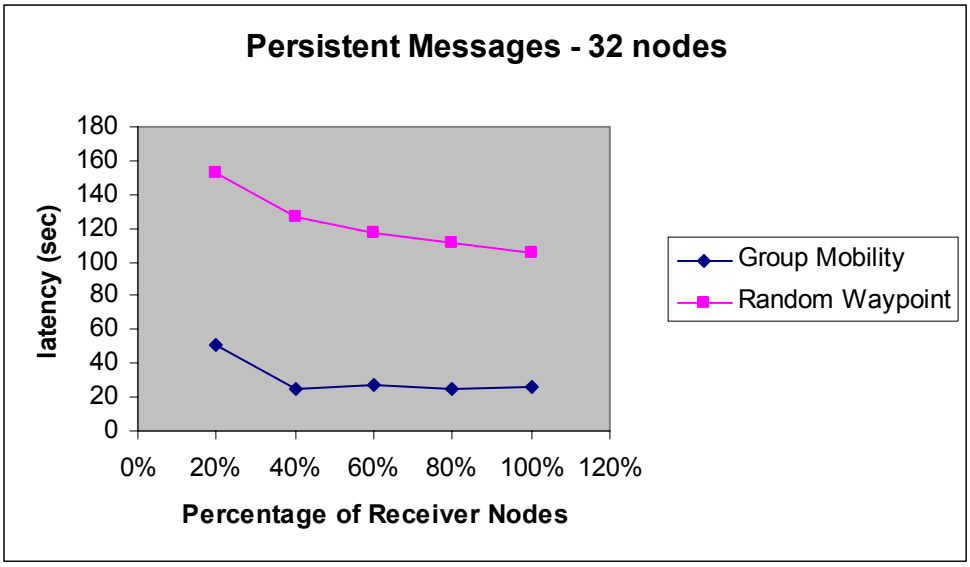

Figure 31 – Latency for different percentage of receivers for 32 nodes

Figure 32 shows the message ratio as the cumulative distribution function of latency for GMM. The results reflect a combination of the results of Figure 28 and Figure 30; hence, the same observations and reasons for behavior apply. Message ratio increases with latency and population density. It is remarkable that message ratio doubles at 50s to 80% for 16 nodes from 40%+ for 8 nodes. This implies that for a node population density of >=16, *exactly once* may be achieved!!!

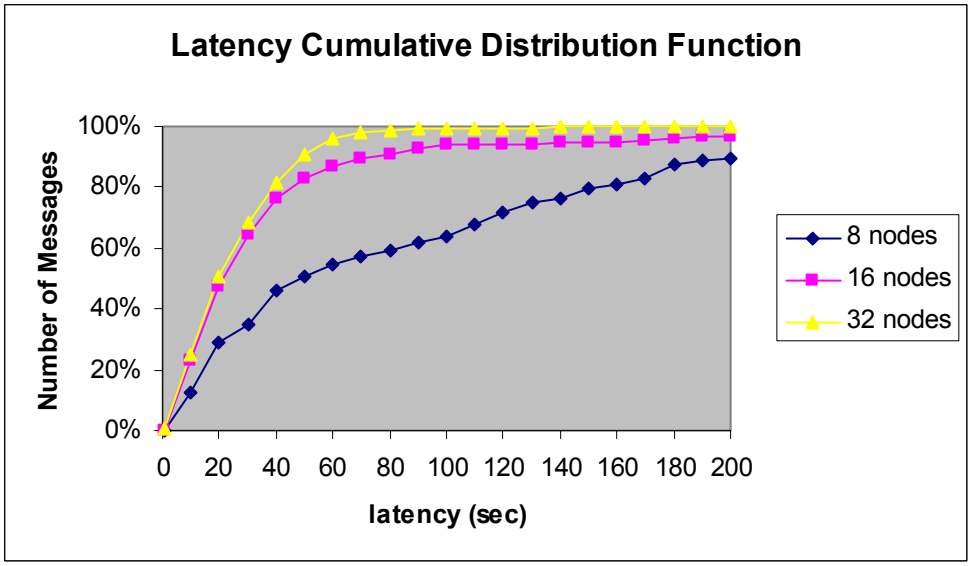

Figure 32 – Message ratio as a function of time and population density for GMM

Figure 33 shows a clearer picture of latency per message. Following are our conclusions:

- For 8 nodes, the values of the message latencies are highly spread (the confidence interval is very high). This indicates that for this number of nodes we cannot get a good estimation of the mean value for the latency. On the other hand, for 32 nodes the distribution of the message latency is closer to the mode value, which is around 25 sec, so the confidence interval in this case is far lower.
- For 16 and 32 nodes majority of the messages have a latency of 10-20s. For 16 nodes, the majority of the messages that have been received have a latency of less than 50 sec, thus we can make a better estimation of the mean latency. This characteristic is improved further for 32, hence, the confidence interval is even smaller.

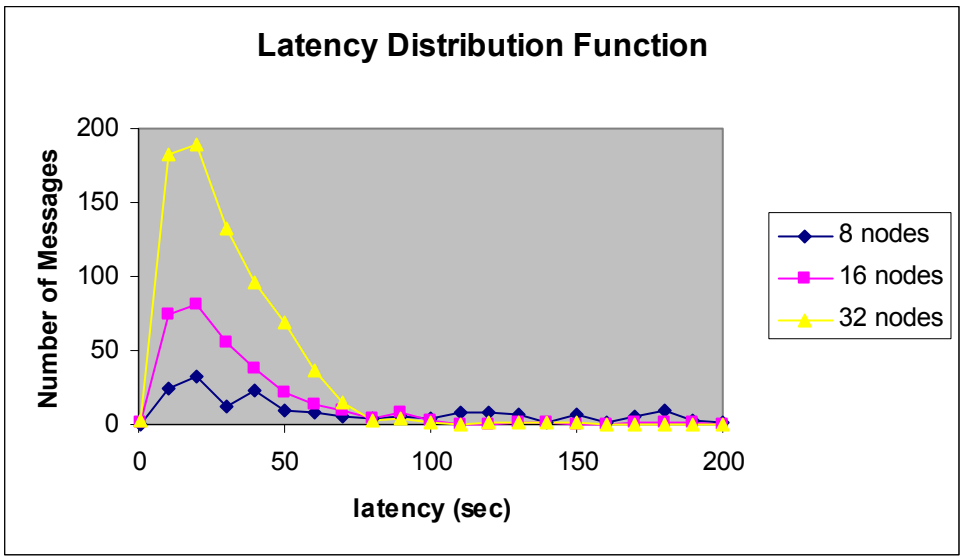

Figure 33 – Latency distribution for different population densities for GMM

In *[19]* it is proved that speed is one of the important features that influence the throughput and delay in MANETs. Next we evaluate the effect of speed of the nodes on message ratio. Figure 34 shows the results. We see that as expected, message ratio is higher for node speed of 2m/s as compared to 1m/s. Increasing speed effectively means increasing the simulation time. As we have seen in Figure 30, message ratio increases with time. There are times when senders are completely isolated from the receivers for the whole simulation time. Increased speed reduces the number of such situations. Hence, by increasing speed, senders/sociable nodes meet receivers/sociable nodes faster resulting in increase in message ratio. However, this trend may not be true for very high speeds, as nodes may not have the opportunity to exchange messages.

Table 7 gives a clear picture of the average message ratio for the two speeds. While the average message ratio for 2m/s remains higher the difference reduces as population density increases.

Table 7 – The Random Waypoint performance for two different speeds

	Random Waypoint – Speed =1 m/s				**Random Waypoint – Speed = 2 m/s**			
Host Number	8	16	32	64	8	16	32	64
Avg Message ratio	30.45% ± 0.11	41.11% ± 0.09	42.37% ± 0.12	54.50% ± 0.99	33.54% ± 0.13	58.23% ± 0.10	63.69% ± 0.07	68.07% ± 0.07

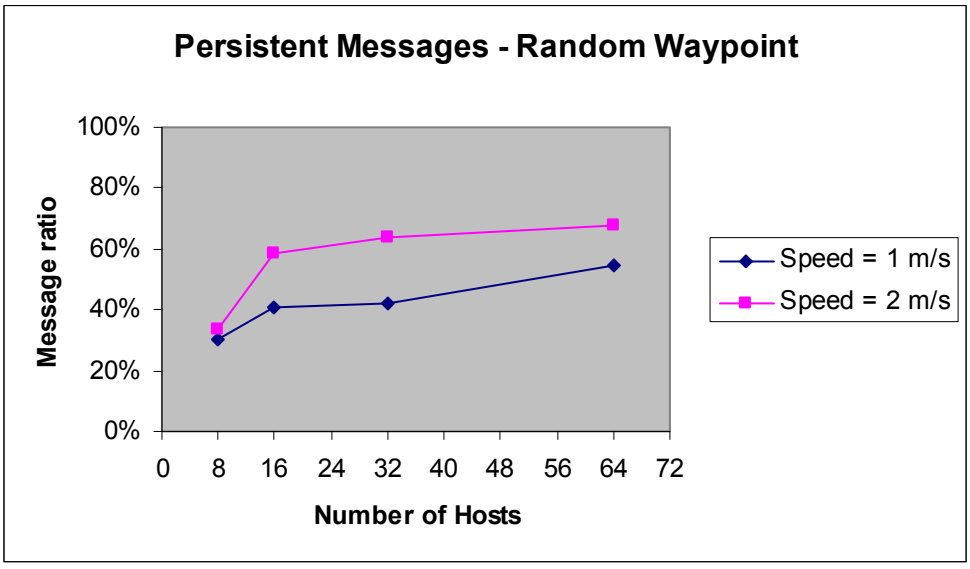

Figure 34 – Message ratio for different speeds for 50% of the population as receivers

7. Prototype

7.1 Introduction

This chapter explains the proof of concept for the JIMS algorithm. The first section gives the overview of a message application. In the following sections, we discuss the design, implementation and testing of the APIs and the GUI.

7.2 Overview

A message application similar to the Short Message Service (SMS), which runs on a mobile handset, is implemented along with the GUI, with the JIMS algorithm running underneath. The application is based on the Publish/Subscribe model of JMS. The user is allowed to publish a message or subscribe to a chosen topic. Message delivery is according to the JIMS algorithm. When a message is received, the user will be alerted and the message will be displayed on request. All the messages can be saved in an inbox for browsing. All this can be done using a user-friendly GUI.

We chose to develop an SMS type application for two reasons:

- **Simple and text-based**

  As long as we can provide reasonable proof of concept for the algorithm through a simple text based application, developing more complex applications like multimedia applications will be more a matter of application design rather than designing the underlying message delivery mechanism.

- **Delay tolerant application**

  As mentioned before, MANET applications need to be fairly delay tolerant. An SMS type application complies with this requirement and is well suited for a MANET environment.

7.3 Design

The application is targeted at mobile devices such as PDAs, smart phones etc. Hence, we chose Java J2ME, Personal Profile, supported by most of these devices, as our development platform. Throughout design and implementation, we have taken into account the computational power and memory constraints in these devices and tried to be as conservative as possible. The packages used are:

- java.lang
- java.util
- java.net

Design and implementation involved three main modules:

- JMS APIs
- JIMS algorithm (underlying delivery mechanism)
- GUI

Application developers can use the JMS APIs as specified in the JMS specification to develop messaging applications for MANETs. These APIs will invoke the underlying message delivery mechanism, which is the JIMS algorithm. The GUI makes these issues transparent to the users.

7.3.1 Class Diagram

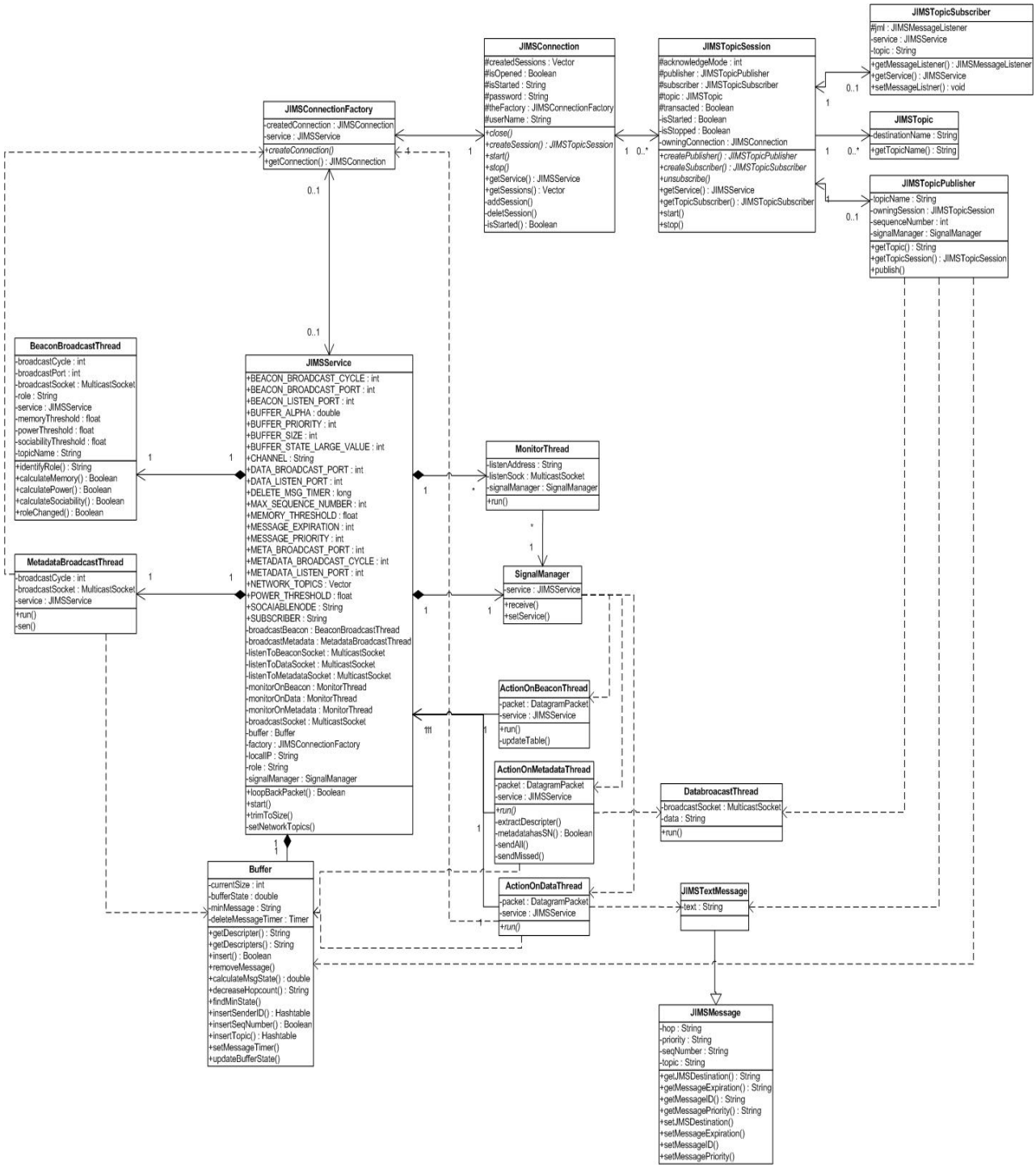

Figure 35 – Class Diagram

A copy of the class diagram may be found on the CD for a more comprehensive picture.

Details of each class follow.

7.3.1.1 JIMSService

JIMSService is the administration, configuration and resource core of JIMS. It allocates and manages the resources needed to run the delivery daemon.

JIMSService has defined a set of public static final parameters that are aimed for administration and configuration use. It specifies:

- The port numbers used for different sockets
- The size of local buffer
- The memory, power and sociability thresholds are for decision making.
- Some public static utility methods shared by the whole package. For example, loopBackPacket() is capable of detecting whether a received packet is sent by the host itself. A node may be a publisher and subscriber of the same topic, however packets sent by itself should not be buffered.

As it is the core, an instance of JIMSService is the central point where all the other objects will obtain information from or update information to. As illustrated in the class diagram, JIMSService is composed of BeaconbroadcastThread, MetadatabroadcastThread, MonitorThread, SignalManager and Buffer classes. These five classes are immediately instantiated when JIMSService is started. These are the fundamental classes, which have methods carrying out the core activities of the algorithm.

JIMS users have to start the JIMSService daemon before using the application. JIMSService may be started by calling the method start().When started JIMSService sets up the following:

- A multicast socket to send out any kind of message i.e. beacon, meta-data or data message.
- Three sockets, one each for listening for beacon, meta-data and data messages.
- Three instances of MonitorThread, which listen on the sockets described in 2.
- An instance of the BeaconbroadcastThread to broadcast beacons periodically
- An instance of the MetadatabroadcastThread to broadcast meta-data periodically

In all, 4 sockets are set up and 5 threads are started.

7.3.1.2 SignalManager

SignalManager is instantiated when JIMSService is started. It distinguishes between various signals received (beacon, meta-data, data message) by examining the port number on which the signal is received. SignalManager has an public method receive(), which determines the type of the message, it starts the corresponding thread i.e. ActiononBbeaconThread when a beacon is received, ActiononMetadataThread when meta-data is received and

ActiononDataThread when a data message is received. Once the thread is started, the received packet is passed on to it for further processing. As soon as the thread is started, SignalManager continues to handle the next signal.

7.3.1.3 Buffer

This class is instantiated when JIMSService starts. Buffer is an implementation of the storage space of a node for all the messages it accepts. This is another important and complex component. It implements the buffer data structure and provides fundamental interfaces for buffer operations. For example, insert and remove. It also implements buffer management operations as described in step 4 of the algorithm.

The implementation of the data structure should allow easy insertion and deletion of messages according to the message state. Hence, we decided to implement it using chained hash-tables. This greatly reduces the computation and complexity involved in implementing a complex search mechanism. However, we do appreciate that it is a more complex data structure than a Queue, but it is certainly deployable in PDA like devices. However, the choice of the data structure is flexible depending on the device and the applications as long as it provides the interfaces for queue management.

A hash table is characterised by two fields, key and value. Three hash tables have been chained. The first has the topic name as its key the second hash-table as its value. This chains the first two hash tables. This chained hash-table has the sender ID as its key and the third hash-table as its value. Now we have a chain of three hash tables, with the third hash table having the sequence number as its key and a message as its value. Thus, a message is identified by a combination of its topic, sender ID and sequence number.

Buffer provides two main interface methods, insert() and getDescriptor().insert() has three other private methods: insertSeqNumber(), insertSenderID() and insertTopic(), which is nested as a parameter to insert() as shown:

```
insertSeqNumber(insertSenderID(insertTopic(topic), senderID), seqNumber,
payload);
```

insertTopic() returns the hash-table of sender-IDs; insertSenderID() then searches for the SenderID of the message in this hash-table, and returns the third hash-table of sequence numbers. Finally, the payload is inserted into this hash-table. insertSeqNumber() returns a boolean value, which indicates whether the insertion operation has been successful. insertTopic() and insertSenderID() has to create new hash-table if the topic or sender id of the message doesn't exist and this new hash-table will be returned instead.

When a message is inserted, two other operations are carried out:

- A timer is set for the message equal to the remaining time to live as described in step 2 of the algorithm.
- Update the buffer state if it is full (updateBufferState()), calculate the message with the minimum message state (findMinState()), so that the message with the least message

state may be replaced if a new message arrives. calculateMsgState() is used to calculate message state.

The hop count of the message is decreased (decreaseHopcount()), before it is inserted.

Another java Timer object is instantiated in the constructor of Buffer. It provides an interface, called "schedule()", which schedules TimerTask operation after certain time. A class that extends TimerTask is one of Buffer's private attribute. It is effectively a thread provided by java.lang and in the interface method run(), it is programmed to remove a message that has been passed in to constructor by calling the private method removeMessage().

7.3.1.4 BeaconBroadcastThread

BeaconbroadcastThread is started as soon as JIMSService is started. This thread periodically broadcasts the beacon.

The current profile of the node is indicated in the beacon. This is done by calling, calculateSociability(), calcuatePower() and calculateMemory() are in method identifyRole(). Different thresholds are utilized in the three methods for decision-making. The default thresholds are defined in JIMSService. The node's profile is checked periodically, however, this cycle, defined in JIMSService, is longer than the beacon sending cycle. If the node profile changes, a call-back method in JIMSService setRole() is called to reflect this. As described in the algorithm, a lot of logical decisions are made based on this information. The thread is put to sleep for sometime after the beacon is sent. This saves power.

7.3.1.5 MetadataBroadcastThread

MetadataBroadcastThread is started as soon as JIMSService is started. This thread periodically broadcasts meta-data according to the cycle defined in JIMSService.

Initially the thread is sleeping. Once active, it finds out the topic/s of the messages it needs to indicate in the meta-data depending on the node profile. For a subscriber, a handle to JIMSConnectionFactory is used to trace down to JIMSTopicSubscriber where the topic that specific subscriber subscribes to can be found out. Messages of only this topic are indicated in the meta-data. For a sociable node, all the messages in the buffer are indicated.

In order to construct the descriptor of the meta-data, this class uses the method getDescriptor() provided in the Buffer class, which returns the descriptor. The meta-data message is passed on to a local private method send(), which carries out the actual broadcast. If a node has more than one subscriber instances, meta-data for each instance is sent one by one.

7.3.1.6 DataBroadcastThread

Unlike BeaconBroadcastThread and MetadataBroadcastThread, DataBroacastThread is not instantiated when JIMSService is started. This is because messages are published only on request by the application and messages are relayed only after processing the meta-data. MonitorThread

In contrast to the three broadcasting threads, MonitorThread is designed for listening. When JIMSService is started, three MonitorThread is instantiated and started thrice on three different ports. The first listens for beacons, second for meta-data and the third for data

messages. Once a packet is received, the packet as well as the port number will be passed to SignalManager using the callback function receive() for further processing. This enables decoupling of listening and processing of signals.

7.3.1.7 ActionOnBeaconThread

ActionOnBeaconThread is instantiated and started by the SignalManager when a beacon is received. ActionOnBeaconThread performs the appropriate "action" when a beacon is received, which is updating the table (updateTable()) as described in step 1 of the algorithm.

7.3.1.8 ActionOnMetadataThread

ActionOnMetadataThread is instantiated and started by the SignalManager when a meta-data message is received. ActionOnMetadataThread performs the appropriate "action" when metadata is received i.e. analyze the signal pattern of the received meta-data descriptor and send messages accordingly as described in step 5 of the algorithm. This involves the Buffer class, which constructs the descriptor of the meta-data receiver node and the DataBroadcastThread, which multicasts the data messages.

ActionOnMetadataThread is one of most complex components of the prototype, technically and logically. It has several private methods, extractDescriptor(), sendMissed(), metadatahasSN() and sendAll(), which process the two descriptors and determine which methods need to be sent.

7.3.1.9 ActionOnDataThread

ActionOnDataThread performs the appropriate "action" when a data message is received i.e. pass the message to the application if the appropriate subscriber has been instantiated. The JIMSConnectionFactory class is used to trace the appropriate subscriber instance. The onMessage() method of the JMS API MessageListener is called in this thread to carry out the action as directed by the application on receipt of a message.

The processing involved in onMessage() may be varied and dependent on the application. Since the application can have many subscriber instances, it is possible that while processing a particular data message, ActionOnDataThread misses out on receiving other data messages.

Hence, it is better to have another thread so that ActionOnDataThread can continue to process and without missing too many data. So, a private class ReceMessageThread, has been defined in this class. It simply passes the text messages to onMessage(). Therefore, there will be no blocking. ActionOnDataThread doesn't have to wait if the recemessagethread is taking of a time-consuming application.

7.4 Basic Building Blocks of a JMS Application

The prototype strictly follows the steps defined in JMS specification1.1 to establish a publisher and subscriber and prepare to send and receive messages as in the Publish/Subscribe model. Figure 36 shows the basic components required to design a JMS application as specified in the JMS specification. First, we get the JIMSTopicConnectionFactory in order to create JIMSTopicConnection and further the JIMSTopicSession. Then by calling JIMSTopicSession's method createPublisher and

createSubscriber (), passing parameter topic as the destination, the publisher and subscriber will be created correspondingly and start sending and receiving messages.

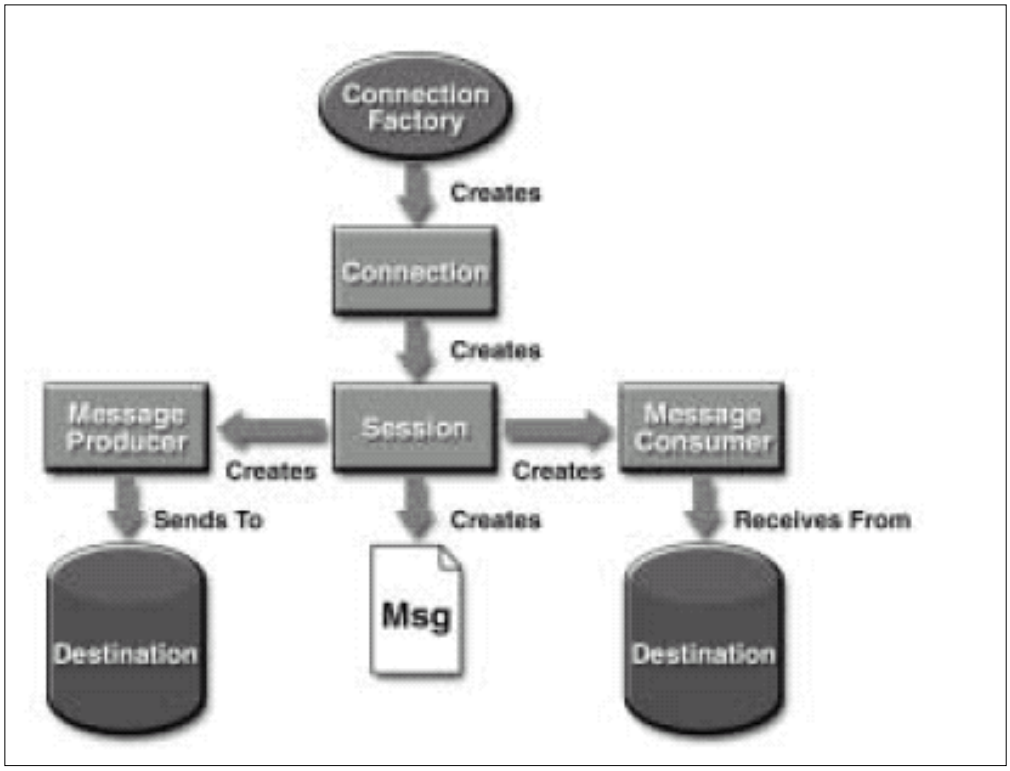

Figure 36 – Building Blocks of JMS Application

7.4.1 Administered Objects

Administered objects are objects ore-configured and stored by JMS administrators for the use of JMS clients. These objects are normally stored in the namespace of a naming and directory service and are available to the JMS clients for standard JNDI lookup.

Implementing JNDI would mean a central JNDI server, which is not suitable for a MANET scenario. Hence, each node will have its own set of configuration files, which it may look up and create administered objects on its own based on the information in these files.

We have implemented few basic APIs of the JMS Publish/Subscribe model, which are essential for developing a JMS publish/subscribe application. Please note that the APIs implement only the basic methods required by the application and not all the methods specified in the specification.

7.4.2 Application Programmable Interfaces (APIs)

Following is a list and details of each API we have implemented:

7.4.2.1 JIMSConnectionFactory

JIMSConnectionFactory is JIMS equivalent to JMS ConnectionFactory. Connection factories are administered objects used by JMS clients for creating connections to a JMS provider.

JIMSConnectionFactory provides methods to create JIMSConnection (createConnection()) to connect to JIMSService (provider). JIMSConnectionFactory also keeps track of which connections have been created and their states i.e. whether they have been started or stopped. It contains private parameters. One is for JIMSService, the other is for created JIMSConnection. Two methods, getService() and setService() are meant for public access to the private JIMSService parameter.

7.4.2.2 JIMSConnection

JIMSConnection is JIMS equivalent to JMS Connection. JIMSConnection is created using the methods provided by JIMSConnectionFactory (createConnection()). It has a parameter, theFactory, which records which factory it belongs to. Since ours is a publish/subscribe application, to keep things simple, we have implemented JIMSConnection as the topic connection in JMS. Hence, JIMSConnection acts as a factory to create JIMSTopicSession. Hence, it keeps track of the number of sessions created, started and stopped.

7.4.2.3 JIMSTopicSession

JIMSTopicSession is JIMS equivalence to JMS TopicSession. JIMSTopicConnection is created using the methods provided by the JIMSConnection (createSession()). It has a parameter, owningConnection, which records, which connection it belongs to. JIMSTopicSession provides methods to create JIMSTopicPublisher (createPublisher()) and JIMSTopicSubscriber (createSubscriber()). In order to keep the prototype simple, JIMSTopicSession supports a maximum of one JIMSTopicPublisher and one JIMSTopicSubscriber. However, it can easily be enhanced to support multiple publishers and subscribers by using Vector.

7.4.2.4 JIMSTopicPublisher

JIMSTopicPublisher is JIMS equivalent to JMS TopicPublisher. JIMSTopicPublisher is created using the methods provided by JIMSTopicSession (createPublisher()).It has a parameter, owningSession, which records, which connection it belongs to. This class is used to publish a message of a particular topic specified by the application (publish()). It instantiates DataBroadcastThread class, which multicasts the constructed JIMSTextMessage.

JIMSTopicPublisher also needs to remember what topic it is publishing and more importantly, the sequence number for each message it sends. The sequence number is increased every time after publish() is called.

7.4.2.5 JIMSTopic

JIMSTopic is JIMS equivalent to JMS Topic.

7.4.2.6 JIMSMessage

JIMSMessage is JIMS equivalence to JMS Message. To simplify the design, JIMSMessage only defines the fields, hop count, priority, sequence number and topic. JIMSMessage is created by JIMSTopicSession. This class provides get and set method pairs to get and set the parameters.

7.4.2.7 JIMSTextMessage

JIMSTextMessage is JIMS equivalent to JMS TextMessage. JIMSTextMessage extends JIMSMessage. It has an additional field, for the payload i.e. the text message. This class provides a get and set method pair to get and set the text.

7.5 The Graphical User Interface (GUI)

We have implemented a user-friendly GUI so that the user may interact with the application easily.

7.5.1 GUI Classes

We explain the GUI class in the following section.

7.5.1.1 MainWindow

Once called this class creates the GUI. This class provides an easy user-friendly method for the user to start/stop publishing or subscribing messages to a particular topic. It also allows users to review message history i.e. view all the messages sent and received.

The four main components of the class are:

- **Set up Tab**

  This tab allows the user to choose a topic from the listed topics and create the publisher/subscriber instance for a particular topic specified by the user. If "NONE" is selected nothing will be published or received. A subscriber instance needs to be registered to a message listener so that the user can be informed when a message is received.

- **Send Message Tab**

  Allows users to write the message they are going to publish. The text is then passed on to the publisher instance, which handles the publishing. The user has an option to save the message. If this option is chosen, the message is saved to an external file.

- **Popup window**

  When a new message with a topic to which the user is subscribing is received, it is passed up to a popup window to alert the user. The user has the option of saving the message to view it at a later time.

- **History Messages Tab**

  Gets the history information from the external file, all the messages, sent and received, together with the sender's name, the date, and exact the time when they are published will be displayed.

7.6 Enhancements and Issues

In the initial design of the algorithm, we decided that meta-data is sent whenever a beacon from a subscriber or sociable node is received. This was done to reduce latency and leverage

the proximity of a potential proxy-publisher. However, from our observations of the simulations, we concluded that this led to too many meta-data messages in the system. Since, a node receives beacon from multiple nodes at a time, it sends meta-data in response to each of them. This results in multiple copies of the same meta-data. Also, it is possible that multiple neighbours have copies of the same message, which are sent multiple times as a response to meta-data received from the node. Hence, we decided to send meta-data periodically, with a cycle longer than the beacon cycle. Since, the data messages are multicast, a node can receive missing messages, which may be in response to another node's meta-data, if it is in range of the sender. Hence, it sends meta-data only for those messages, which it genuinely has not received. This improvement can have an enormous impact on the amount of network traffic, thus saving bandwidth and increasing the survivability of the network.

Since multicast sockets are used to send different kinds of packets, any node listening to the multicast address will receive any packet that is sent. So a wireless network where a node can be out of range of another node cannot be emulated. Packets loop back to the sender as well. The only way to deal with this is to discard such packets.

7.7 Profiles and Instances

It is important to differentiate between node profiles and subscriber/publisher instances. Nodes may assume a certain profile, as described in section 4.6, according to their available resources and mobility pattern. However, from the user's point of view, every node should be able to publish and subscribe to topics. Hence, every node can create JMS publisher/subscriber instances, however, whether or not they can be proxy-publishers/proxy-subscribers depends on the node profile.

This introduced a new challenge while implementing the prototype. When a node with a subscriber profile creates a subscriber instance of a topic, it is capable of receiving messages only of that topic and each message received is shown to the user. However, if a sociable node creates a subscriber instance, it is capable of receiving messages of any topic. When a sociable node creates a subscriber instance of a particular topic, there may already be some messages of the topic in the buffer, which had arrived before the instance was created. However, the user is alerted only about those messages that are received after the instance is created. Although not implemented, we have come up with a solution. Each message bears a flag indicating whether or not it has been read. Each time a subscriber instance is created, the user can be alerted of all the messages of the topic, which have not been read in the buffer.

7.8 Test Cases

Following are the test cases used for the prototype and the simulation implementation:

Table 8 – Test Cases

Receiver / Sender	**Publisher** (Expected behavior from publisher's point of view)	**Subscriber** (Expected behavior from subscriber's point of view)	**Sociable** (Expected behavior from sociable node's point of view)
Publisher	• Send out the beacon periodically • Receive the beacon from the sender • Will receive all the messages sent by the sender, but won't insert them	• Send out the beacon periodically • Receive the beacon from the sender • Send out the metadata with the subscribing topic • Will receive all the messages sent by the sender, but only insert and show those messages it is subscribing if they don't exist in the buffer	• Send out the beacon periodically • Receive the beacon from the sender • Send out metadata with all different topics in the network correspondingly • Will receive and insert all the messages sent by the sender, but only those messages it is subscribing will be shown
Subscriber	• Send out the beacon periodically • Receive the beacon from the sender • Receive the metadata with a topic subscribed by the sender • Send back the missing messages with this topic according to the metadata received • Will receive the messages sent by the sender, but won't insert them	• Send out the beacon periodically • Receive the beacon from the sender • Send out the metadata with the subscribing topic • Receive the metadata with a topic subscribed by the sender • Send back the missing messages with this topic according to the metadata received	• Send out the beacon periodically • Receive the beacon from the sender • Send out metadata with all different topics in the network correspondingly • Receive the metadata with a topic subscribed by the sender • Send back the missing messages with this topic according to the

		• Will receive all the messages sent by the sender, but only insert and show those messages it is subscribing if they don't exist in the buffer	metadata received • Will receive and insert all the messages sent by the sender, but only those messages it is subscribing will be shown
Sociable	• Send out the beacon periodically • Receive the beacon from the sender • Receive metadata with all different topics in the network correspondingly • Send back the missing messages with all correspondent topics according to the metadata received • Will receive the messages sent by the sender, but won't insert them	• Send out the beacon periodically • Receive the beacon from the sender • Send out the metadata with the subscribing topic • Receive metadata with all different topics in the network correspondingly • Send back the missing messages with all correspondent topics according to the metadata received • Will receive all the messages sent by the sender, but only insert and show those messages it is subscribing if they don't exist in the buffer	• Send out the beacon periodically • Receive the beacon from the sender • Send out metadata with all different topics in the network correspondingly • Receive metadata with all different topics in the network correspondingly • Send back the missing messages with all correspondent topics according to the metadata received • Will receive and insert all the messages sent by the sender, but only those messages it is subscribing will be shown

8. Project Management

8.1 Introduction

With our goals and deliverables clearly defined, we decided to get our foundation right. Both the simulation implementation and analysis and the prototype were heavily dependent on a good algorithm design. Hence, we decided to design a good algorithm, which meets all the functional and non-functional requirements, focus on implementing and testing it rigorously since the simulation results would reflect how effective it is. Once this is done, we move on to the prototype to show a practical application of the algorithm.

Since ours is a pure research project, it was very difficult to define requirements precisely right in the beginning. The best way to achieve and deliver concrete results was to start simple and keep building on this with new features and modifications. Hence, we decided to follow the Incremental Iterative Development (IID) *[20]* methodology for each of our deliverables. This is explained in detail in the next few paragraphs.

8.2 Iterative Incremental Development

Feedback between phases

A research project, as the name suggests involves continuous research and thus constant improvement and modifications of ideas and implementation. Hence, it cannot have clear requirements, design, implementation and testing phases. The cycle needs to be repeated several times in order to make refinements and have a reasonably good result/deliverable. Other models like the waterfall model do not cater to this requirement.

For all out three main deliverables, algorithm design, algorithm implementation and evaluation and the prototype, we decided to have three main phases. In each phase we focus on a particular feature. Hence, all the features were categorised into one of three models, core, publish/subscribe or point-to-point. Figure 37 illustrates the development process:

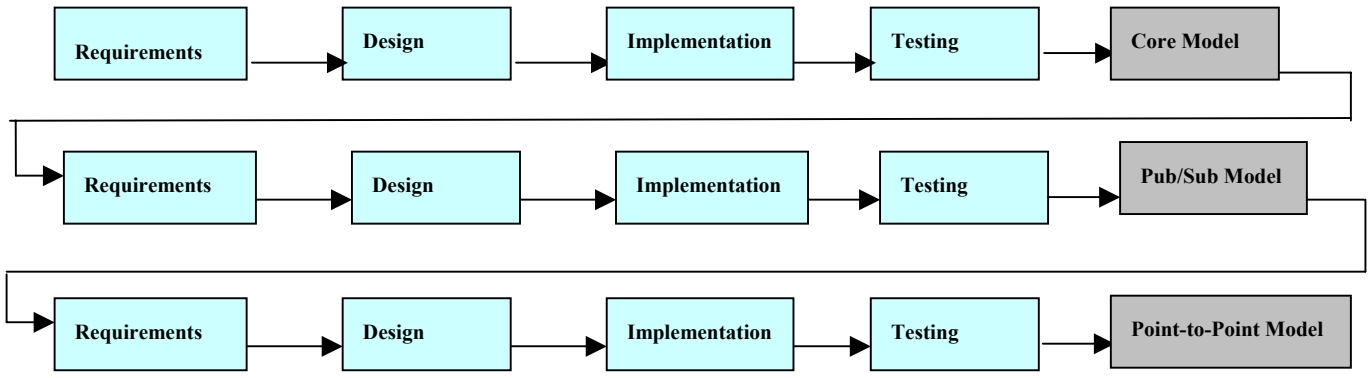

Figure 37 – Iterative Incremental Development

The implementation of the algorithm and prototype would involve all the four phases, however, the design phase of the algorithm itself we have only the requirements and design phase. As seen in Figure 37 each phase gives a feedback to the next phase. Each of these phases would have several smaller phases to enable systematic build up of ideas. This will enable us to spot weaknesses and rectify them earlier rather than later. Further, according to the IID philosophy [*20]*, we chose to deal with the core model i.e. the basic functionalities first as that is the most high risk and error prone module. If we got that right then the other two modules would follow into place fairly smoothly.

Again, according to the IID philosophy *[20]*, for the implementation of the algorithm and the prototype an entire module was broken into small slices, which were as far as possible independent of each other. This would enable us to gain better control over the project and measure our progress tangibly. In order to do this, we designed use cases [see UseCase] and treated each use cases or a set of use cases as appropriate as one slice. We designed each slice such that it would take a maximum of two weeks to complete. For example, the table implementation and the queue management implementation were two separate slices, which were implemented in parallel and later integrated after rigorous testing. Each of these slices almost always had their own requirement, design, implementation and testing micro cycle.

8.3 Project Management Methodology

The choice of project management methodology had to satisfy three criteria:

- It should tie in with IID
- It should work well with changing requirements
- It should be suitable for a small team (5-10 members)

We concluded that Agile Project Management (APM) methodologies best suited our requirements. Out of the existing APM methodologies, we decided to use the eXtreme Programming methodology.

Extreme Programming

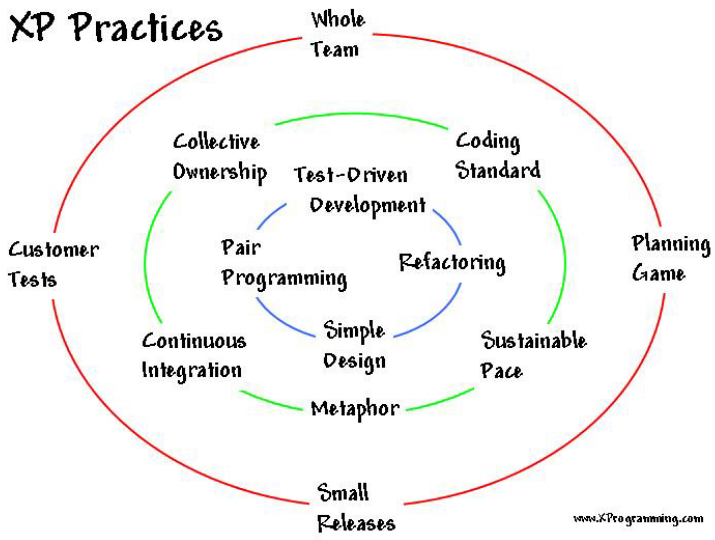

Figure 38 – Extreme Programming Practices (source – Xprogramming.com)

Although not all the features of XP were suitable for us, following are some of the most core practices, which led us to adopt XP:

- **Whole Team**

  All the contributors to an XP project sit together, members of one team. There is commonly a coach, who helps the team keep on track, and facilitates the process. There may be a manager, providing resources, handling external communication and coordinating activities. None of these roles is necessarily the exclusive property of just one individual: Everyone on an XP team contributes in any way that they can. The best teams have no specialists, only general contributors with special skills *[21]*.

  Team spirit was very important in order that we achieve our full potential able to gain something technically and personally. Throughout the project development period all five of us worked together, looked out for and extended a helping hand to each other. Cecilia as our guide ensured that we were on track and encouraged us to stretch ourselves. Kavita as our manager scheduled regular meetings, delegated tasks and coordinated the team's activities. However, there were no clear task descriptions for each individual. We had no "boss" and hence it was important for us to work together and take ownership and responsibility. The deliverables are a result of proactive and team attitude, where each one contributed in any and every way they could.

- **Small Releases**

  XP teams release tested and running software frequently *[21]*.

  This ties in very well with the IID methodology of having small slices. Each slice was a release in itself as it was tested rigorously and ready for use. This enabled us to estimate

the time required for future releases and have working software at any point in time. When things went wrong, we could always pick up the last version of working software and continue from there.

- **Simple Design**

  XP teams build software to a simple design. They start simple, and through test-driven development and design improvement, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system *[21]*.

  In keeping with both IID and XP practices, we tried to keep our design simple and portable, from simulation to prototype. The design evolved and the final version looks a lot different from the first, however, these changes were iterative and hence, at no stage did we feel that things were getting over complicated because of the design.

- **Design Improvement**

  XP focuses on keeping the design simple and uses a process of continuous design improvement called *Refactoring*. The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code, while lowering the "coupling" *[21]*.

  While implementing the algorithm, we had to ensure as far as possible that the design is reusable for the prototype. Since, we were scheduled to start the prototype at a later time, it was important that we ensure smooth development in order to complete it in time. Hence, we endeavoured to keep the design good and simple by continuously removing duplicates and decoupling, during integration. This tied in well with IID as IID also aims at improving design with each iteration. This also enabled us to maintain a steady development pace. However, in order to ensure improvement in design we had to couple this practise with test-driven development.

- **Test-driven Development**

  Extreme Programming is obsessed with feedback, and in software development, good feedback requires good testing *[21]*.

  This again tied in well with IID, as IID also demands continuous feedback so that things may be improved. As extreme programmers, we were all testers. Each time a new piece of code was developed, not only unit testing but also integration testing was performed. Test cases followed the use cases exactly. Please see appendix for test cases. This enabled us to get immediate feedback on what was working and what was not and what could be improved.

- **Continuous Integration**

  Extreme Programming teams keep the system fully integrated at all times *[21]*.

  Although we try and keep the individual slices independent from each other, they will have to be integrated for the final product. Also, although the individual slices are

tested thoroughly, there may be problems when they are integrated. Hence, each new slice was developed, tested thoroughly, integrated with the rest of the code and again tested thoroughly. This enabled us to spot bugs at an early stage and always have working software ready. In short, we always had something to show even if some features did not work or were not ready eventually.

- **Pair Programming**

  All production software in XP is built by two programmers, sitting side by side, at the same machine *[21]*.

  Since, we were a team of five, we decided to have two sub teams, comprising of two and three. The team with three members had to work on a more difficult or time consuming module to balance things out. This resulted in better design, code, testing and integration. It also fostered team spirit and camaraderie. It also ensured that the knowledge on any particular module is shared by at least two people. Hence, this also enabled collective code ownership.

- **Collective Ownership**

  On an Extreme Programming project, any pair of programmers can improve any code at any time *[21]*.

  There were times when one person from one of the sub teams was busy resolving something else, however, this did not stall the development process as any part of the code was known to at least two people.

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

8.4 Team Structure

Dr. Cecilia Mascolo – Supervisor

Cecilia was supportive and encouraging throughout. She gave our ideas and thoughts direction. We appreciate her help and guidance.

Kavita Gupta – Project Manager

Kavita scheduled both team meetings and meetings with the supervisor, delegating tasks and co-ordinating the team's activities.

Liang Chen (Ben) – Chief Software Engineer

Ben was responsible for the design. He ensured that we revisited our design at regular intervals and ensured right refactoring.

Waigit Teo – Configuration Manager

Teo set of the central CVS repository. He ensured that the repository was updated with the latest documents, minutes of meeting and working code.

Christos Savvidis – Chief Test Engineer

Christos designed test cases. He ensured that unit and integration testing was done thoroughly and regularly.

Haitchen (Sky) – Risk Manager

Sky was responsible for tracking risks and updating the risks. He flagged issues and encouraged others to do so, so that appropriate mitigation strategies may be adopted.

We defined roles for each of us so that someone takes the "ownership" of each task and avoid chaos. This also enabled us to be professional and get a feel of the real world. We gained some good project management skills. However, each of us were involved in every activity and contributed in whatever way we could. For example, we all helped Christos to design test cases. However, it was his responsibility to seek help and ensure that it is done.

All of us participated in requirements engineering and as extreme programmers all of us were application programmers and testers.

The sub teams of 2 and 3 changed depending on skills and interest. Once, the implementation of the simulation was complete, Ben, Kavita and Sky focussed on the prototype and Teo and Christos focussed on collecting and analysing simulation results. Once the prototype reached a comfortable stage, Kavita focussed on the report, while the others continued with their respective tasks.

8.5 Team Communication and Status Monitoring

We planned to have a weekly meeting among ourselves and with the supervisor. In the initial stages during the design of the algorithm, we met more frequently. We started meeting and discussing since January '04, even with our supervisor. Later, during the implementation stage, since we worked together, we had whenever required either to discuss issues or make decisions. Hence, we continued to meet our supervisor weekly but had more meetings among ourselves before we met Cecilia to ensure that we discussed everything with her. At the end of every meeting, Christos produced minutes of meeting. This included the issues discussed, goals for the coming week and who needs to do/send what. This enabled the team to focus on what needed to be done and by when.

This way everyone was kept abreast of the latest developments and upcoming tasks. A mailing list was created, which was used to send minutes of meetings, to enable effective communication about progress of individual tasks between team members and also with the supervisor.

We also set up a forum, which we used to post research material and communicate ideas "asynchronously". Hence, the forum was like a repository of information, which was easily accessible to everyone from everywhere, including our supervisor.

Since, this is a research project, all of us had strong opinions and decisions were taken on a consensus basis rather than dictated by any member of the team. There were informative and friendly debates in which we discussed serious issues and came up with creative ideas. This enabled all of us to learn from each other.

8.6 Risk Management

We identified few potential risks, prioritised them and identified prevention and mitigation strategies.

Description	Priority	Mitigation Strategy	Owner
Not clear understanding of requirements	High	- Requirements analysis - Extensive research - Continuous integration - Test-driven development - Regular review of requirements	Ben / Christos
Deviation from goals	High	- Keep track of the requirements	Kavita / Sky
Bad design of algorithm	High	- Fast prototyping - Incremental Iterative Development - Continuous integration - Simple design - Feedback through simulation	Kavita
Progress not according to schedule	Medium	- Progress tracking - Learn from mistakes that cause delay - Teamwork	Kavita
Selection of inappropriate tools	Medium	- Extensive research and trial	Ben / Christos / Teo
Absence of supervisor / team members	Medium	- Inform team and keep track of absence well in advance - Plan accordingly - Work extra hours	Sky
Insufficient technical competency	Low	- Practise the tools prior to implementation	Sky

Figure 39 – Risk Management

8.7 Project Milestones

Project milestones for each month were as follows:

8.7.1 May

This month will be dedicated to *research and background reading and requirements gathering*. All the team members aimed to understand the JMS specification and read relevant papers to accumulate information and come up with novel ideas for the algorithm.

This phase had begun in January and gathered momentum in May. Once the skeleton of the algorithm was in place, we will deal with the details of each aspect according to the requirements, keeping in mind the various challenges and constraints of MANETs. In order to focus adequately on various aspects, we will divide ourselves into sub teams, with each sub team focussing on a particular aspect.

We also have the Z26 presentation as one of the milestones this month. Hence, the requirements gathering will help us present our ideas clearly during the presentation.

At the end of this phase, we will have clearly defined functional and non-functional requirements, which will be the same for both the algorithm and the prototype. Functional requirements mainly included the functionality proposed by the JMS specification. Non-functional requirements included reliability, scalability etc. We will also have the design of the algorithm ready with all the details of each step for the implementation to begin.

8.7.2 June

We will *implement the algorithm on OmNet++*. Again, following the practices of XP, we will work in pairs, have weekly/bi-weekly milestones and continuously integrate and test the algorithm for compliance with the requirements. Requirements and design will regularly be revisited to make modifications and enhancements. We will allocate more time for the core-model and comparatively less time for the Publish/Subscribe and Point-to-Point models. We will work together as a team to ensure we have a successful release.

At the end of this phase, we will release a fully working model of the algorithm, ready to be used for data collection. At this point, we will decide whether or not to go ahead with the prototype implementation. As the algorithm and simulation analysis, which proves its effectiveness is our main focus, we will continue to dedicate resources for the simulation implementation, data collection and analysis if we are not in a comfortable position. We realise that this part is very risky and we want to have at least one concrete deliverable rather than having two “not so good” deliverables.

8.7.3 July

We will start with *data collection and analysis*. We will evaluate which parameters will reflect the performance of our algorithm the most and collect data accordingly. Continuous feedback from Cecilia and Mirco will be top priority in order to ensure that we obtain reasonable results portray the right picture. This is important, as none of us are familiar with simulation.

If we decide to go ahead with the prototype, we begin its *design and proceed with its implementation* in parallel with data collection and analysis.

At the end of this phase, we will have graphs ready to be analysed and formally documented. The prototype should be in its last stages.

8.7.4 August

We will gather our documentation and work on our final report. We will finish collecting data and focus on analysis. Finishing touches, enhancements and bug fixing to the prototype

can go on in parallel. We will also take time to evaluate the course of the project as a whole and the effectiveness of our team organisation. We will also develop plans and suggestions for future work.

At the end of this phase we will release the prototype, complete our analysis and have the report ready for submission.

9. Conclusion

9.1 Introduction

In this chapter, we discuss related research, how they compare with JIMS, critical assessment of our work and what further enhancements may be made.

9.2 Related Research

We discuss two related research works and followed by a comparison with JIMS.

9.2.1 Epidemic Messaging Middleware for Ad hoc Routing (EMMA) [33]

The epidemic routing is to distribute application messages to hosts, called carrier, within connected portions of ad hoc networks by anti-entropy session (to avoid redundant connections, each host maintain a cache of hosts that is has spoken with recently). In this way, messages are quickly distributed through connected portions of the network. Epidemic routing then relies on carriers coming into contact with other connected portions of the network by node mobility. At this point, the message spreads to an additional island of nodes. Through such transitive transmission of data, messages have a high probability of eventually reaching their destination.

9.2.2 JMS implementation of On-demand Multicast Protocol (JOMP)

JOMP is an application-level implementation of the basic On Demand Multicast Routing Protocol (ODMRP). This ports the HP-MS product, implemented for wired networks on to MANET environments by implementing the ODMRP underneath.

ODMRP is an on-demand multicast protocol. This means that a node will only attempt to create a route through the network once it has data to send (delay associated with creating a route to a multicast group), which obviously reduce channel overhead and improve scalability. A soft-state approach is taken to maintain multicast group members. No explicit control message is required to join or leave the group. The protocol, however, may suffer from excessive flooding when there are a large number of multicast senders.

When a node has data to send, it will attempt to create a mesh of forwarding nodes, which are responsible for forwarding multicast data on shortest paths between any member pairs. Providing multiple paths by formation of mesh configuration makes the protocol robust to mobility. Alternate route enable data delivery in terms of mobility and link breaks while the primary route is being reconstructed. These forwarding nodes re-broadcasts any packet it receives if it is not a duplicate and multicast group has not expired in order for the packet to reach all interested multicast receivers. The multicast mesh is created through a reply-response phase, which is repeated at intervals to keep the routes to the multicast receivers fresh. Network host running ODMRP are required to maintain some data structures like route table, forwarding group table, and message cache.

9.3 Comparison

In this section we present a comparison between JIMS, EMMA and JOMP.

- **Message Spread**

  JIMS: Node mobility and resources are intelligently used to spread messages. Node profiles are defined; subscribers and sociable nodes are primarily responsible for spreading messages.

  EMMA: Any pair of nodes that come into contact with each other exchange messages.

  JOMP: A mesh of forwarding nodes is created. These nodes are responsible for forwarding multicast data on shortest paths between any member pairs.

- **Anti-entropy**

  JIMS: Anti-entropy is supported by sending meta-data consisting of a bit descriptor.

  EMMA: Anti-entropy is supported by sending summary vectors. To avoid redundant connections, each host maintain a cache of hosts that is has spoken with recently

  JOMP: Does not support anti-entropy

- **Message property**

  JIMS: In keeping with the JMS specification, message fields consist of: priority, hop count, and time to live, which reflects the importance and freshness of a message

  EMMA: Messages contain hop-count, which reflects priority

  JOMP: No approach to differentiate messages

- **Buffer management strategy**

  JIMS: A message will be deleted from the buffer once its time to live expires. If the buffer is full when a message is received, the message with a minimum state (means least important) according to message priority and time to live will be replaced with a new incoming message if it has higher state. Duplicates are avoided by scanning the buffer before inserting a message and discarding duplicates. This ensures that the priority and "persistent" properties of the messages are according to the JMS specification.

  EMMA: Adopts a simple FIFO strategy.

  JOMP: Uses message cache to detect duplicates.

- **Path maintenance**

  JIMS: No path maintenance required, as message delivery is one-hop. Minimum state maintenance required.

  EMMA: No path maintenance required, as message delivery is one-hop.

  JOMP: The routes to the multicast receivers are kept fresh. The reply-response phase may suffer from excessive flooding when there are a large number of multicast senders. Network host running ODMRP are required to maintain some data structures like route table, forwarding group table, and message cache.

  It is clear that JIMS has a more intelligent mechanism for spreading messages compared to both EMMA and JOMP. This and a clever strategy of sending meta-data periodically, saves bandwidth and thus increases the survivability of the network. Also intelligent buffer management strategy and incorporation of all the message fields as specified in the JMS specification enables JIMS to closely implement the message priority and delivery mechanism semantics as in the specification. However, this may mean a higher latency as compared to EMMA and JOMP.

9.4 Critical Analysis

Throughout the project, the team always kept the primary goal in mind, implementation of a reliable middleware for MANETs. The wide scope and time constraints led us to make some assumptions and chalk out feasible goals. We have tried to present an objective assessment of our successes and failures in the next few paragraphs.

9.4.1 Success

We have addressed most of the challenges imposed by MANETs in terms of resource constraints and achieved a high delivery ratio and message ratio for both RWP and GMM. JIMS is both “system aware” and “context aware”. We leverage both resources and mobility pattern of nodes intelligently by introducing the concept of “sociable nodes”. This saves both memory and bandwidth in the network and increases network survivability. Intelligent use of Jacobson’s formula allows us to discover sociable nodes in a computationally inexpensive manner. Intelligent exchange of meta-data enables us to reduce the control message traffic in the network. Our buffer management strategy is novel in terms of calculation of message state and compliance with the JMS specification. Rigorous research and effective brainstorming resulted in an overall robust and “intelligent” algorithm. We have not only proved our algorithm through simulation results but also by implementing a prototype, which proves the practical application of the algorithm. Our work is of high significance as to our knowledge there is no other product or research available, which has implemented JMS semantics in MANETs this extensively.

9.4.2 Failure

Our inexperience with simulators and simulation analysis took its toll. A lot of data had to be recollected because of things going wrong in terms of simulation time, parameter settings etc.

This led to a lot of trial and error, although time allocated for this phase was relatively longer than that for other phases. We do realize in addition to allocating extra resources, more brainstorming and discussions for the parameters could have avoided this. Ideally, we would have liked to present more analysis and different perspectives through more graphs. However, we believe, we have been able to present the most important ones.

Also, we had initially aimed to port the prototype on an emulator or actual mobile devices. However, lack of time and experience prevented us from doing so.

9.5 Future Work

Our work may be enhanced to incorporate the following:

9.5.1 Power Management

Although we have done our best to manage memory constraints in mobile nodes and bandwidth constraints in the network efficiently, we have not paid much attention to power management, as it is a huge research area in itself. Hence, power management may be easily plugged into the algorithm, as we have already taken power threshold into consideration, although it is hard coded.

9.5.2 Security

The research community has realized that MANETs cannot be applied widely if secure mechanisms are not able to create a sense of confidentiality. The areas that are currently under research follow in the directions of key management and intrusion detection *[22]*. It is believed that the security requirements of a networked system such as availability, confidentiality, integrity, authentication and non-repudiation should be transferred to MANET environments, in order for them to be used and of course lead to their success.

The way JMS focuses on security mechanisms for wired networks is described in *[23]* where an example of an XML router is used and refers to the direction of document signing and encryption of XML documents. Further details are out of scope of our work.

However, we do realize that for the commercial application of JIMS it is important to plug in security mechanisms.

9.5.3 Transaction Support

JMS sessions can be optionally defined as transaction bound. Transacted sessions can treat a set of messages sent and received as a single atomic unit. This means all the messages in a transaction are either processed completely or not at all *[23]*.

This one feature of JMS is not implemented in the existing JIMS algorithm. However, we have thought about it and believe the Associativity Based Routing (ABR) *[24]* algorithm semantics may be used to implement this. A publisher could use the most “stable” sociable neighbor so that all the messages can be passed on. Control messages will have to be implemented to indicate the start and end of a transaction. These have to be used by both the publisher and the sociable node while interacting with the subscriber. A time out will have to be implemented, which will be effective from the time the “start” control message is received.

If all messages are received within that time out, the sociable node goes off to look for the subscriber else all the messages received is discarded (rollback). An ACK/NACK mechanism may also be needed to ensure reliability and facilitate retransmission. Many of the semantics implemented for the Point-to-Point model may be used for transactions.

9.5.4 Simulation

While we have measured the affect of the most important parameters on the performance of the algorithm, it is worth investigating the bandwidth saved due to various steps taken to increase the survivability of the network. Also, the Point-to-Point model may be investigated further for non-persistent messages.

9.5.5 Prototype

The prototype may be enhanced in two ways:

- The Point-to-Point model may be implemented.
- The prototype may be ported on an emulator and eventually on actual devices.

10. Appendix I – Simulation files, Compilation and Execution

In this section we describe the contents of each of the simulation files, followed by the steps to compile and execute the simulation.

Node.cc and Node.h

Node.cc contains the implementation of the algorithm. Most of the code is common to both Publish/Subscribe and Point-to-Point models. However, there are some parts which are model specific.

In Node.h file there are also many different parameters that are injected in our simulation implementation, such as the total simulation time, the time that a message stays in the queue of a node, the queue size, the maximum hop count allowed for a message, the maximum sequence number. Also, in this file there is the declaration of the class Node, that in terms of the OMNeT++ semantics it is a *module*.

Engine.cc and Engine.h

These files contain the implementation of the mobility models, Random Waypoint and Group Mobility. The execution of these files in the graphical user interface (Tkenv) of OMNeT++ shows the simulation area with nodes moving around following the patterns of the mobility model they are following. Both the engines were provided by Mr. Mirco Musolesi.

Omnetpp.ini

This is the configuration file for the simulation runs. The description below has been taken by *[7]*. The format that we have used for a case of 64 nodes and 30 runs is as follows:

```
[General]
sim-time-limit= 7200.0
ini-warnings = yes
output-scalar-file = Scalar.sca
[Run 1]
network = mobilityscenario64;
mobilityscenario64.engine.numRun=1;
gen1-seed=1227283347
.
.
.
```

gen20-seed=33648008

.

.

.

[Run 30]

network = mobilityscenario64;

mobilityscenario64.engine.numRun=1;

gen1-seed=1227283347

.

.

.

gen20-seed=33648008

[Tkenv]

default-run=3

runs-to-execute = 1-10

animation-speed=1.0

update-freq-fast=100.0

update-freq-express=100.0

extra-stack=32768

[Cmdenv]

runs-to-execute = 1-30

module-messages = no

verbose-simulation = yes

The `[General]` section applies to both Tkenv and Cmdenv, and the entries in this case specify that the network should be simulated and run for 7,200 simulated seconds, and the results should be written into the `Scalar.sca` file. The ini-warnings Helps debugging of the omnetpp.ini file. If turned on, OMNeT++ prints out the name of the entries it that it wanted to read but they were not in the configuration file.

The [Run N] sections contain per-run settings. These sections may contain any entries that are accepted in other sections. In each run the name of the network being simulated is called mobilityscenario64. Also the seeds for the given random number generator are given, different for each run. The animation-speed parameter defines the message flow of the animation, thus it reflects the tasks being shown in the graphical user interface window per simulation unit time.

Tkenv supports interactive execution of the simulation, tracing and debugging and is has mostly been used in the development stage of a simulation, since it allows us to get a detailed picture of the state of simulation at any point of execution and to follow what happens inside the network. The [Tkenv] section contains Tkenv-specific settings, where default-run specifies which run Tkenv should set up automatically after startup. If there's no default-run= entry or the value is 0, Tkenv will ask which run to set up. The parameter runs-to-execute defines which simulation runs should be executed. The update-freq-fast parameter specifies the number of events executed between two display updates when in *Fast* execution mode. It is worth mentioning that this mode is one of the options that the user has. The other options are *Step, Run* and *Express*. Similarly, the update-freq-express is the same as previously, but for the *Express* mode. The parameter extra-stack specifies the extra stack that is reserved for each execution of the activity function, that has been implemented in th file Node.cc.

Finally, the [Cmdenv] section gives the opportunity of command-line execution of the simulation. The parameter module-messages defines whether the print statements of the simulation program should be printed in the command line, the verbose-simulation indicates whether the print statements that show the simulation time with the particular event happening should be printed or not.

mobMessage.msg

This file contains the fields that are contained in each message type. For example, a Beacon message is described as:

```
message Beacon {
        fields:
                int senderNodeID;
                int nodeType;
                double socValue;
                bool topicID[2];
};
```

which means that the beacon contains the id of the sender node, the type of the sender node (sociable node, publisher or subscriber) and the topic in which it is interested in. After the

compilation of the whole program using *make*, the files mobMessage_m.h and mobMessage_m.cc are created. For the above example, the former contains the declaration of the class Beacon with the protected variables as defined in the fields and the relevant getter and setter methods. In the mobMessage_m.cc, the implementation of these methods is being made automatically.

Mobility.ned and SocialMobility.ned

These two files define the network topology for the Random Waypoint and Group Mobility Model accordingly, by using the NED (Network Description) Language. These files contain a description of the different modules that are involved in the construction of the network, such as different mobility scenarios that reflect the number of nodes in the network and the different nodes. Also, the interconnection of these modules is made by the use of *gates.* More details about the implementation using the NED language are out of scope.

Simulation Compilation and Execution

Following are the steps used for the compilation and execution of the simulation:

- The Makefile needs to be created at first, along with defining the running environment. The command used is (in UNIX environments):

  opp_makemake –f –u Cmdenv or opp_makemake –f –u Tkenv

- Then the make command is used in order to execute the Makefile. In this step, all the files are compiled using automatically the appropriate compiler, as well as different files (such as mobMessage_m.cc) are generated that can be used by the main code (Node.cc).

- In this step and if there were no program errors in the previous steps, we are ready to run the simulation, using the command (in UNIX environments):

  ```
  ./SimulationExecutable
  ```

11. Bibliography

1. Gregory D. Abowd and Elizabeth D. Mynatt. Charting Past, Present, and Future Research in Ubiquitous Computing. Georgia Institute of Technology. [1] 8
2. Internet Engineering Task Force (IETF), Mobile Ad Hoc Network (MANET) working group charter, http://www.ietf.org/html.charters/manet-charter.html 8
3. Wolfgang Emmerich. Engineering Distributed Objects. John Wiley and Sons, 2000. [3] 8
4. OMG. Common Object Request Broker Architecture (CORBA). http://www.omg.org/ [4] 8
5. Guanling Chen and David Kotz. Asurvey of Context – Aware Mobile Computing Research. Department of Computer Science, Dartmouth College. 8
6. Mirco Musolesi, Cecilia Mascolo, and Stephen Hailes. Adapting Asynchronous Messaging Middleware to Ad Hoc Networking. Department of Comupter Science, University College. 9
7. Andras Vargas. The OMNeT++ discrete event simulation system. http://www.omnetpp.org/ 9
8. Stephen Ferg. Introduction to Event-Driven Programming, 2003-02-16. http://www.ferg.org/projects/ferg-event_driven_programming.html 14
9. Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. Java Message Service Specification Version 1.1 Sun Microsystems, Inc., April 2002. http://java.sun.com/product/jms/ 15
10. Musolesi, M., Mascolo, C., Hailes, S., Raimko, A.. EMMA: Epidemic Messaging Middleware for Ad hoc networks. 33
11. Jen – Yeu Chen, Yen – Shiang Shue, Hakeem Ogunleye, and Saurabh Bagchi. A comparative study on data fault tolerant requirement for data propagation in sensor networks. Department of Electrical and Computer engineering, Purdue University. 34
12. Jen – Yeu Chen, Yen – Shiang Shue, Hakeem Ogunleye, and Saurabh Bagchi. A comparative study on data fault tolerant requirement for data propagation in sensor networks. Department of Electrical and Computer engineering, Purdue University 36
13. Einar Vollset, "Extending an enterprise messaging system to support mobile device", Msc thesis, University of Newcastle upon Tyne, September 2002. 39
14. Sangkyung Kim and Sunshin An. History – Aware Multi-path Routing in Mobile Ad – Hoc Networks. Service Development Lab., KT. Department of Electronic Engineering, Korea University. 42
15. Tracy Camp, Jeff Boleng and Vanessa Davies, A Survey of Mobility Models for Ad Hoc Network Research, 10 September 2002 44
16. D.Johnson and D.Maltz. Dynamic source routing in ad hoc wireless networks. In T.Imelinsky and H.Korth, editors, Mobile Computing, pages 153-181. Kluwer Academic Publisher, 1996. 44
17. Mirco Musolesi, Stephen Hailes and Cecilia Mascolo, An Ad Hoc Mobility Model Founded on Social Network Theory, June 2004 ....44, 48
18. Matthias Grossglauser and David Tse, Mobility Increases the Capacity of Ad-hoc Wireless Networks,2001 48
19. Matthias Grossglauser and David Tse, Mobility Increases the Capacity of Ad-hoc Wireless Networks,2001 61
20. M.C.Robert, Iterative and Incremental Development, 1999 76
21. http://www.xprogramming.com/ 80
22. Buchegger, S., Le Boudec,. J.-Y. (2002). Nodes Bearing Grudges: Towards Routing Security, Fairness and Robustness in Mobile Ad Hoc Networks. In Proceedings of the 10th Euromicro

Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain. pp 403-410.....88
23. Giotta, P., Grant, S., Kovacs, M., Maffeis, S., Scott Morisson, K., Raj G.S., Kunnumpurath M. M. (2000). Professional JMS Programming. Wrox......88
*24. S. J. Lee, M. Gerla, C.K. Toh, A simulation study of Table-driven and On-demand Routing Protocols for Mobile Ad hoc Networks, University of California, Georgia Institute of Technology*88
26. W. Ye, NS-2 Simulator Manual, Chapter 17, pages 164 - 168, www.isi.edu/nsnam/ns/doc/index.html.....45
27. Mario Bisignano, Andrea Calvagna, Giuseppe Di Modica and Orazio Tomarchio, Expeerience : a Jxta middleware for mobile ad-hoc networks, In Third Int. Conference on Peer-to-Peer Computing (P2P2003), Linkoping (Sweden), September 2003......8
28. Ad hoc Networking :: Applications,.....8
29. Qun Li and Daniela Rus, Communication in disconnected ad hoc networks using message relay, 17 October 2002......13
30. Einar Vollset, Dave Ingham and Paul Ezhilchelvan, JMS on Mobile Ad-hoc Networks, In Personal Wireless Communications (PWC), 2003......13
30. Stefan Aust, Daniel Proetel, Carmelita Görg and Cornel Pampu, Mobile Internet Router for Multihop Ad hoc Networks, July 2003......13
31. Stephen S. Yau and Fariaz Karim,A Lightweight Middleware Protocol for Ad Hoc Distributed Object Computing in Ubiquitous Computing Environments, 14 May 2003.....15
32. K.Fall, A Delay Tolerant Network Architecture for Challenged Internets, Intel Research, Berkeley.....48
33. Abbas El Gamal, James Mammen, Balaji Prabhakar, Devavrat Shah,(Stanford University), INFOCOM 2004.....48