2002

Throughput control for a many-to-many system

Ravi Nareppa

M.S. Computer Science Thesis
Rochester Institute of Technology
Rochester, New York

Throughput Control for a Many-to-Many System

By,

Ravi K Nareppa

The M.S. Computer Science Thesis of
Ravi K. Nareppa is approved.

Faculty Chair: Prof. Hans Peter Bischof

Signature: _____ Date: Dec/16/2002

Faculty Reader: Prof. Alan R Kaminsky

Signature: _____ Date: 16-DEC-2002

Faculty Observer: Prof. Paul Tymann

Signature: _____ Date: 16-Dec-2002

Copyright Notice

Throughput Control for a Many-to-Many System

I, Ravi K. Nareppa, hereby grant permission to the RIT Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 12/17/2002 Signature of Author: _____

Acknowledgements

I would like to thank Prof. Bischof and Prof. Kaminsky for accepting me as their graduate student, advising me during the course of the project and for guiding me to successfully finish the project and thesis. I would also like to thank Prof. Alan Kaminsky for helping me understand the Many-to-Many Protocol, helping me with the project architecture and for providing me with resources to complete my thesis.

I am very grateful to Xerox Corporation for sponsoring my graduate program at RIT and bearing all the expenses associated with it. I would like to thank my managers at work, Mr. Ken Buck, Mr. Ajay Amin, Mr. Mark Judd, Mr. Doug Kowiak and Mr. Mark Brotzman for supporting me in this endeavor for the past couple of years. At work, I would like to thank Juan Acebo and Chris Villone for providing suggestions for my school projects and patiently listening to my questions.

I am grateful to the Computer Science department for providing the excellent computing resources for my project and to RIT for providing a very good library that helped me complete the research work for my thesis. I would like to offer a special thanks to the author of the plotting package that was used in this project, Mr.Joseph A. Huwaldt.

Most of all, I would like to thank my wife, Divya, for her patience and support, without which it would have been impossible for me to have finished my second graduate program. I would also like to thank my family members back in India for their moral support. I would like to thank Kiran Hegde and Archana, very good friends at and outside school for their moral, social and academic support. Many thanks to Kaesy Sanders and Anil Menon for proof reading my thesis.

Table of Contents

Table of Figures

1 Introduction

A wireless local area network (WLAN) is a data communication system that is much more flexible than a wired communication environment. It can be argued that wired environments provide very good reliability and bandwidth, and that their data transmission rates are about 10 times faster than WLANs. However, when it comes to flexibility, a wired network simply does not provide many options.

Some of the advantages that WLANs offer over their wired counterparts can be listed as follows:

1. Easy setup and fast installation because no cabling is necessary. Installation can be done in places where cabling would be expensive or impossible.
2. The overall life-cycle cost of a WLAN can be cheaper than a wired system, especially if a lot of relocation is necessary.
3. The most important advantage of a WLAN is its mobility and flexibility, making it the most sought after in warehouses, hospitals, training centers, disaster relief etc.
4. The wireless networks are more configurable than a wired network topology. The WLANs can be configured to meet specific needs of applications and then re-configured very easily if the requirements change.

Hence we can see that a WLAN can provide the functionality of wired LANs, without the physical constraints of the wire itself. Unfortunately, all these benefits come with some negative characteristics as well, as listed below:

1. A typical wireless network could be as much as 10 times slower than wired networks. One of the reasons for this is that communications are unreliable and prone to interference.
2. The initial cost of setting up the system could be very high, because several transmitters may be required. Careful planning is necessary to help keep initial costs down.
3. Security is a major issue with WLANs because it could be very easy for others to intercept signals in the air. Hence good encryption technologies are required for WLANs.

Since the speed and reliability of wireless communications is of major concern, these are the areas that will be tested in this project for the M2MP, which is supposed to run on WLANs. A WLAN will be simulated in a wired network so that speed control (throughput) and bottlenecks can be simulated. The advantage is that testing could be easily performed in a controlled environment.

1.1 The Anhinga Project

The Anhinga project is a project of the computer science department at the Rochester Institute of Technology (RIT). The researchers for this project are the staff members and students of the computer science department at RIT. The purpose of the Anhinga project is to develop an infrastructure for distributed communication and collaboration that could be used in small mobile wireless devices. The wireless devices

could run applications that operate on spontaneously formed, ad-hoc, peer-to-peer networks of the devices themselves, without requiring central servers or a fixed network infrastructure. The currently available network protocols cannot support this environment and hence the Many-to-Many Protocol was developed. The Anhinga Infrastructure[36] is shown in Figure 1.

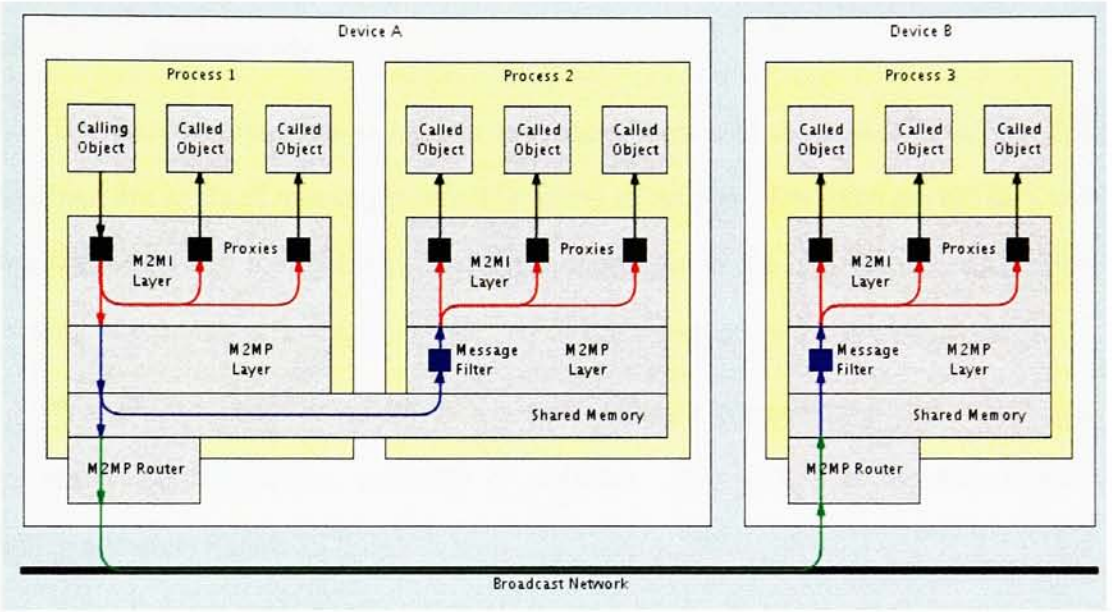

Figure 1: The Anhinga Infrastructure [36]

1.2 The Many-to-Many Protocol

A new network protocol is being developed at RIT as part of the Anhinga project and is called the Many-to-Many Protocol (M2MP)[1][4][35]. This M2MP will ultimately be deployed in an unreliable wireless environment but is currently being developed in a reliable wired environment.

The M2MP is designed specifically for a wireless network where devices could enter and exit all the time. Some of the basic characteristics of M2MP are:

1. No device addresses need to be specified for this protocol.
2. Messages are always broadcasted to all the proximal devices since broadcasting to all devices is as easy as transmitting to one device.
3. Most of the time the message delivery is reliable, but sometimes some packets might get lost. The M2MP, however, is not responsible for reliable message delivery. The application layer is supposed to take up that responsibility.

The applications have to register message filters with the M2MP to let the M2MP know about the kinds of messages that it is willing to receive. For each packet that comes in, the M2MP tries to match the packet's message filter to the one that the application is expecting. If a match is found, the M2MP sends the message to the application layer.

These characteristics result in a simple network protocol that does not have to worry about retransmissions, reliability or ordering. The M2MP architecture high-level view[5] is shown in Figure 2.

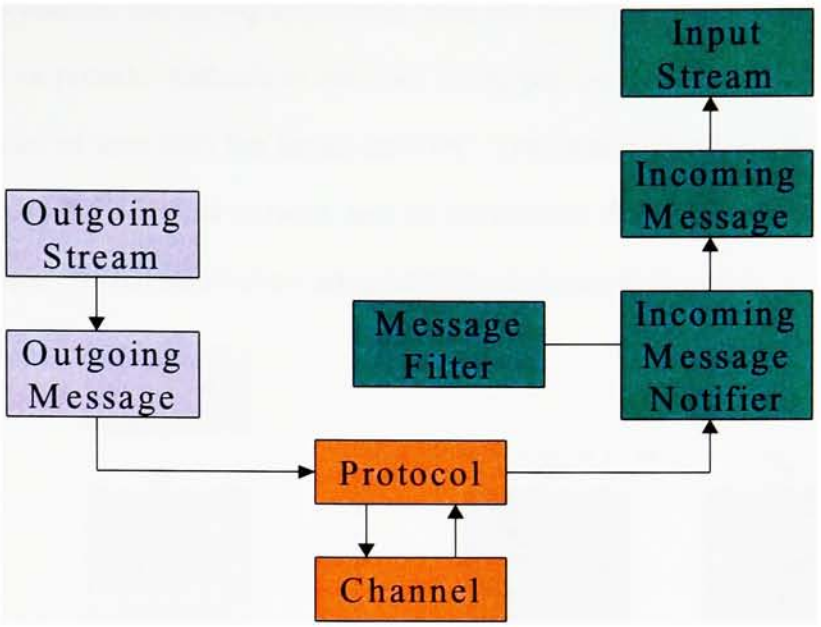

Figure 2: M2MP high-level architecture overview [3]

1.3 Many-to-Many Invocation

Many-to-Many Invocation (M2MI), also developed by the computer science department at RIT, is a new way for either wireless or wired devices to communicate in a wireless proximal ad hoc network[1][3][5][35]. An M2MI based application broadcasts a series of method invocations, which are received and executed by many target devices simultaneously.

1.3.1 M2MI Design Overview

M2MI uses the Many-to-Many Protocol (M2MP) to broadcast its messages to the proximal devices. An application using M2MI invokes a method declared in an interface. The interface's methods can take arguments but should not return a value or throw exceptions.

The M2MI creates proxies (stubs and skeletons) at runtime unlike the existing remote method invocation (RMI) systems that need to deploy the proxies ahead of time.

Due to this feature, the calling application does not need to know the identities of the target application or provide methods to discover the target devices. No connections need to be set up ahead of time with the target devices. The calling devices will broadcast method invocations to the proximal network and all the devices that implement those methods will execute them. A chat application using M2MI[5] is shown in Figure 3.

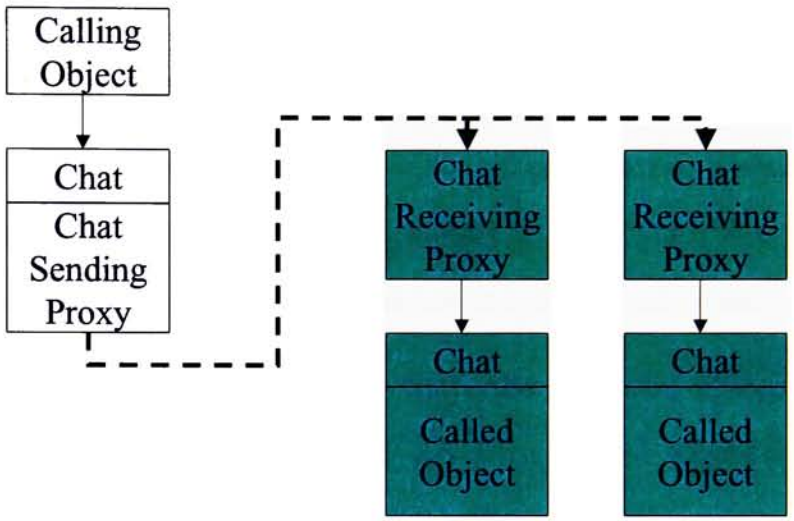

Figure 3: M2MI Chat Communication

M2MI uses something called *handles* that are actually references upon which invocation is to be performed. An application using M2MI can request the M2MI layer to create an *omnihandle,* which would mean that "every" object implementing that particular interface would execute this method when called. M2MI also provides what's called a *unihandle,* and one particular object that is associated with this unihandle will perform the specified method.

M2MI uses *Service Discovery* so that applications can send out an *omnihandle* requests whenever necessary, asking for service providers like printers, scanners etc. If a provider is present in the proximal network, it will reply with a *unihandle* confirmation. Now the user application can contact the service provider using its *unihandle.*

1.3.2 M2MI Advantages

The new and innovative design of M2MI provides numerous advantages[5] to the devices that implement it. Some of the advantages are listed in this section.

- M2MI can efficiently run on small devices that use battery power, since the device need not be constantly connected to the network. Due to its size and efficiency, M2MI is a very good candidate for embedded devices with limited memory and CPU capacity.
- M2MI can be used to run collaborative systems without a need for any central servers that could be a bottleneck. Since M2MI can run on a wireless environment, devices can be added and removed continuously.
- M2MI based applications do not require the overhead of implementing routing algorithms or networking protocols, hence removing network administration.

1.4 Overview of this study

The Many-to-Many Protocol is a wireless ad hoc network protocol that will be eventually deployed in a wireless environment, but some of the initial testing should be performed in a controlled, wired environment. This study focuses on developing a system to perform throughput tests for the M2MP in a controlled environment. The research process consists of (a) studying the existing network protocol testing tools, (b) identifying the necessity to develop a throughput control system, (c) designing and implementing the throughput control system for a many-2-many protocol, (d) designing and implementing a RemoteControl mechanism to monitor the behavior of all the applications using M2MP, (e) designing and implementing a simple application that exercises the M2MP and the

throughput control system, (f) performing tests by simulating various wireless environments using the tools developed during this research, (g) charting and analyzing the results of this research work and (h) recommending future research that could be extended from the system developed by this research.

1.5 Thesis outline

Chapter-1 provides an introduction to the Anhinga research project, Many-to-Many Protocol and Many-to-Many Invocation and an overview of this project and research. Chapter-2 gives a preview of the various mobile wireless networking technologies, and reviews some of the current methodologies for developing and testing network protocols. Chapter-3 elucidates the necessity for this research and the objectives of this research work. Chapter-4 provides detailed design descriptions of the core tools developed for this research work, including class diagrams, sequence diagrams and communication descriptions. Chapter-5 outlines the support tools developed for this project, and mentions the components that were customized for using in our research work. Chapter-6 presents some of the results of this research work, summarizes the conclusions and provides future recommendations that could be extended to this research. Chapter-7 provides the detailed usage guide, outlining the necessary steps required by a user to successfully setup and execute the throughput control system in their environment. This chapter also highlights the limitations of this system so that the user can develop a work-around. Chapter-8 provides a glossary of the terms and lists the abbreviations used in this report. Chapter-9 lists the previous research works, projects and textbooks that were referred during the course of this research.

2 Literature Review

This chapter provides an introduction to mobile wireless technologies, the advantages and disadvantages of wireless networks and the problems faced during the development of a wireless network protocol. This chapter also highlights research work being performed on distributed computing and ad hoc network protocols. A majority of this chapter discusses the existing techniques for developing and testing network protocols.

2.1 Introduction

Mobile wireless technology, though very promising and attractive, has its share of unsolved challenges[34]. Some of those challenges are reliability, integrity, network compatibility, interoperability, interference, coexistence, licensing issues, scalability, battery life, poor quality, bad coverage, limited availability, questionable security, high costs etc.

From home to enterprise, wireless communication seems to be the hype. Almost everybody owns at least one wireless device out of the myriad available in the market. Wireless devices are very popular in the work environment as well. Industries are investing a lot of money for research in wireless communications. This has resulted in many protocols being developed, some generic and some very specific for particular applications. As a result, mobile wireless is flailing around in a "technology soup". Wireless devices hardly find peers to communicate with; no clear standards have yet been established. A review of some of the various mobile networking technologies will be discussed in section 2.2. Some studies of ad hoc networks and distributed computing being done on these

networks will be studied in section 2.3. Section 2.4 discusses some ongoing research in the areas of developing and testing network protocols.

2.2 Mobile Wireless Device Networking Technologies

During the 1990s, when networking technologies for wireless devices was nascent, many proprietary networks and devices were developed. This lack of standards caused a lot of problems such as incompatible devices, expensive equipment, not much advancement due to many different protocols, and hence not suitable for widespread usage. This section gives an overview of the wireless networking technologies, the wireless protocols and the standards developed for wireless solutions.

Wireless networking can be divided into three distinct areas[33]:

1. The Personal Area Network (PAN)
2. The Wireless Local Area Network (WLAN)
3. The Cellular Network (Wireless WAN).

The networking standards and protocols used in each of these areas are discussed in the next few sections.

2.2.1 The Personal Area Network (PAN)

An example of a Personal Area Network would be a Personal Digital Assistant (PDA) connecting to a computer, or a laptop connecting to a printer in the office. Bluetooth is the technology that is extensively used in this area of wireless networking. This technology, though branded by some as complex, has some advantages such as small footprint, and the fact that it is a solid, well-tested technology. This technology has proven

its excellence as a cable substitute in devices when distances between them are a few feet. This makes it ideal for use in cars or offices.

2.2.2 The Wireless Local Area Network (WLAN)

For networks that cover an area larger than personal spaces, a WLAN provides a bunch of standards. 802.11 was one such standard that was developed by IEEE. This was one of the first internationally recognized WLAN standard that was proposed when there were a lot of non-standard protocols and devices in this sector.

Some of the advantages [10] of the IEEE 802.11 wireless standards are:

1. Appliance interoperability – Wireless appliances could be marketed by different vendors but those that are 802.11 compliant will be compatible with each other.
2. Faster product development – Since the protocol is well tested, devices can use it safely and quickly, which reduces product turn-around time.
3. Stable future migrations – By using standards such as 802.11, products will not become legacy systems and hence will help easy migrations to new developments.

Even though wireless devices offer a lot of convenience, wired LANs have been used for the last several years, and hence WLANs are un-chartered territories for many networking engineers [11]. Some of the questions the engineers need to answer before they can embrace WLAN include:

1. How can WLAN be integrated with the already existing wired networks?
2. How is multiple access of the network handled?

3. How secure is a wireless network?

4. Is it cost effective and does it solve or enhance the current problems?

Most of the solutions to these issues have been addressed in the 802.11 standard. For instance, Access Points (AP) is used as a bridge between wireless and wired LANs, hence enabling a WLAN to be integrated with an already existing wired network. The 802.11 standard specifies usage of Carrier Sense Multiple Access / Collision Avoidance (CSMA / CA) protocol, that specifies that a node will listen to the channel, and only transmit when the channel is idle. If the channel is busy, the node waits for transmission to stop, then backs off randomly to avoid collisions and then tries to transmit again after the timeout. The security issue has been addressed by the IEEE 802.11 standard by two methods: *authentication* and *encryption.* Each station needs to have authorization to communicate with another station by using an open system of requesting authentication or by using a shared key. Encryption techniques are used to provide security comparable to that of a LAN system. A cost and problem solving effectiveness of a WLAN has to be determined on an individual basis.

2.2.3 Wireless WAN (Cellular)

The cellular sector is an area where the market is huge. Consequently, the demands made by the customers in this sector are very high. Every day we see people switching between services, constantly in search of a service that can provide a decent connection and stable reception.

Companies have been developing new technologies in this sector vying for supremacy. Some of the technology buzzwords touted in this arena are CDMA, GPRS, CDPD, GSM etc. Some of these terminologies are explained below:

- GSM (Global System for Mobile Communications) – This is a set of specifications that was first developed in Europe and then brought to the USA as GSM 1900 during the early 1990's. The GSM used unproven digital technology at that point of time, as opposed to the then standard analog system. The hope was that advancement in compression algorithms and digital signal processors would improve in the future and aid the GSM technology.

- GPRS (General Packet Radio Service) – This is an add-on to the GSM system, which greatly improves its throughput capability by increasing the use of available frequency slots, and by increasing the number of radio channels being used.

- CDPD (Cellular Digital Packet Data) – This is a two-way digital transmission technology that divides data into packets to transmit over a single channel.

- CDMA (Code Division Multiple Access) – This technology encodes each piece of data transmitted over a single frequency. Each phone will decode data from the bit sequence that is unique to its connection.

- WCDMA (Wideband CDMA) – This technology is an improvement on CDMA, greatly enhancing the transmission throughput. The reason for this is that WCDMA uses higher chip rates and wider Radio Frequency channels, along with higher density modulation to increase bandwidth to 2MBS.

- G, as in 3GPP, refers to "generation", or the evolution of the cellular system.
    - 1G: First Generation analog cellular system
    - 2G: First Digital cellular system

- 2.5G: Increase in digital data rates
- 3G: Increase in functionality of digital cellular system
- 4G: Future re-architecting of digital cellular infrastructure

2.3 Mobile Ad Hoc Network (MANET)

A mobile ad-hoc network (MANET) is a WLAN wherein the network devices are part of the network only when they are in close proximity to the network, or only for the duration of the communication. A MANET is also known by other terms such as a fully mobile network, spontaneous network, peer-to-peer network, multi-hop network or just an ad-hoc network[16]. These terms are usually used interchangeably. There are differences that exist between each of these terms, though very subtle. A *fully mobile* network means that all the nodes can move and there are no fixed nodes. A *spontaneous* network is when the network device is part of the network only for the duration of the communication session. A *peer-to-peer* network refers to a non-cellular organization. A *multi-hop* network is when the nodes in the network can forward packets for other nodes. And finally, an *ad-hoc* network indicates the fact that the networks may be temporary and operating without any infrastructure or any central servers administering the network.

2.3.1 MANET Routing Protocols

The routing protocols used in ad-hoc networks must deal with problems like low bandwidth, high power consumption and high error rates[17]. These routing protocols can be broadly classified into *table-driven* and *source-initiated on-demand driven*.

2.3.1.1 Table-driven routing protocols

Table-driven routing protocols maintain the current routing information from each node to every other node in the network using one or more tables. Depending on how the tables are maintained and the changes propagated, the table-driven ad-hoc routing protocols can be classified as follows:

- Destination-Sequenced Distance-Vector Routing (DSDV)[17][18][20]: This algorithm is based on the classical Bellman-Ford routing mechanism.
- Clusterhead Gateway Switch Routing (CGSR)[17][19][20]: This algorithm uses DSDV as the basis, modifying it by using a hierarchical cluster head-to-gateway routing approach to route the traffic from source to destination. The gateway nodes are nodes that are within communication range of two or more cluster heads.
- Wireless Routing Protocol (WRP)[17][19]: This protocol maintains routing information among all nodes in the network. Each node in the network is responsible for maintaining four tables, a distance table, a routing table, a link-cost table and a 'message retransmission list' table.

The *table-driven* protocols have the overhead of high signaling traffic and power consumption, resulting in low bandwidth and efficiency. But these protocols are much faster than *source-initiated on-demand routing* protocols.

2.3.1.2 Source-initiated on-demand routing protocols

The *source-initiated on-demand routing,* as the name indicates, creates routes only when requested by the source node by using the route discovery process. The following protocols use this paradigm:

- Ad-Hoc On-Demand Distance Vector Routing (AODV)[17][20]: This routing protocol builds on the DSDV, but is an improvement because it creates routes on an on-demand basis instead of maintaining a complete list.
- Dynamic Source Routing (DSR)[17][20]: This protocol is based on the concept of source routing. The nodes maintain route caches that contain the source routes that the nodes are aware of, and this cache is updated continually as new routes are learned.
- Temporally-Ordered Routing Algorithm (TORA)[17][20][21]: This protocol is based on the concept of link reversal, is loop-free and very adaptive. This is source-initiated too, and provides multiple routes for any desired source/destination pair.
- Associativity-Based Routing (ABR)[17][20]: In ABR, a route is selected based on the degree of association stability of mobile nodes. Each node maintains an associativity table when a beacon is received from its neighbor. The more the beacons received, the more stable the neighbor is and hence that node moves up in the table. This protocol is also free from loops, deadlocks and packet duplicates.
- Signal Stability Routing (SSR)[17][20][22]: SSR selects routes based on the signal strength between nodes and on a node's location stability, hence resulting in picking nodes with stronger connectivities.

The main drawback of the on-demand routing protocol is the fact that it has to wait for a route to be discovered before transmission.

2.3.2 Problems and Performance Issues in MANET

The mobile wireless networks have dynamic, rapidly changing, random, multihop technologies that are composed of relatively bandwidth-constrained wireless links. The technology is so nascent that there are many problems associated with MANETs. Some of these problems and issues are discussed in this section.

2.3.2.1 Organizational problems in peer-to-peer networks

Due to the mobility of the nodes in a MANET, some organizational problems arise. The need to control the spatial reuse of the channel, in terms of frequency and/or time, is one of the problems that need to be addressed for a mobile ad hoc network [16]. Another issue is the necessity to reduce overheads such as routing and control so that the devices can respond to rapid node movement. One of the solutions to such problems is organizing the nodes into clusters. In this organization, *clusterheads* are elected by assigning weights to each node in a network. A clustering algorithm, similar to the greedy algorithm, is obtained, using which an efficient and deterministic bounds on the distributed clustering problem for peer-to-peer networks can be resolved.

2.3.2.2 Broadcasting problems in peer-to-peer networks

Communications in a MANET usually involve broadcasting at some point of time in a few protocols, and some other protocols completely depend on broadcasting for wireless message transmission (to find routes to a host, to page a host etc). The algorithms for implementing broadcast primitives in peer-to-peer networks and the various system services in networking environments have been widely investigated and numerous algorithms have been proposed [16]. Most of these algorithms use randomized techniques to deliver broadcasted messages to all the nodes; hence these groups of algorithms are called *randomized algorithms*. Another group of algorithms assume that each node knows the topology of the network, and these algorithms are known as *centralized algorithms*.

Though these protocols give efficient broadcasting, they may not be suitable for time-dependent applications that run over the network, or for MANET's wherein the network topology is changing constantly. Research work has been done and protocols have been developed so that they are *deterministic, delivery-guaranteed* and *completely distributed.*

Another problem related to broadcasting that has been studied in detail is known as the *broadcast storm* problem [15]. As mentioned before, devices that are part of a MANET will need to broadcast to find routes, to poll for a host, to advertise, or just to communicate. In order to make sure that all the nodes receive a message, broadcasting messages could be flooded but this will result in one or more of the problems mentioned below:

- Redundancy – Since the radio propagation is omni-directional and many hosts can be transmitting in that physical area, many retransmissions result in redundancy.
- Contention – Since the re-broadcasting hosts are close to each other, serious contention can occur between them.
- Collision - There is a high possibility of collision since many hosts can be broadcasting at the same time.

These problems are collectively known as the *broadcast storm* problems. One of the ways to reduce this problem is to reduce the possibility of redundant rebroadcasts. Another approach is to differentiate the timing of rebroadcasts. Several schemes, such as *probabilistic, counter-based, distance-based, location-based* and *cluster-based* schemes have been developed to help MANET broadcasting.

2.3.2.3 Problems with Jini™ Technology for MANET

Jini is a fairly new distributed programming framework developed by Sun Microsystems [23]. The recurring theme of Jini specifications is ad-hoc networking. In a

nutshell, Jini is a system for discovering and accessing network services. Jini has a lookup service that enables devices and services on the network to be able to find each other and communicate without any prior knowledge about each other. Another theme of Jini is its ability to adapt to changing network conditions. Jini maintains leases for the list of devices and services available, and hence if a service becomes unavailable, the lease will just run out without renewal. When looking up a service, the client downloads a proxy object that allows the client object to communicate with and access the service as though it were a local object. Thus, services can implement their own custom communication protocols while still allowing any client to communicate with them. An example used often is the idea that printers provide their own drivers and the computers can use Jini to get them to use the printer.

Some of the problems faced by the Jini technology could be the reason that it is not as popular as it could have been:

- Jini technology requires a full J2SE virtual machine to be present on the device, or at least a version of J2SE capable of supporting Jini. In addition to this requirement, the wireless devices that run different networking technologies pose a challenge for Jini to be successfully deployed in a MANET [24].
- Jini technology is nascent and seems very promising for the future. Jini was designed for pervasive computing environments, where all devices would be networked and would be talking to each other. But we have very few devices that can run pervasive computing and also run the minimum JAVA environment necessary to support Jini. With the advent of powerful embedded systems developed, this could change in the near future.

- Jini architecture has the advantage that devices do not require server addresses to be known beforehand, but the devices should be able to discover and interact with the lookup service, in addition to their normal functions. This results in complicated applications that need to be developed for the devices [5].
- The Jini architecture still relies on central servers, it would be hard to develop collaborative applications for a MANET.
- Since Jini requires that code be downloaded, there could be a security risk involved when these devices are used in a MANET. This would mean additional software techniques such as encryption would need to be implemented.

2.3.2.4 Performance Issues of MANET routing protocols

In order to assess the performance of any routing protocol [25], we need to have metrics for measurement. For a MANET, a few quantitative and qualitative metrics can be specified. Some of the quantitative metrics are:

- End-to-End data throughput and delay – Measures the effectiveness of the routing policy by measuring the routing performance.
- Route Acquisition Time – The time required to establish route(s) when requested, especially in the case of *on-demand* routing algorithms.
- Percentage Out-of-Order delivery – A measure of packets arriving out of order.
- Efficiency – This is the internal measure of the routing policy's effectiveness. A few of the ratios can be measured to track the efficiency of a policy.

a) $\eta_{data} = \frac{\text{Average number of data bits transmitted}}{\text{Data bits delivered}}$

where;

η_{data} = Bit efficiency of delivering data within the network.

b) $\eta_{control} = \frac{\text{Average number of control bits transmitted}}{\text{Data bits delivered}}$

where;

$\eta_{control}$ = Bit efficiency of the protocol in expending control overhead to delivering data.

c) $\eta_{access} = \frac{\text{Average control and data packets transmitted}}{\text{Data packets delivered}}$

where;

η_{access} = Protocol's channel access efficiency.

Some of the qualitative properties of MANET routing protocols could be:

- Distributed operation – This is the definition of mobile network devices.
- Loop-freedom – Should be free from loops to avoid packets spinning around a network.
- Demand-based operation – The routing algorithm must be able to adapt to the traffic pattern.

- Proactive operation – In case bandwidth and energy resources exist, proactive operation is desirable by the protocol.
- Security – Necessary to have some kind of network-level or link-layer security.
- Sleep period operation – Protocol should account for sleep periods for nodes that are part of a MANET.
- Unidirectional link support – Since bi-directional support is not always available, the protocol must be able to support both.

Some of the parameters that could be varied during performance measurement test for a MANET protocol can be:

- Network size – Number of nodes in a network.
- Network connectivity – The average number of neighbors to a node.
- Topological rate of change – The speed with which a network's topology is changing.
- Link capacity – Effective link speed after accounting for losses.
- Fraction of unidirectional links – How effective is the protocol when unidirectional links are present.
- Traffic patterns – Effectiveness of the protocol during non-uniform or bursty traffic patterns.
- Mobility – The most appropriate model for simulating node mobility in a MANET.

- Fraction and frequency of sleeping nodes – The performance of a protocol in the presence of sleeping and awakening nodes.

A MANET protocol should perform effectively over a wide range of contexts, and the above-mentioned parameters cover a wide range of the wireless traffic that the protocols will be deployed under.

2.4 Developing and Testing Network Protocols

The performance of networking applications depends on a number of factors[27]. Some of them are:

- Physical characteristics of the processor (CPU speed, memory, disk size, etc.)
- Bandwidth of network connections.
- The application's communication efficiency.
- Efficiency of the network protocol that is used for communication.

The network protocol used in an application plays quite an important role in the overall performance of the system. Hence testing of these protocols before deploying them in a system is very important. The performance evaluation of a network protocol involves the analysis of the following criteria:

- Latency – The period of time that a packet or datagram is held by a network protocol or its sub-layer before it is forwarded to other protocols or network components. The latency period should be low.
- CPU utilization – The percentage of CPU time the protocol takes when running a certain application. It denotes the percent of time that a CPU is

busy for a process. Efficient utilization of CPU time means faster communications. Hence the CPU utilization of the stack should be low.

- Throughput – The amount of data that a protocol stack can process per time unit (KB/s or Mbps).

Testing network protocols, such as the M2MP, is not a very straightforward job. The two common ways of testing network protocols[13] are:

- Using the protocols in operational networks, or
- Testing the protocols in simulated environments.

If protocols are used in operational networks, it is hard to debug problems due to the fact that the parameters involved are not controllable. For example, the speed of transmissions, the queue sizes, the delays, bandwidth, etc., are not settable, and hence failures may be hard to trace. For this reason, the second approach to testing network protocols, using simulated environments, is generally preferred.

2.4.1 Dummynet

Dummynet [12][14] is one of the tools that could be used for testing networking protocols, debugging the protocols or conducting performance evaluations by simulating a real network environment. Dummynet manipulates the IP queue inside the kernel to simulate network delays and losses. Dummynet creates a pipe object with the capability for the users to configure the pipe with the desired delay and loss parameters. This pipe gets added to a list of filtering rules that are used by Dummynet to modify the treatment of IP packets in whatever way one desires. The principle of operation of Dummynet is shown in Figure 4.

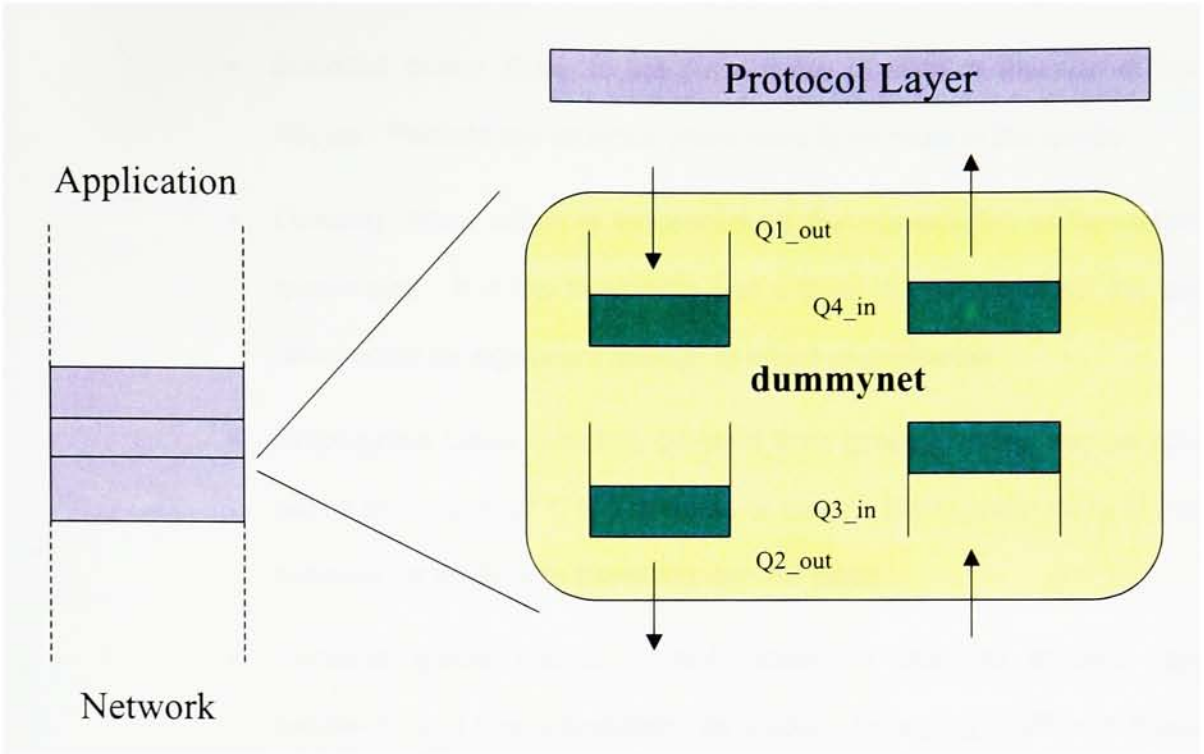

Figure 4: Principle of operation for Dummynet

Dummynet works by intercepting communication between the protocol layer and the underlying one, and simulating the presence of a real network with finite-size queues, bandwidth limitations, communication delays, and possibly lossy links. As shown in Figure 4, a couple of queues are inserted between the protocol layer under observation and the lower layer in each direction. These queues simulate a) routers with bounded queue size and a given queuing policy; b) communication links (*pipes*) with given bandwidth and delay.

Some of the simulations that could be achieved by Dummynet are:

- Random packet losses, which could be used to simulate noisy links in a networked environment.
- Bandwidth limitations, which could be specified in a few different units such as bit/second, Kbit/second and Byte/second.

- Bounded queue sizes, to set the number of slots or the size in bytes or Kbytes. Packets are dropped when there is no room in the queue.
- Queuing delay, which is influenced by the combination of bandwidth and queue size. If a low bandwidth and a large queue are used, the queuing delay could be significant enough to affect performance.
- Propagation Delay, which is different from queuing delay, can be specified with a granularity of 1/100 seconds or lower. Propagation delay is the time required for packets to travel through the pipes.
- Dynamic queue creation, which could be used to enforce separate bandwidth and queue limitations for packets belonging to different flows.
- Loopback interface, which is very helpful for running experiments on a standalone machine because this lets the users run the senders and the receivers on the same machine.

All these features make Dummynet a very ideal tool for testing or debugging network protocols, and for performance evaluations. Dummynet is not suitable for testing an ad-hoc communication protocol since; a) Dummynet controls all traffic as it works by modifying queues at the kernel-level (OS-level). b) Since our ad-hoc applications are JAVA based, and hence OS-neutral, a user-level protocol tester as opposed to the OS-level testing offered by Dummynet would be much more faster to setup and use. c) Dummynet can be viewed as a hardware solution for protocol testing whereas our solution can be viewed as a software solution. d) Dummynet requires too many parameters to be set for fine control, and hence is time consuming to setup and use.

2.4.2 ALPINE

ALPINE (Application Level Protocol Infrastructure for Network Experimentation) [28] is a user-level networking infrastructure that supports a FreeBSD networking stack on top of a Unix operating system. Since the networking stack traditionally resided in the kernel, modifying the network protocol code was error-prone and tedious. Moving the network stack to the user-level is risky because the user has to modify the kernel and/or the application, and it is hard to move code seamlessly between the user-level and kernel stacks. However, the advantages of moving the networking stack to a user-level library rather than developing protocols in the kernels are:

- Debugging is easier – The user-level development will allow for easier source level debugging.
- More stable – If the protocols developed in a user-level environment results in an unstable stack, it affects only the application using it and does not cause a system to crash.
- Faster turnaround – If the protocol is embedded in the kernel, testing every change made to the source code will need a system reboot in order to test the change. Hence this is inconvenient and less efficient.

The advantages of Alpine are that:

- It shortens the revise/test cycle since switching between two Alpine stacks takes very little time, as opposed to switching between kernels, which requires a reboot.
- Once modifications have been developed and tested on Alpine, they can be moved back into the kernel. This is possible because Alpine and the kernel use the exact same source files.

- Alpine works with unmodified application binaries and requires no kernel modifications.
- Alpine improves protocol debugging since the networking stack runs in the user-level and can be examined and changed like any other source code.

Some of the drawbacks of Alpine, and hence the reason we could not use it for our testing purposes are:

- Since Alpine supports a FreeBSD networking stack for the Unix OS only and because we needed something that is architecture and OS-neutral, it would not be suitable for our project.
- To be able to use Alpine (opening a raw socket, capturing packets etc), root privileges are needed, and this wouldn't work very well for this project.
- Alpine does not seem to handle forking processes very well, and hence applications that fork off processes would not be able to use Alpine to test their network protocols.

3 Research Statement

This chapter illustrates the necessity of the throughput-layer to effectively simulate a wireless environment and states the proposed project and research involving throughput control.

3.1 Research Necessity

The M2MP was developed for a proximal wireless environment that is mostly reliable, where all the devices are close together and there is little chance of losing packets. However, the development and testing of this protocol is being done in a reliable wired environment. We need a system that could simulate a wireless environment, so that testing can be performed under controlled conditions. Adding a layer to the M2MP that controls the throughput of the packets passing through it will provide a means for us to control and analyze the system. And by controlling all these applications remotely, we will be able to simulate a wireless environment pretty closely.

3.2 Research Objectives

The objectives of this thesis could be summarized as follows:

- A M2MP throughput-control layer would be designed, developed and integrated into the existing M2MP stack. The throughput-control layer would be responsible for measuring the throughput of packet traffic passing through in both directions and would pass this information to the

RemoteControl application for display on its GUI. The throughput-control layer would also impose a delay on packets passing through the M2MP pipeline according to the rate set by the RemoteControl application. By controlling the send and receive throughput, the user would be able to simulate a wireless environment that the Anhinga devices would be deployed into.

- The throughput-control layer would not alter the current M2MP internals and would be developed as a plugin object that could be built into the M2MP to construct a pipeline. Hence the application could choose the channels to include or omit in the pipeline.
- A RemoteControl application would be developed that would be responsible for displaying the current communication rate achieved (packets/second) as a graph and would provide the GUI for the user to change transmit and receive rates. The RemoteControl application would be multi-threaded and would display a separate window for each host connected to it.
- An application would be developed that would use this new M2MP. This application would exercise the message sending and receiving mechanisms of the M2MP with the throughput-control layer.
- Some of the currently existing network evaluation packages (Example: Dummynet and ALPINE) would be studied and analyzed.

4 Plan Outline

4.1 Architecture Overview

The M2MP architecture supports construction of pipelines by plugging together a series of objects that are necessary for communication. This architecture is shown in Figure 5. This mechanism lets the application choose the filters that will be part of the pipeline. For example, the throughput channel could be chosen to be part of the pipeline or could be omitted. In a similar way, multicast IP channel could be chosen by the application or Bluetooth could be chosen.

The functions of each object in the pipeline can be summarized as follows:

- M2M Protocol – The application layer uses the M2MP layers for communication. The M2MP is made up of different objects as shown in Figure 5. The M2MP breaks down the outgoing messages into packets and sends it down the pipeline. It also receives incoming messages from the pipeline and reassembles them into M2MP messages to be sent to the application layer.

- M2MP Throughput channel – The application decides if the throughput channel is to be added to the pipeline. If so, then the application also has to specify the channel to be used for transmitting and receiving data, such as, multicast channel, unicast channel etc. The throughput channel measures the packet traffic in both directions and sends this information to the

RemoteControl server. This layer delays the packets depending on transmit and receive rates set by the RemoteControl, acting as a bottleneck for communication, hence simulating a wireless environment.

- M2MP Medium Access Layer – Transmits and receives packets over the broadcast medium. This layer can use any of the M2MP's channels, such as multicast, unicast, loopback, to send and receive data.

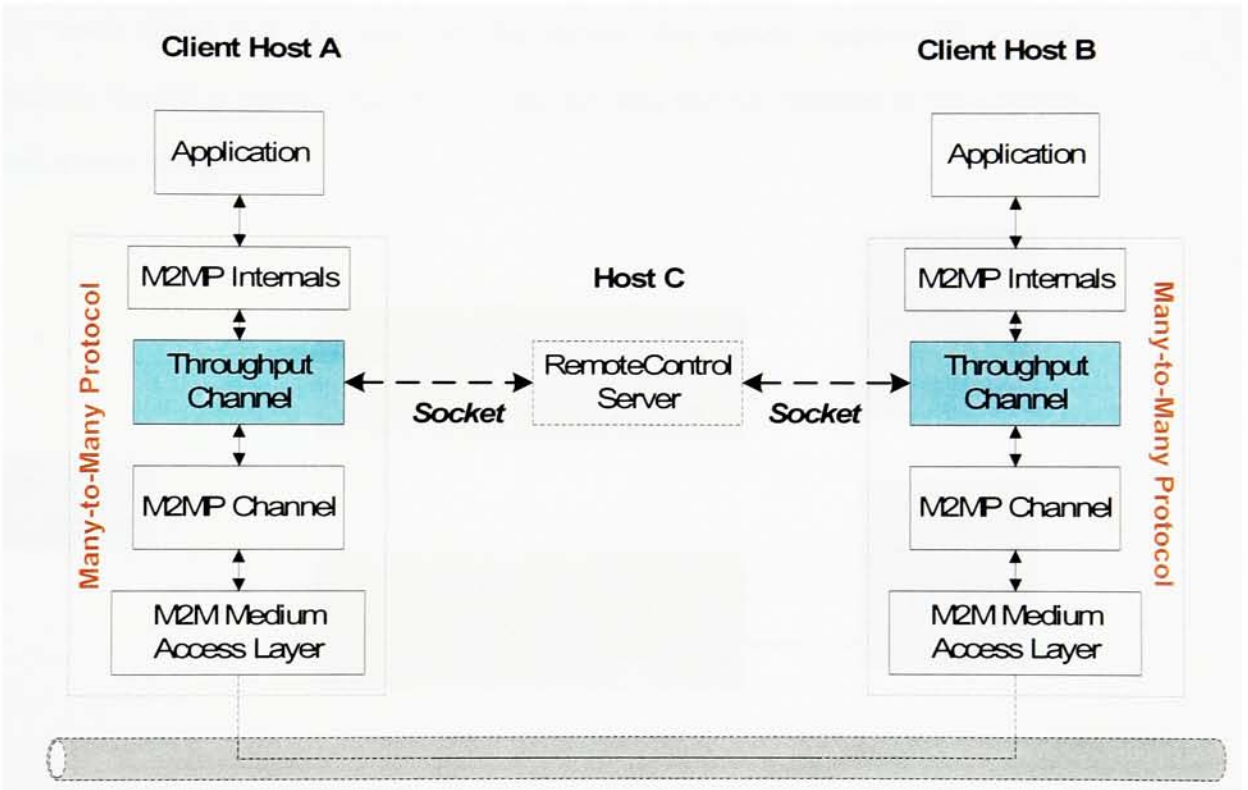

Figure 5: Throughput Architecture Overview

4.2 Remote Control Design

The remote control is a socket server that will be running on a well-known host, listening for clients on a well-known port. If the user decides that the client application should add the throughput channel into its pipeline, then the user has to specify the host name and the port that the RemoteControl is running on at the command line. Using this information, the client application will establish a socket connection with the RemoteControl server.

For each client that connects to the server, the server spawns off a new RemoteService thread to service that client. This process can be depicted in the concept diagram as shown in Figure 6.

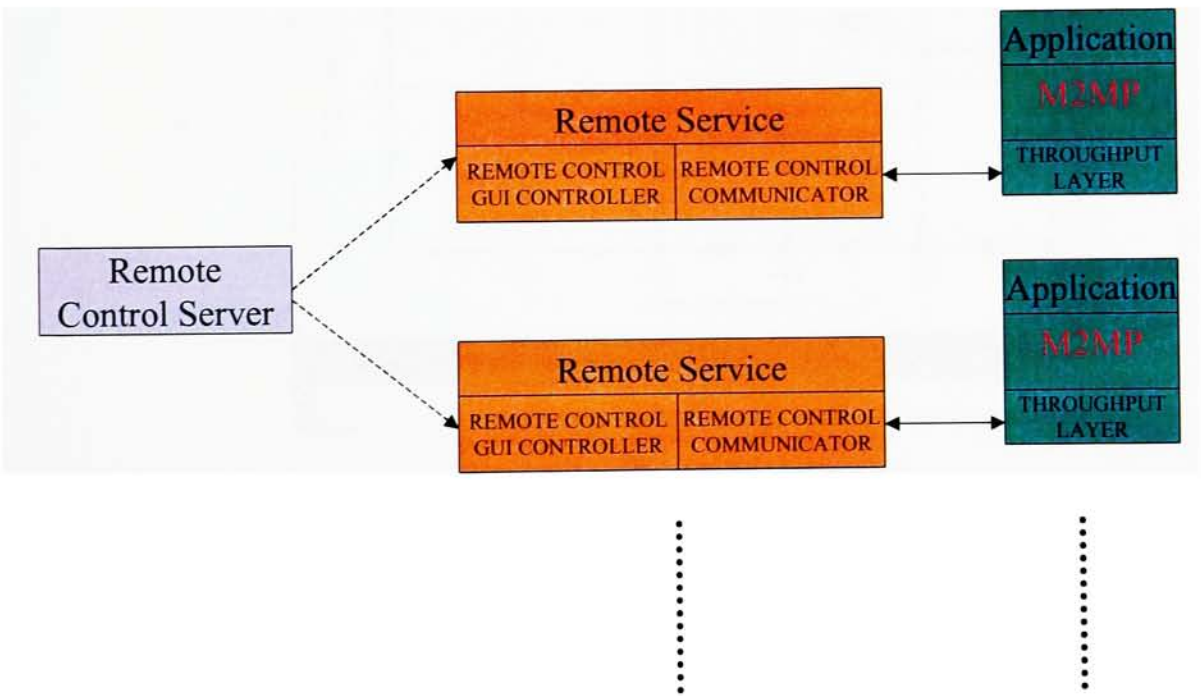

Figure 6: Remote Control concept diagram

The RemoteService thread opens a RemoteControl window at the server. This window will let the user specify the delays for send and receive throughput for that

particular client. In addition to this, the RemoteControl also plots the number of packets sent/received by the clients. The plot is updated every 10 seconds. The data plotted is the number of packets transmitted or received in the last 10-second duration. The plot is wide enough to display last 10 values, i.e., the number of packets sent in the last 100 seconds in 10-second intervals. This GUI for the RemoteControl window is shown in Figure 7.

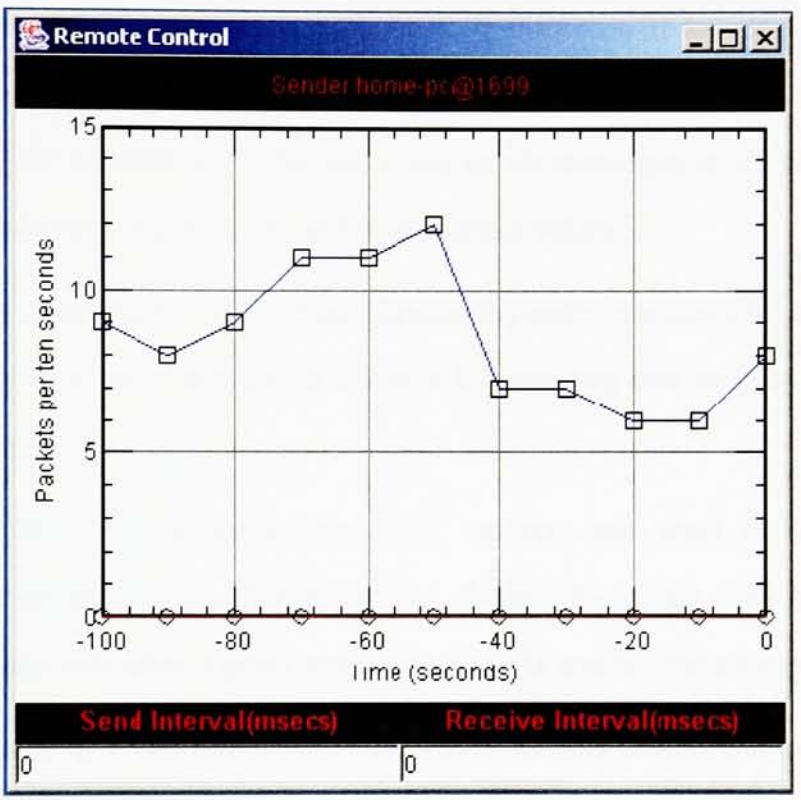

Figure 7: The Remote Control Window

4.2.1 Detailed Design Description

Some of the important classes in the RemoteControl application are summarized below.

- RemoteControl – This is the main class for the application, which continuously listens for clients trying to connect to the server. For each client that requests a connection, the RemoteControl creates a

RemoteService object for servicing that client. The server terminates when the *Enter/Return* key is hit in the terminal window. Before the server shuts down, it terminates connections with all the clients connected to it.

- RemoteService – One RemoteService object is created for each client that requests a connection. The RemoteService is responsible for handling communications to its respective client. It processes all the messages sent by the client, like the client name message, the throughput message sent every 10 seconds etc. This class also sends messages to the client when the user requests a change in the throughput values.

- TCPCommunicator – This class handles the communications for the server, by creating input and output streams for receiving and sending serialized objects.

- Plot MVC – The well-known MVC[1] pattern was used to create the RemoteControl GUI. The panel that displays the graph uses the Plot2D package that takes a given array of data points and returns a panel.

- Name MVC – This the panel on the RemoteControl display that displays the name and port of the client that the RemoteService is connected to. Every message that is received by the RemoteService has the client name and port information in it, so that information can be parsed and displayed on this panel.

- ClientFrame – The RemoteControl display window, its layout and frames are specified in this class. This class also processes the window events when the user specifies new throughput values or if the user closes the window.

[1] The Model-View-Controller design pattern used to create GUI's. [30], [31], [32]

The class diagram for the RemoteControl application is shown in Figure 8.

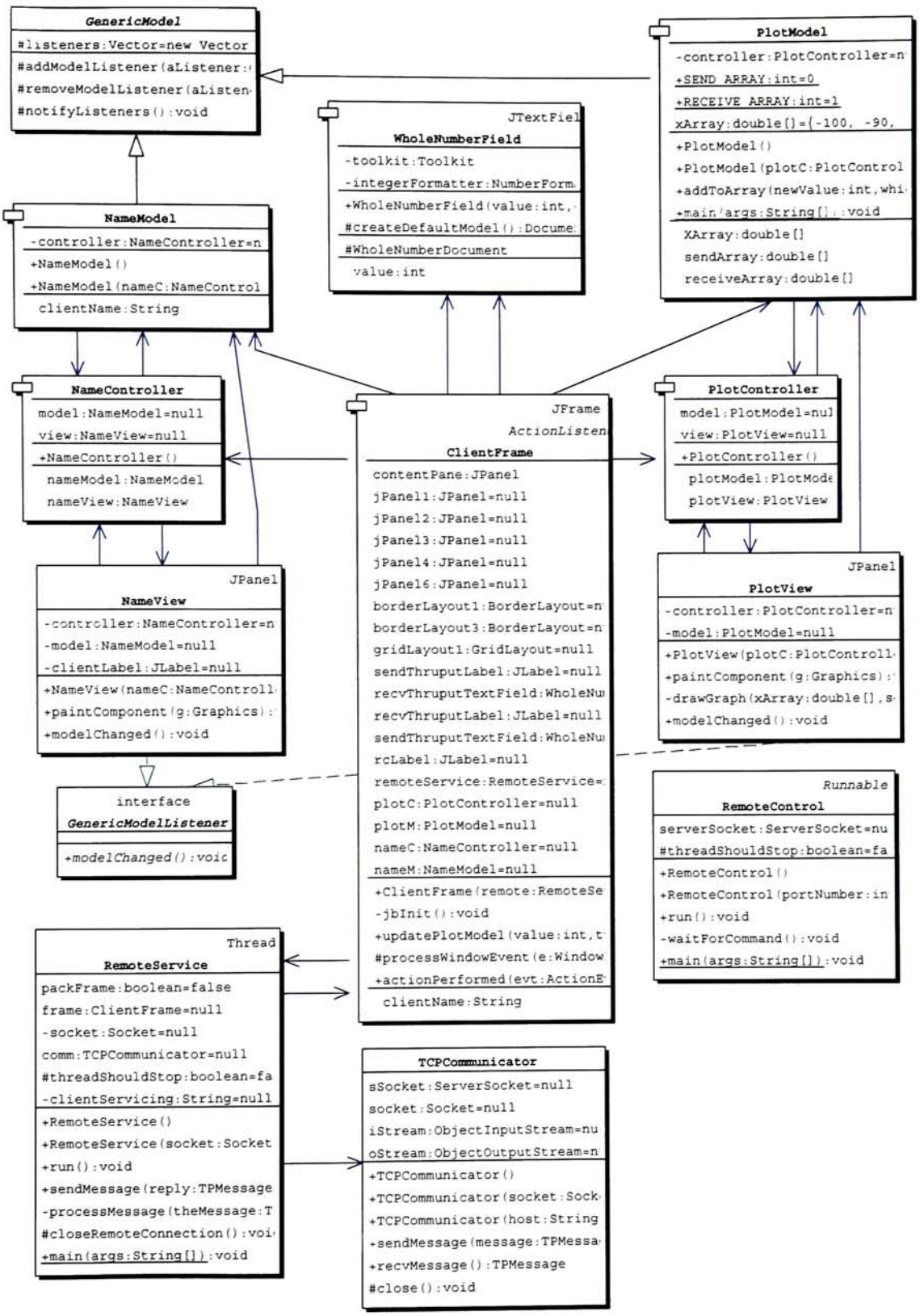

Figure 8: RemoteControl Class Diagram

4.2.2 Communication Description

The communication highlights of the RemoteControl server are as follows:

- The RemoteControl Server listens for clients trying to connect and for every client that connects, the RemoteControl server will spawn off a new thread to service that client. The new thread created for each client is called the RemoteService.
- The RemoteControl establishes a socket connection with each client, and during the handshake, receives the client host name and port number for display at the server.
- The RemoteControl receives serialized messages from the clients every ten seconds. This message contains the number of packets that the client sent/received in the past ten seconds. The RemoteControl plots this number on a graph displayed at the server as shown in Figure 7.
- If the user at the server sets new values for the send and receive throughputs, the RemoteControl will communicate these values to the respective clients.

All these communication sequences are shown in Figure 9.

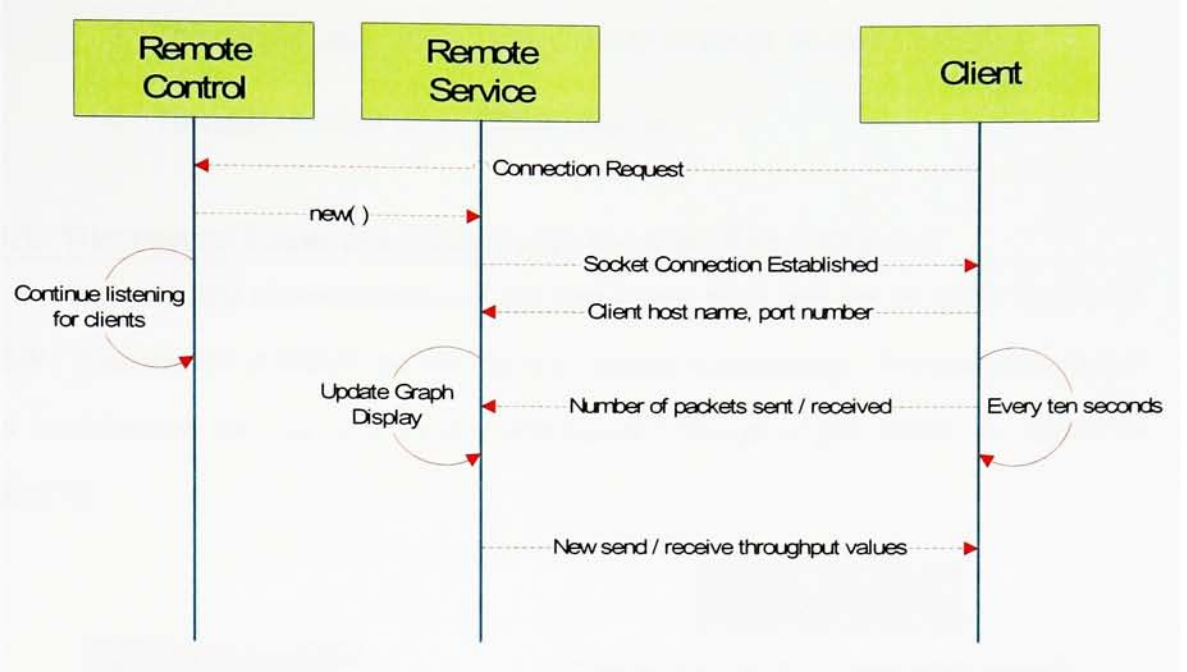

Figure 9: Communication sequence for RemoteControl

4.3 Throughput Layer Design

The throughput layer is a part of the M2MP that controls the rate at which the packets are sent / received from the application layer. The RemoteControl server can control the delay values used by the throughput layer. This throughput layer is added to the M2MP only for the purposes of testing the M2MP under controlled conditions. Wireless network conditions can be simulated using the throughput layer.

This section discusses the various implementations of the throughput layer code and their pros and cons, if any. The following four variations of implementations were tested:

1. Throughput Layer associated with the IPMulticast Channel.
2. Throughput Layer as part of the M2MP Internals (Blocking Receive)

3. Throughput Layer as part of the M2MP Internals (Packet Controlled)

4. Throughput Layer as an M2MP Channel.

4.3.1 Throughput Layer associated with the IPMulticast Channel

The very first implementation of the throughput layer was on an older version of M2MP. This version of M2MP did not use the concept of pipelining. The throughput layer was implemented for use only by the IPMulticast Channel of the M2MP as shown in Figure 10.

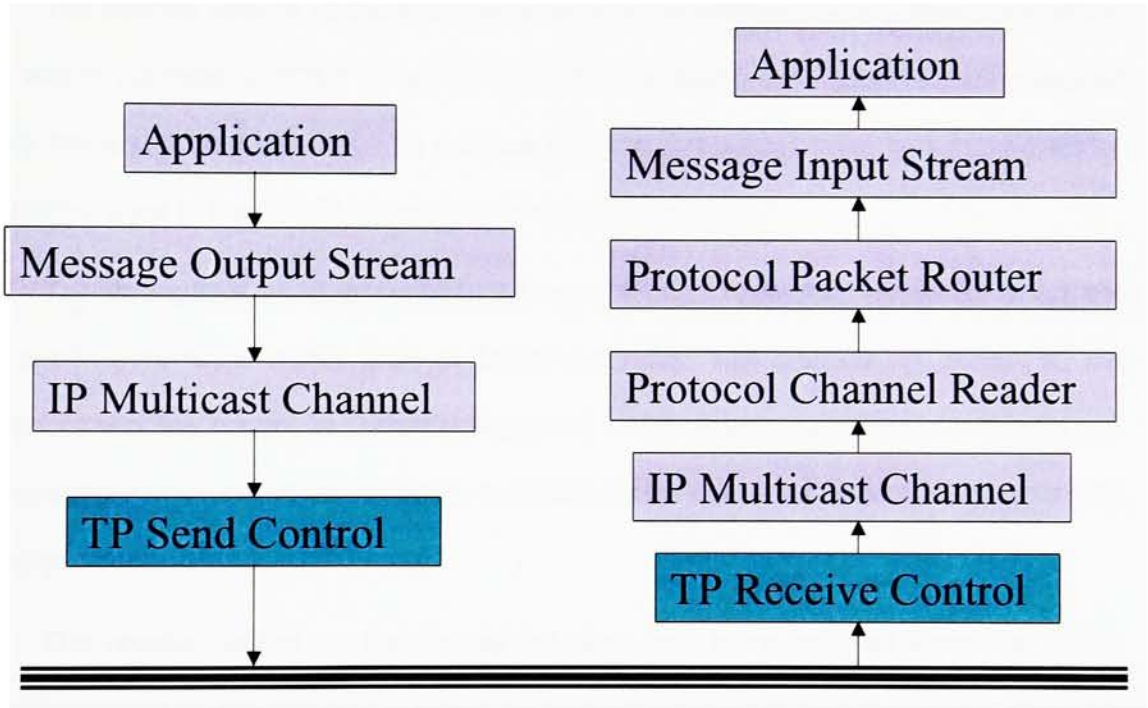

Figure 10: Throughput layer as part of the IP Multicast Channel

This implementation modified the IP Multicast Channel class to communicate with the throughput layer's send and receive methods. The throughput layer's logic implemented the actual send and receive methods. This implementation worked well with the older M2MP code, and later was ported to work with the new M2MP. The main

disadvantage of this implementation was that only the IP Multicast channel could use the throughput layer, the other channels could not. Hence it was decided to move the throughput layer higher up so that any channel could be included in the communication. Another disadvantage in this implementation was that a queue was used for send throughput control, which is not an efficient solution, especially when the size of the queue was large.

4.3.2 Throughput layer as part of the M2MP Internals (Blocking Receive)

The second version of the throughput layer implementation was a little more robust since any of the existing M2MP channels could be specified for the communication instead of only the IPMulticast channel. To achieve this, the throughput layer was moved above the channel layer in the M2MP internals as shown in Figure 11.

The throughput sending control was more straightforward to implement since the MessageOutputStream object (part of M2MP internals) was hooked up directly to the throughput sending control as shown in Figure 11. Whenever a packet was to be sent out, the throughput layer would decide when it would be sent out based on the delay setting for throughput send control.

The receive control for the throughput layer was more involved since the M2MP internals needed to be modified. The throughput receive control was hooked up to the MessageInputStream object (part of M2MP internals). Any packets received would be added to the link list available in the M2MP internals until the throughput receive control decided it was time to send the packet to the layer above it. This was implemented using blocking calls in the throughput control.

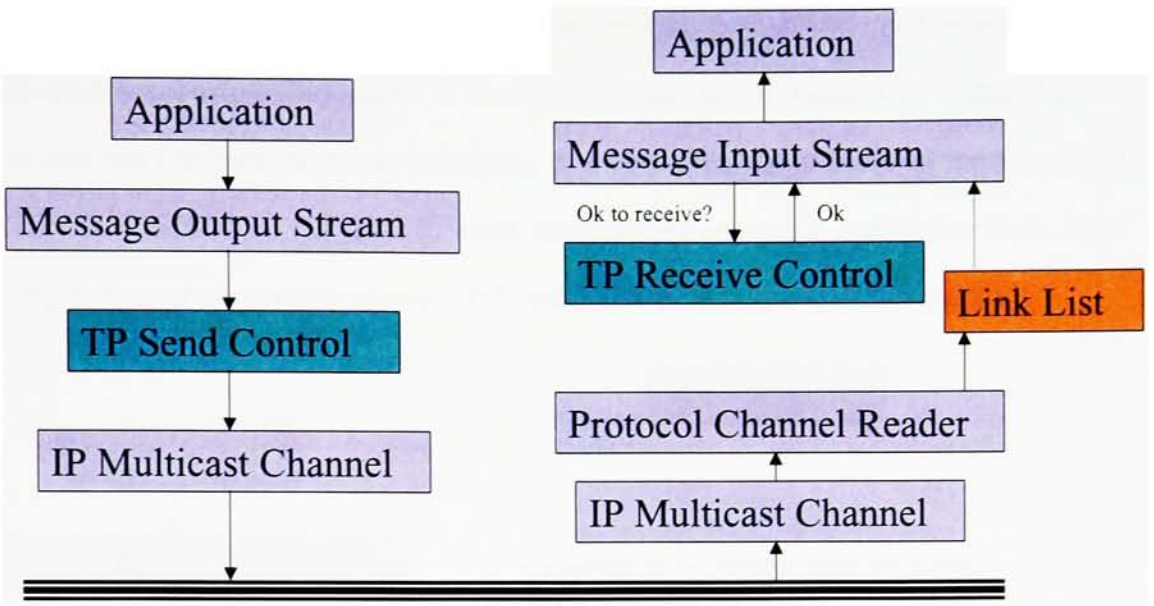

Figure 11: Throughput layer as part of the M2MP Internals (Blocking Receive)

The disadvantage of this implementation was that the M2MP internals was changed and this made the code less modular. The blocking call for receive was not efficient since the packets created within the M2MP could run out pretty fast, and if the packets in the link list were not released, the M2MP could fail. Hence another variation was tried without a blocking call.

4.3.3 Throughput layer as part of the M2MP Internals (Packet Controlled)

This version of the throughput layer implementation for send control was the same as the previous version. The receive control, however, was changed to be a non-blocking call. The M2MP internals were modified for this implementation. The number of packets to be used by the M2MP buffer is specified by the application up front. The throughput layer did not use this buffer but created its own queue to hold the incoming messages. The throughput layer also had to create its own instances of *IncomingPackets*. The throughput layer copied each packet received from the *ProtocolChannelReader* object, stored it in its

own queue as objects of *IncomingPackets*, and notified the M2MP mechanism that the packet was successfully processed. Thus, the M2MP would release its buffer for this packet and use it to receive more messages. The throughput layer waits for the number of milliseconds specified by the user to send each packet up to the application layer, thus resulting in throughput receive control. This mechanism can be shown in Figure 12.

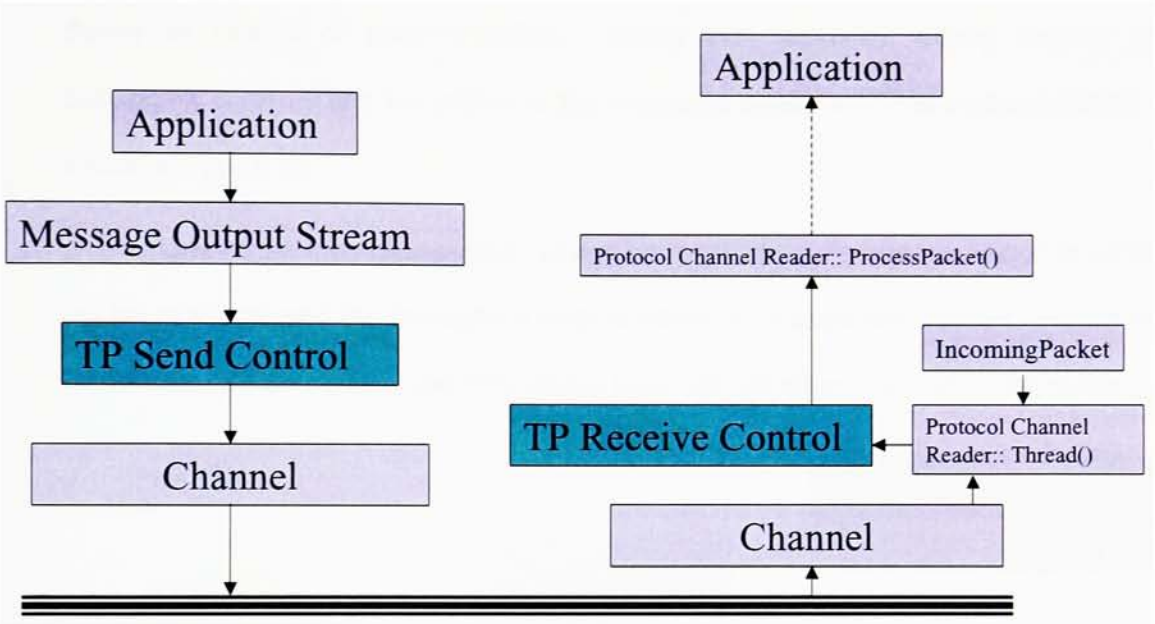

Figure 12: Throughput layer as part of the M2MP Internals (Packet Controlled)

Again, one of the main disadvantages of this implementation was that the M2MP internals were altered. The mechanism used to test a protocol should, as far as possible, not interfere with the protocol itself but should reside above (closer to the application) or below (closer to the MAC layer). Keeping this in mind, the final version of the throughput layer was implemented as one of the channels to the M2MP.

4.3.4 Throughput Layer as an M2MP channel

This final implementation of the throughput layer utilizes the pipelining architecture provided by the M2MP. The throughput layer is written as a channel to the M2MP. This

channel can be used by an M2MP protocol instance to perform the communication. A channel pipeline can be created by hooking up multiple channels and the final channel can hook up to the communication medium. Hence the throughput channel can use any of the other M2MP channels (IP Unicast channel, IP Multicast channel or Loopback channel) in its pipeline. The outgoing and incoming M2MP messages will traverse this channel pipeline during its course of communication. Using this capability, debug logging or other debugging controls can be added to the message streams. This implementation can be shown in Figure 13.

One of the main advantages of this implementation is that the M2MP internals need not be changed, and the throughput layer is added as a separate channel. Hence the code is modular and changes to the throughput layer will not affect the rest of the protocol.

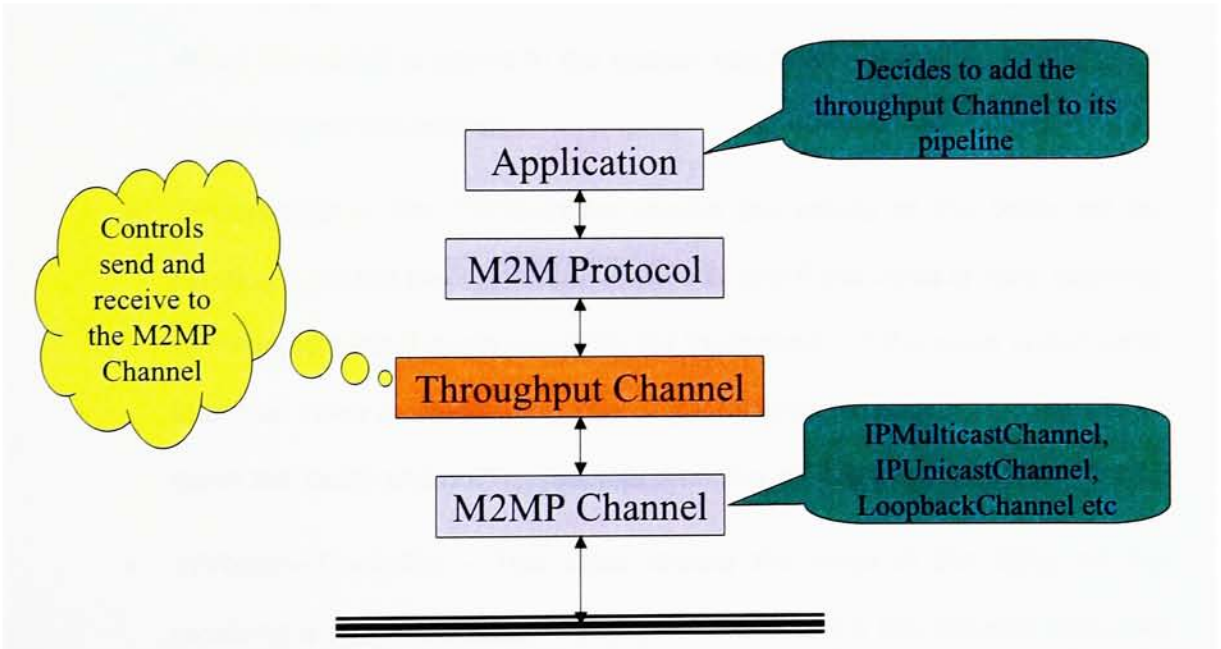

Figure 13: Throughput Layer as an M2MP Channel

4.3.4.1 Detailed Design Description

This design was the final implementation for this research and hence will be discussed at length. Some of the important classes in the throughput layer are:

- TPChannel – The TPChannel class encapsulates a throughput channel that controls the speed of sending or receiving packets. This class implements the *Channel interface* thus providing the transmitPacket and receivePacket methods, which could be used to transmit or receive packets with throughput controls.

- TPManager – This class has a separate thread that is responsible for receiving new throughput values from the RemoteControl server. This class also maintains a timer that ticks every 10 seconds, so that the client can send information to the server.

- TPReceiveManager – Receives an IncomingPacket object and creates a TPIncomingPacket object out of it and adds that to the TPDatagramQueue. When the object is added to the queue, any listeners waiting on the queue data structure are notified.

- TPController – The TPController checks the values of the delay set for sending a packet (send throughput control) and if this value is zero, switches to immediate send mode (no delay for the thread). If the value is non-zero, then that value is used as the time to pause between packets. A timer ticks down this delay and notifies the object waiting on it to send a packet.

- TPReceiveController – This class checks the value of the delay set for receiving a packet (receive throughput control) and if this value is zero, and if a packet is available in the queue, then that packet is de-queued for processing. If the value is a non-zero value, a timer is started for de-queuing the next packet from the queue.

- TPDatagramQueue – This is the data structure used to hold incoming packets temporarily. All the packets arriving will be inserted into this queue by the TPReceiveManager object and the TPReceiveController object will remove the packets.
- TPCommunicator – Takes care of the socket communication to the RemoteControl server. Implements the send and receive message objects that will transmit and receive serialized objects.
- TPMessage – This is the serialized object that is transmitted between the RemoteControl server and the clients. This class contains values that can be set by the client before serializing and transmitting over the socket connection.

For a better understanding of the communication sequence between these objects, please refer to section 4.3.5 later on in this thesis. Figure 17 and Figure 18 depicts the communication sequences for send and receive throughputs respectively.

The class diagram for the throughput channel will be discussed in this section. Figure 14 shows the class diagram for the throughput layer.

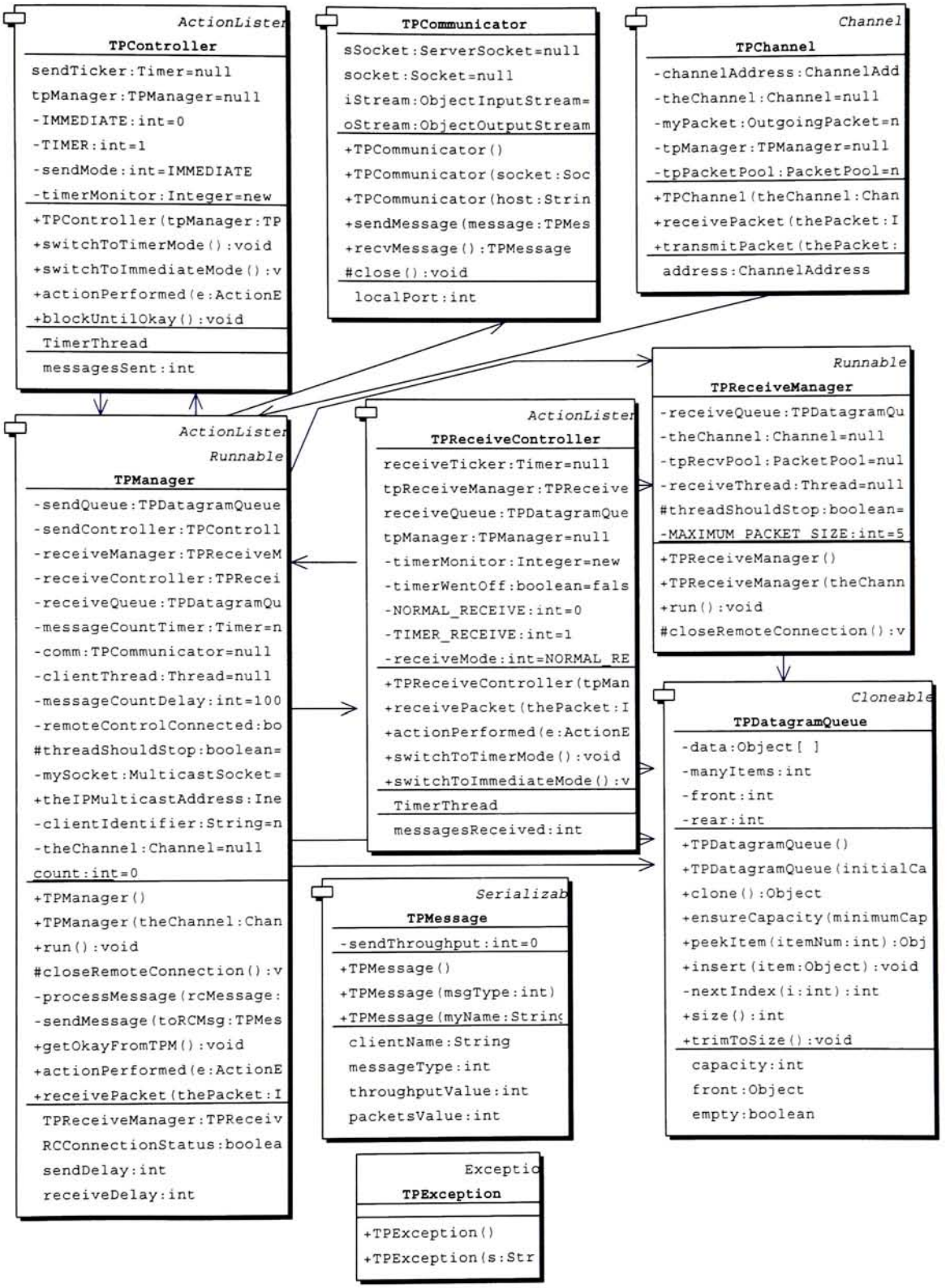

Figure 14: Throughput Layer Class Diagram

4.3.4.2 Packet Send Control

The send throughput control's design is shown in Figure 15 below.

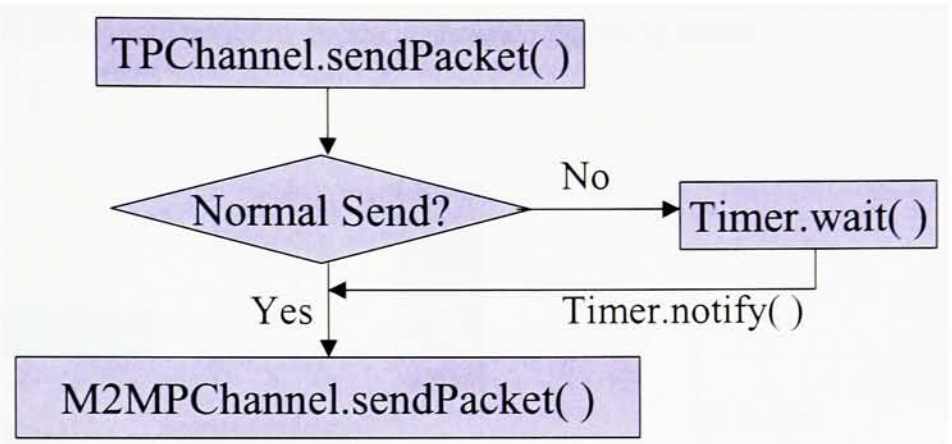

Figure 15: Throughput send-control design

The design for the send throughput control is simpler than that of the receive control; the reason being that this design does not need to use any data structures like queues. When the throughput layer is built into the pipeline by an application, an additional check is done before the MAC layer sends out any packet. One of two things can occur at this point:

If the throughput layer is set for normal send *(i.e., no delay is specified by user for the send part)*, then the packet is sent down to the next layer without any further processing.

If the throughput layer is set for delay send *(i.e., any value for delay other than zero)*, then a timer is set for that delay and the packet is held. When the timer goes off, the packet is sent down for transmission and the M2MP is notified that the packet was transmitted. While the packet is being held, the application is not able to transmit the next

packet and would be in a "waiting" state. When the application is notified about the transmission, it can send down the next packet.

4.3.4.3 Packet Receive Control

The receive throughput control's design is shown in Figure 16 below.

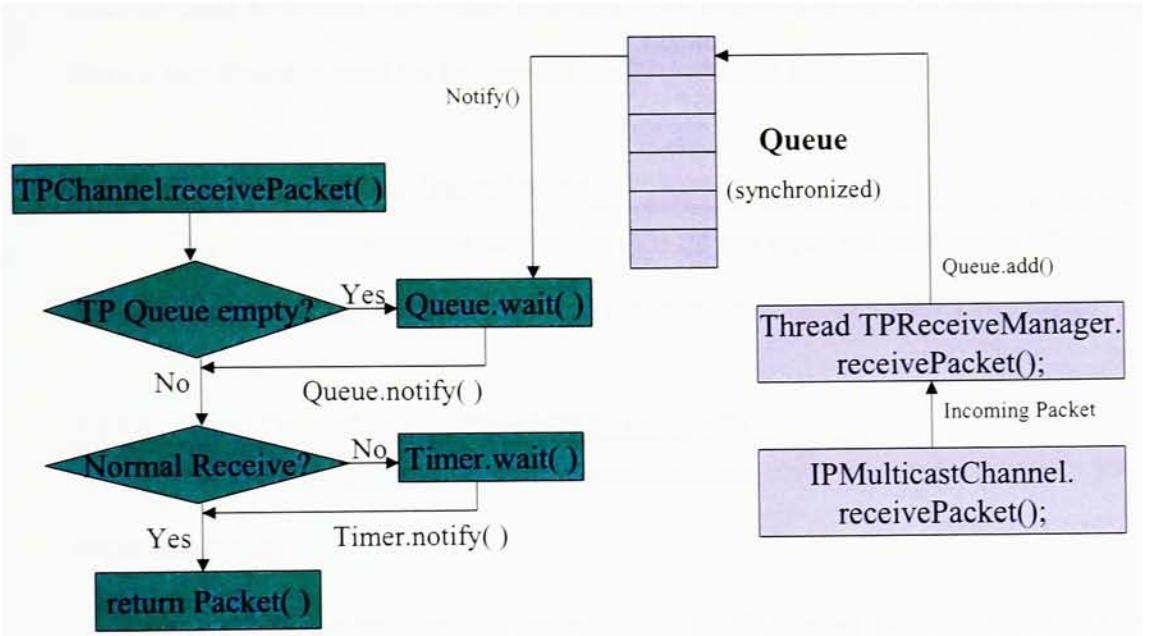

Figure 16: Throughput receive-control design

The receive throughput control is a little more complex, and is a multi-step process. All incoming packets always go into a queue. A thread is running at the throughput layer that is always listening to incoming messages. It receives the packets and stores them away in a queue. Whenever a message is added to the queue, any objects that are synchronized on the queue object will be notified.

The throughput channel waiting for the notification would grab the packet. There has to be one more check before the packet can be sent up to the application layer. If the receive control has been set for no delay *(i.e., zero millisecond delay)*, then the packet is

sent up to the application layer immediately. But if the RemoteControl has specified a delay, then the thread would wait for a timer event to notify the thread to continue.

The queue data structure was adopted from the book "Data structures and other objects using JAVA" by Michael Main [7]. The Queue data structure is not very efficient due to the copying of arrays when the queue grows in size, and could be substituted to another data structure. In faster systems, the use of this data structure did not seem to have a significant impact on the overall performance of the system.

4.3.5 Communication Description

The communication details for the throughput layer will be divided into two sections, the send communications sequence and the receive communications sequence.

4.3.5.1 Send Throughput Communication Sequence

The send throughput control communications can be broken down into the following important steps:

- The sendPacket() method in the TPChannel class is called by the M2MP layer.
- The TPChannel class makes a blocking call to TPManager, so that the thread waits until notified by the TPManager. This logic is shown in Figure 15.
- After the TPManager returns, it is okay to send the packet, so the TPChannel calls the transmitPacket() method provided by the M2MP channel selected by the application.

This sequence is shown in Figure 17 below.

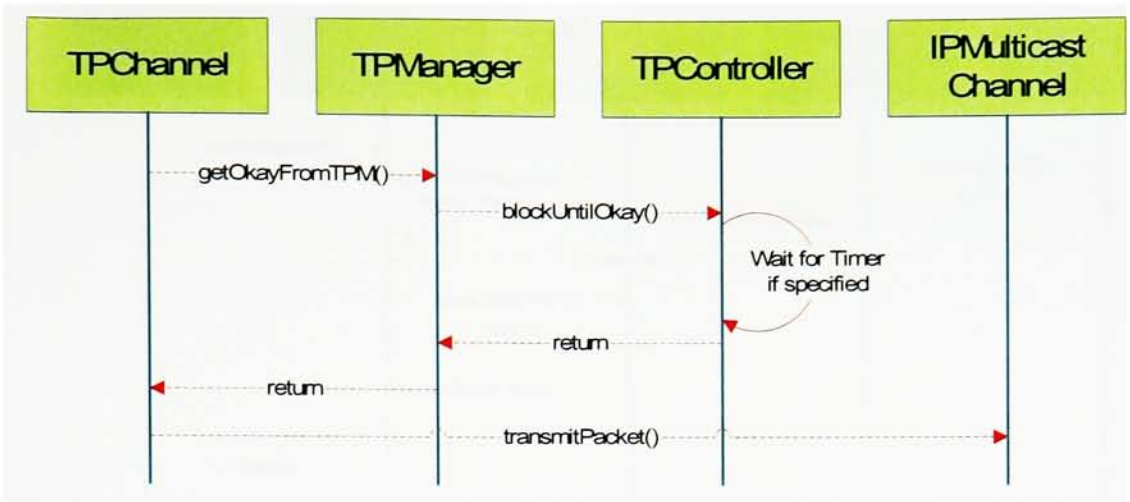

Figure 17: Throughput Send Communication Sequence

4.3.5.2 Receive Throughput Communication Sequence

The receive throughput control communications can be broken down into the following important steps:

- These following two steps continue independent of the rest of the system.
  - The M2MP channel that is selected by the application always continues to receive packets and sends them to the TPReceiveManager class.
  - The TPReceiveManager copies each packet into its own buffer, adds it to TPDatagramQueue and notifies any threads waiting on the queue object.
- The TPReceiveController waits on the TPDatagramQueue object until it is notified that a packet is available.
- The packet is then retrieved from the queue and copied into the buffer provided by the M2MP layer for processing.

The sequence of events is shown in Figure 18.

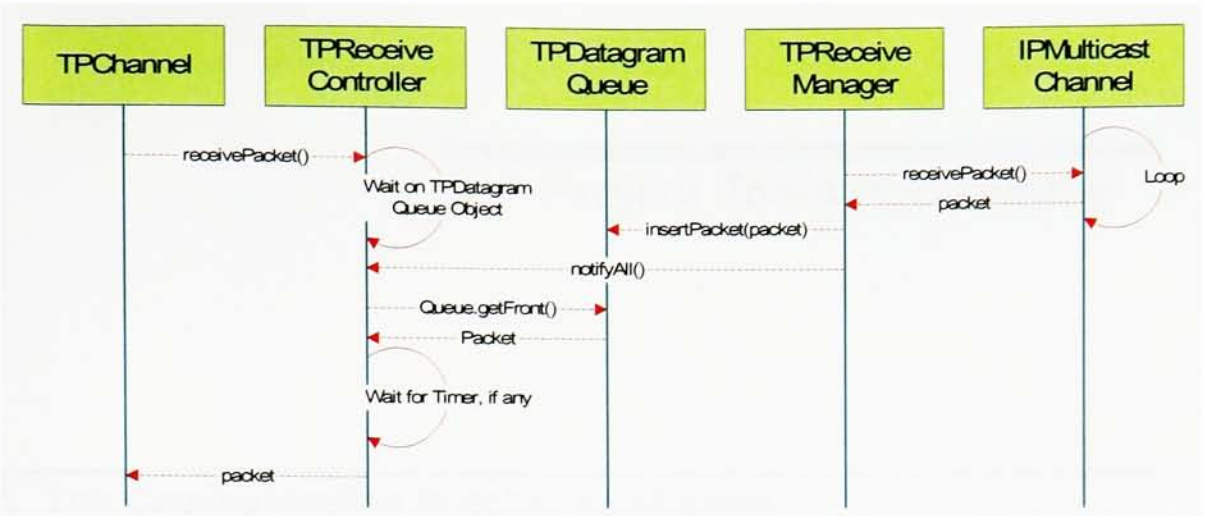

Figure 18: Throughput Receive Communication Sequence

Though this project is not supposed to test packet losses, we could use the mechanism to simulate a packet loss. If the receive timer is specified to be very long and the application specifies a small timeout, then the *Protocol* instance could timeout and declare a timeout.

5 Project Tools and Testing

5.1 The Communication Watcher Application

A simple application was needed to test the M2MP with the throughput layer modifications. The requirements for this application were:

1. The application should utilize the M2MP to send and receive data.
2. The application should provide feedback to the user whenever a packet is sent or received.
3. The application should have minimal overhead so that it does not utilize a lot of system resources.
4. The application should provide controls so that the user can control the rate at which the application can transmit data.

To satisfy these requirements, a very simple communication watcher GUI application was written. The GUI is shown in Figure 19.

Figure 19: Communication Watcher GUI Tool

This application depends solely on the M2MP to transmit and receive data. The application draws a square on its window for every packet successfully sent or received. Double buffering was used for drawing the squares on the window so that redraw would be more efficient and fast. If the application is run as a sender, then it provides a text field for the user to enter the delay value between packets during transmission. This simple application satisfied the requirements needed for exercising our new M2MP, and hence was used extensively during this research.

5.1.1 Communication Description

The communication watcher application can opt in or opt out of adding the throughput layer to its pipeline. In addition to the throughput layer, the application must

choose one of the other M2MP channels for communication. If the application decides to build the pipeline with the throughput layer, then the communication will consist of the following steps:

- The application chooses the throughput channel and one of the other channels (IP Multicast, IP Unicast, Loopback etc) for communication.
- The throughput layer sets up separate threads for sending and receiving messages, with the default values being a zero millisecond delay.
- If the RemoteControl sends new values for send of receive throughput delay, then these values will be used in the threads.

The connections between the hosts and RemoteControl would be established as follows:

Each host connects to the RemoteControl application using server sockets. This is shown in Figure 20.

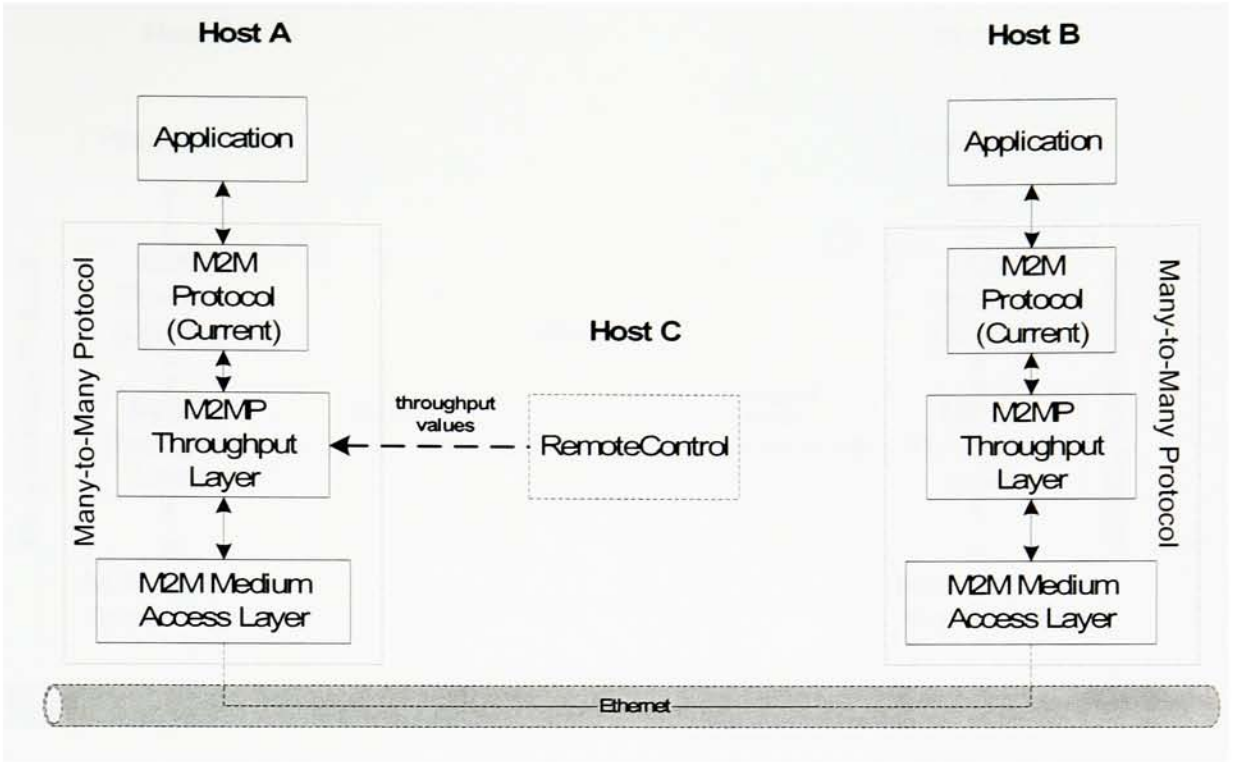

Figure 20: Host A connecting to the RemoteControl

As a default, the M2MP throughput layer uses no delay for send or receive throughput. During the communication, the RemoteControl could set the throughput values that the M2MP Throughput layer should use for its communications. Using these values, the M2MP Throughput layer controls the rate at which the datagram packets are sent / received, hence simulating a real wireless environment that the Anhinga project devices would be deployed into.

When any other hosts establish a connection with the RemoteControl, the new host would receive the throughput values that it should use. Socket connections are established in this way with any number of hosts running the applications. The connection with the second host is shown in Figure 21.

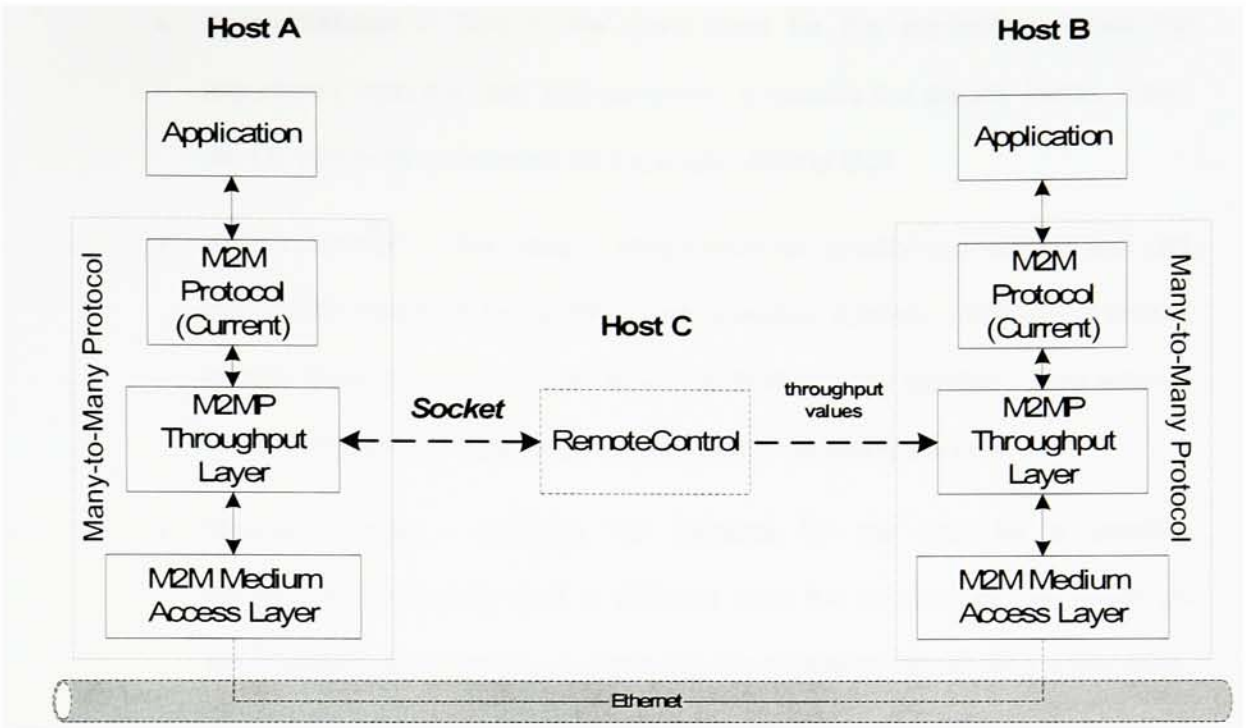

Figure 21: Host B establishing connections with the RemoteControl

For each connection established, the RemoteControl will offer a GUI for the user to control the throughput values in that host's M2MP Throughput layer. The GUI also displays the transmission and receive rate (packets per second) in the form of a graph. This GUI would look similar to Figure 7. The text fields on the RemoteControl GUI could be used to control the send and receive throughput values for both the hosts.

The communication watcher package application is a very simple package used to send and receive packets using the M2MP and to display this information to the user visually. The important classes in the communication watcher application and their high-level description follow:

- CommWatcher – This is the main class for the application, takes the arguments from the user and parses it. It creates the display frame, builds the UI, and sets up listeners to the events for the GUI.
- CommHandler – This class is responsible for specifying methods that use the M2MP channels for sending and receiving packets. This class should specify if the throughput layer would be built into the pipeline. Also sets up the CommMessageReader internal class for receiving packets.
- MessageSender – Handles the repaints for the GUI for a sending application. A sender GUI is different from the receiver in that it has an extra textbox in which the user can specify delays at the sending application, if needed. The default for this is 1000 milliseconds. Double buffering is used for efficiency. A checkbox is provided if the user prefers not to watch the squares being redrawn.
- MessageReceiver – Sets up the repaints for the receiver GUI application. This class uses double buffering too, for efficient redraws. A checkbox is provided in this GUI too so that users can disable the repaints in the window.

The class diagram for this application is shown in Figure 22.

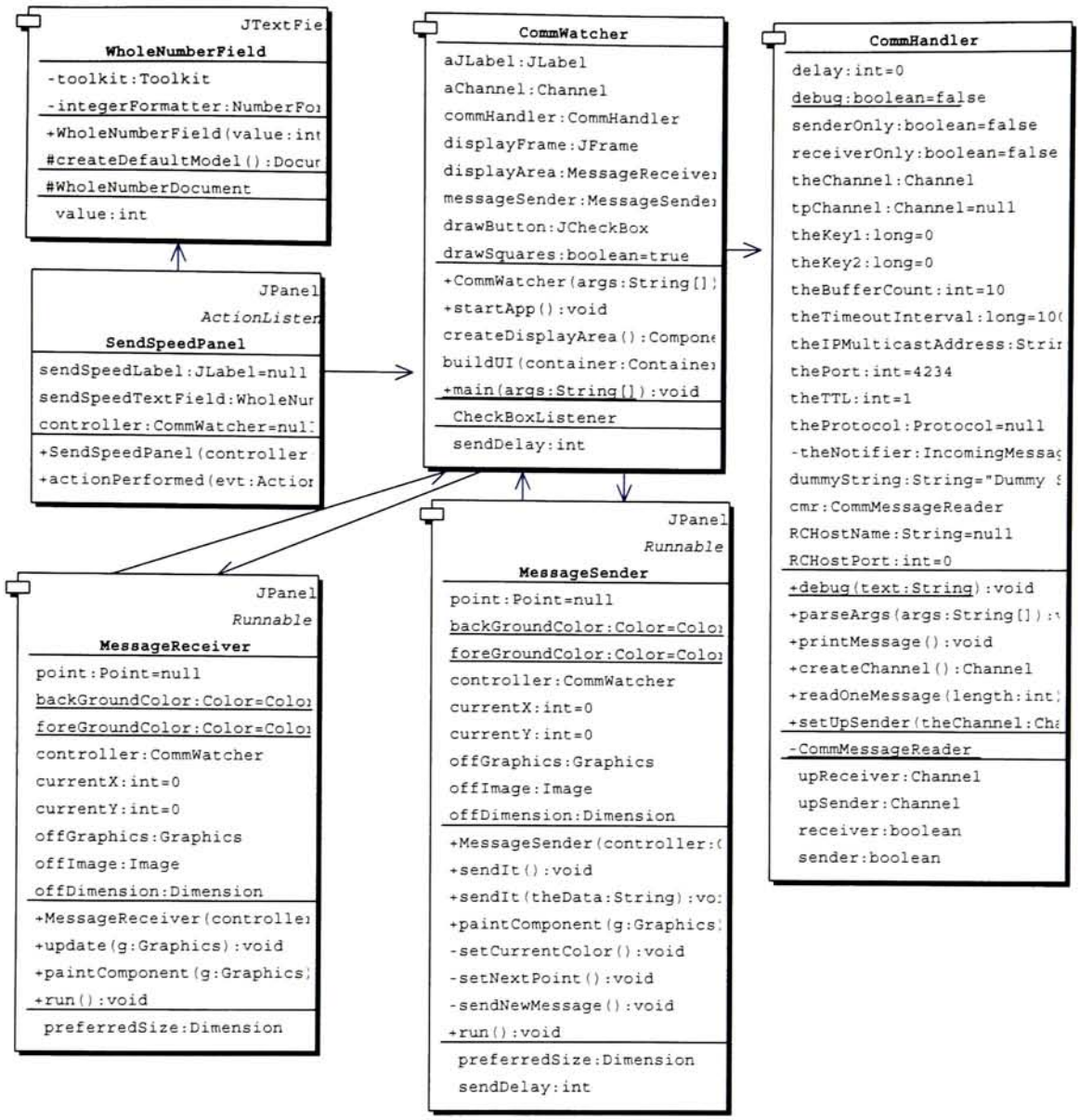

Figure 22: The communication watcher package API

5.2 The Plotting Package

A number of plotting packages were available on the Internet for use in our project. The requirements for the plotting package were:

1. The plotting package should be easy to use and understand.

2. This package must be written in JAVA or some compatible language.
3. The source code needs to be available for customization.
4. The software package must be available free of cost, and must be usable without any restrictions.
5. The plotting package should display in color.
6. The plotting package should be easily added to our RemoteControl application.

All these requirements were satisfied by a plotting package developed by Mr. Joseph A. Huwaldt, version October 14, 2000. The package, available as a downloadable jar file [8], provides a lot more functionality than that required by our project, the interfaces are simple and the code easy to understand. Some of the code was modified in this package to be able to display different colors, and also to be able to add the plotting panel to the RemoteControl application.

6 Results, Conclusions and Recommendations

The main aim of this research was to develop a way of controlling the throughput of an application using M2MP, to monitor this throughput rate, and to implement a way to get real-time feedback using plots.

6.1 Results Summary

The throughput layer was successfully implemented in the M2MP for both sending and receiving packets.

The communication watcher application (Figure 19) sends the number of packets sent and received by it during the last ten seconds to the RemoteControl. The RemoteControl updates the plot every ten seconds using the new data.

The communication watcher application has an option for the user to turn off the drawing of the squares. By using this checkbox option, the user can remove the graphic intensive redraw operations, so that the application can send and receive at maximum speed.

If the RemoteControl (Figure 29) alters the values of send and/or receive throughput, then the new values are communicated to the respective clients. Subsequently, the clients use the new delay value between sending/receiving packets.

The RemoteControl has a text field in which it displays the client hostname and the port on which the client is running so that the operator at the RemoteControl does not get confused between the client windows.

The new layer (throughput layer) added to the M2MP was designed to be a separate channel that could be added to the pipeline constructed by the protocol. The throughput layer by itself does not perform the communication, but it uses one of the other specified channels for sending and receiving data.

Some of the behaviors observed during testing are explained in the following sections.

6.1.1 Normal send and receive throughput case

This section represents a typical wireless communication, with normal throughput between peers, and no delays in the communication. Most of the communications would fall under this category. The sender was setup to send a packet every 800 milliseconds so that the number of packets sent and received could be accounted for. The sender and the receiver were running on the same machine, and hence in the same process. The behavior is shown in Figure 23 and Figure 24.

It can be seen that the receiver has no problem keeping up with the sender. This case proves that the protocol is not dropping packets and works in an acceptable manner. Some of the special cases are shown in the next few sections.

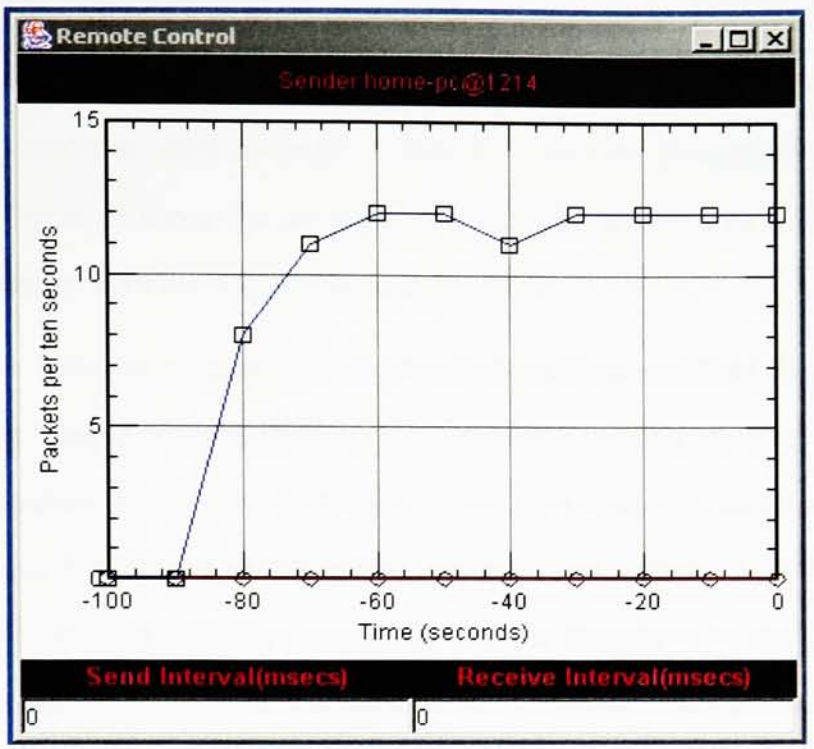

Figure 23: Sender with zero msec throughput-control delay, 800 msec delays between packets

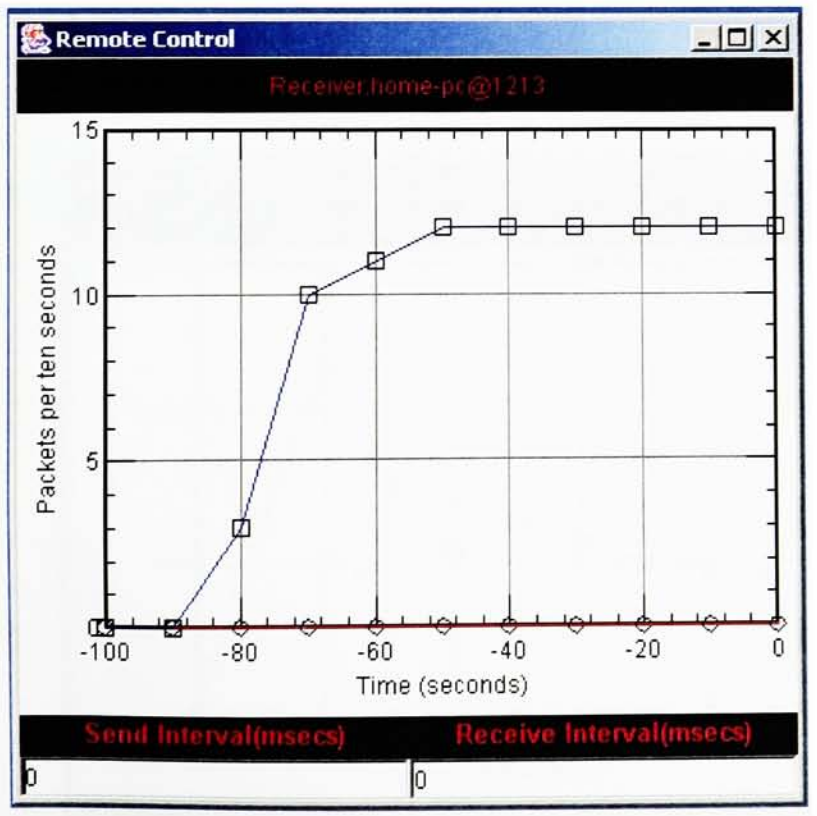

Figure 24: Receiver with zero millisecond throughput-control delay

6.1.2 Maximum send and receive throughput case

The behavior seen was that the throughput peaks to a maximum and slowly reduces to reach an optimum level. Figure 25 shows the throughput plot for a sending client and Figure 26 shows the plot for a receiving client at the same time. Both the sender and the receiver were running on the same machine.

This behavior of attaining a maximum throughput and then dropping down slowly was at first thought to be caused by the application drawing squares, which performs a redraw operation during send or receive. The application was modified so that the user could disable the drawing of the squares. But it was found out that the redraw was not causing this behavior. Another possible culprit was thought to be the queue used by the throughput layer that performs copy operations as the queue size grows. This seemed like a logical explanation to the behavior, but when the throughput layer was removed from the pipeline and the loopback channel was used, the behavior persisted.

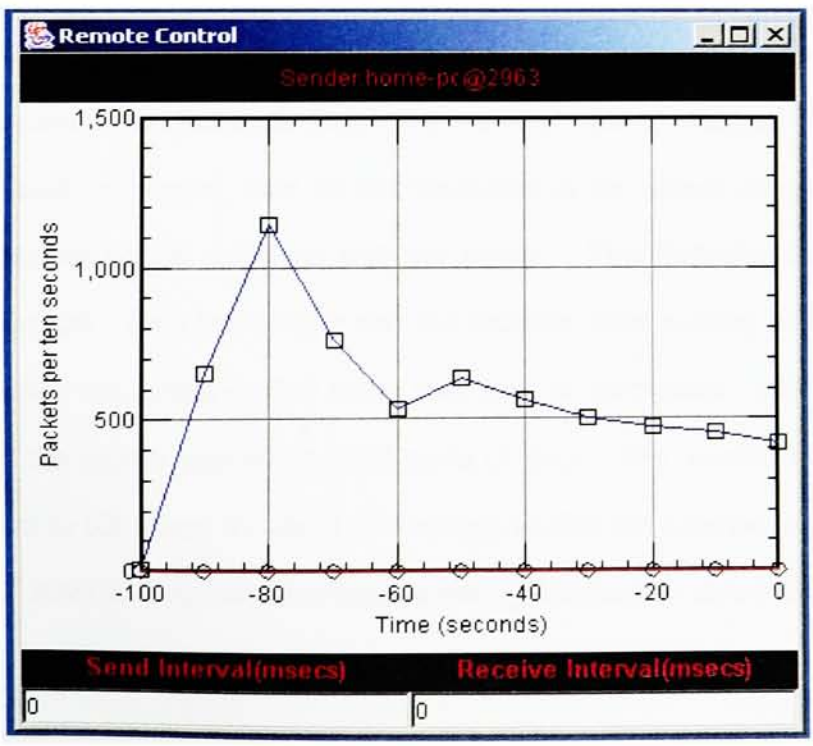

Figure 25: Sender with zero msec throughput-control delay, zero msec delays between packets

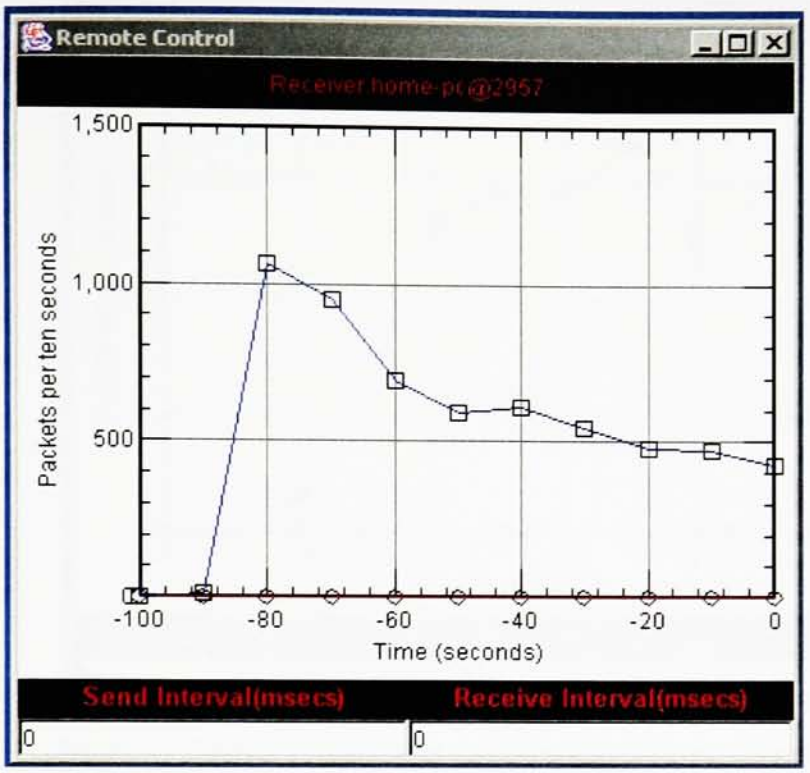

Figure 26: Receiving at zero msec throughput-control delay, without drawing squares

6.1.3 Receiver slower than sender and recovery

In all the cases when the receiver is slower than the sender, the messages get queued up at the receiver and the application does not see these messages. When the delay times return back to normal, then all the messages in the queue are sent to the receiver, so the receiver should catch up with the sender. This behavior is shown in Figure 27 and Figure 28. Both the sender and the receiver were running on the same machine. The sender throughput-control delay was kept at zero msec, and the delay between packets at the sender was set to 1000 msec (1 sec). The receiver throughput-control delay was set to 500 msec for about 20 seconds so that the receiver queue would fill up with packets. After 20 seconds, the receiver delay was reset to zero msec, and we can see from the graph that the receiver catches up to the sender without any problems.

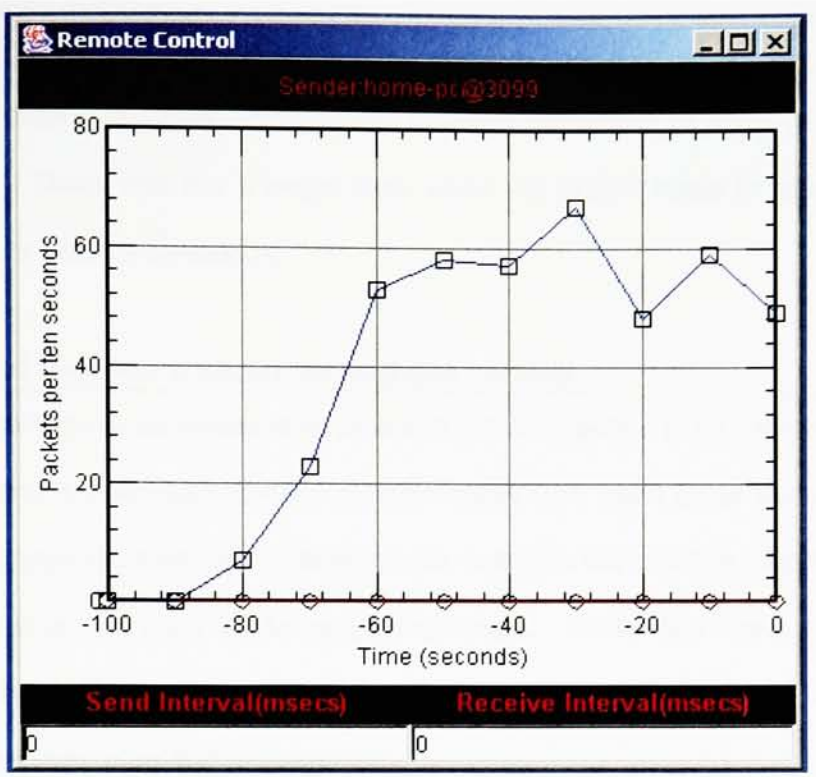

Figure 27: The plot for the sender when receiver is slower than sender

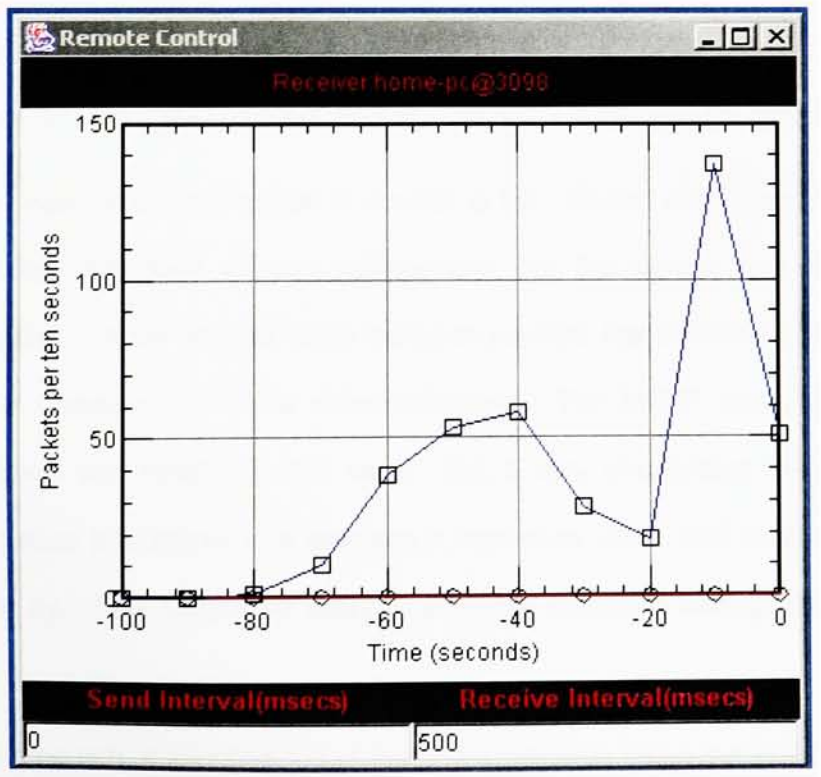

Figure 28: Plot for receiver when the receiver slows down and catches up later

6.2 Conclusions

The M2MP was thoroughly tested under different conditions of send and receive throughput values. There was not a single case when the M2MP failed to recover from unreliable transmit or receive conditions.

6.2.1 Sender and receiver with no throughput control

This test condition is summarized in section 6.1.1 and section 6.1.2. It can be seen that the receiver and sender successfully communicates with each other during normal conditions. In this case the throughput channel was built into the pipeline, but the delays for the throughput channels were set to zero milliseconds. In the first case, the sender was slow in sending packets (simulating a case of slow data transmission). The delay between packets was set to be 800 milliseconds.

This section simulated a wireless case wherein two peers are communicating with each other, without any significant problems like losing packets or retransmits. Another variation of this case with multiple receivers and one sender was tested with similar results.

The second case was summarized in section 6.1.2. During this testing, the send and throughput delay were kept at zero milliseconds, but the sender was allowed to transmit at full throttle. There was no delay between packets transmitted by the sender (simulating an FTP operation, or voice communication). The M2MP was successfully tested during for send and receive in this case. But a very unexpected behavior was observed, with the send throughput rate reaching a maximum value, and slowly dropping down as time went by. This happened every time during repetitive testing, and also on various platforms.

After some research, it was concluded that the application using the M2MP was not responsible for this. During continued research, the throughput layer was removed from

the pipeline and a loopback channel added to disprove that the queue data-structure in the loopback channel is causing this behavior. It was concluded that the operating system, or the LAN might be causing this behavior, and it is recommended that further research be performed on this behavior.

Conclusively, these tests proves that normal wireless communication using M2MP would work well in an ad hoc wireless network, with all the receivers staying within communicable range of the sender.

6.2.2 Sender faster than receiver

Some of the test cases included no delay in send and a high delay for receives as summarized in section 6.1.3. This case would be simulating a WLAN wherein the receiver gets the packets at a much slower rate than the sender is transmitting, and then the receiver catches up with the sender. This case could happen in an ad hoc wireless network if the receiver is far away from the sender and slowly moving closer to the source of the transmission.

The receiver gets the packet at a much slower rate, but nevertheless is successful in receiving and sending the packet up to the application layer. When the conditions returned to normal, the receiver always caught up with the sender by receiving all the messages that had been queued up, as can be seen in Figure 28.

6.2.3 Receiver faster than sender

This case is an ideal case for the receiver, and the behavior was the same as summarized in section 6.1.1. If the receiver is faster than the sender, then it successfully receives any transmitted messages immediately and processes it.

This case could occur in a peer-to-peer communication when a wireless device with a faster CPU and more memory is receiving data from a slower device. This case of testing showed no problem areas that needed further research.

6.3 Recommendations

Some of the recommendations for future research work that could be performed as a continuation of this research are discussed in this section:

- One of the very distinctive behaviors observed was that the send throughput, when sending at maximum speed, peaked to a maximum and then instead of staying at that rate, slowly dropped down. Some speculation was done regarding this behavior that is shown in Figure 25. The reason for this behavior could be the network protocols used in the routers, or limitations of the physical layer. Future studies can be done to find out why the throughput does not stay at a maximum rate.
- The throughput layer could be modified, or another layer added, that would drop packets during send or receive. The application can be tested if it recovers well from these lost packets.
- A reliability layer could be added to the M2MP that would repair any packets that could be lost in an unreliable wireless environment. Algorithms could be used to recover lost data in such a way that this mechanism is transparent to the application programmer. Existing methods and algorithms could be used, or new ones developed so that damaged packets could be repaired using some encoding or error correcting schemes. One such example is the Forward Error Correction (FEC) techniques.

- In the current project, there is no upper bound to the size of the messages that could be queued. The queue grows in size until the program runs out of memory. This could be changed so that once the upper bound is reached; the queue could start losing packets.
- The queue data-structure used in the throughput layer of this project could be replaced with a better data-structure, so that it is more efficient in storing data without the intensive copying operations.
- New applications could be developed, or the current application could be modified to be more robust for the dropping-packets project, so that if a packet is dropped, then it would be marked on the drawing window, and also that packet info could be logged to a text file.

7 User's Guide

7.1 Program Description

The crux of the research was to modify the M2MP to be able to simulate a wireless environment, and also give us a very controlled testing environment. The new M2MP is used by the communication watcher application described in section 5.1. There are two parts to using this system.

1. The RemoteControl server has to be running on a well-known host and listening for clients on a well-known port.
2. The communication watcher applications can be run either as *sender-only, receiver-only* or *sender and receiver*. This application uses the API offered by the M2MP for all its communication. The M2MP is included as a jar file in the application's classpath.

The user can control the application sending data by specifying the delay between the packets (in milliseconds). The user at the server will be notified when a client connects by popping up a window with the remote controls and a plot to monitor the communication. The user at the remote control can change the values of the send and receive throughput for each client connected to the server, so that M2MP can be tested for varying conditions of message throughput.

7.2 Input / Output

To execute the programs, the following steps should be followed. Please note that any examples shown below are for the windows platform, and you will have to customize the commands and scripts to work for any other platform.

7.2.1 Remote Control

To be able to run the RemoteControl server, you will need to make sure the Plot2D.jar file is in the classpath. This jar file contains the code for drawing 2D plots in color. The only other thing that the *input* needs is the port number that the server would listen for connections on. The commands you would need to run the RemoteControl would look something like this in a windows environment:

```
set CLASSPATH=./lib/Plot2D.jar;%classpath%
java edu.rit.rc.RemoteControl <port_number>
```

The code that handles clients trying to connect to the server is shown below:

```
serverSocket = new ServerSocket(portNumber);
serverThread = new Thread(this);
serverThread.start();
waitForCommand();
```

The thread continuously monitors for clients and spawns a new RemoteService thread for each client. This code is shown below:

```
new RemoteService(serverSocket.accept()).start();
```

The *output* would be some information messages that would show up in the command window as shown in Figure 29. The RemoteControl window (Figure 7) would open up for each client that successfully connects to the server. You could hit *Enter* on the command window at any point of time to sever the connections with the clients and

shutdown the server. If a client disconnects, the server would print this information out, and would close its own RemoteControl window for that client.

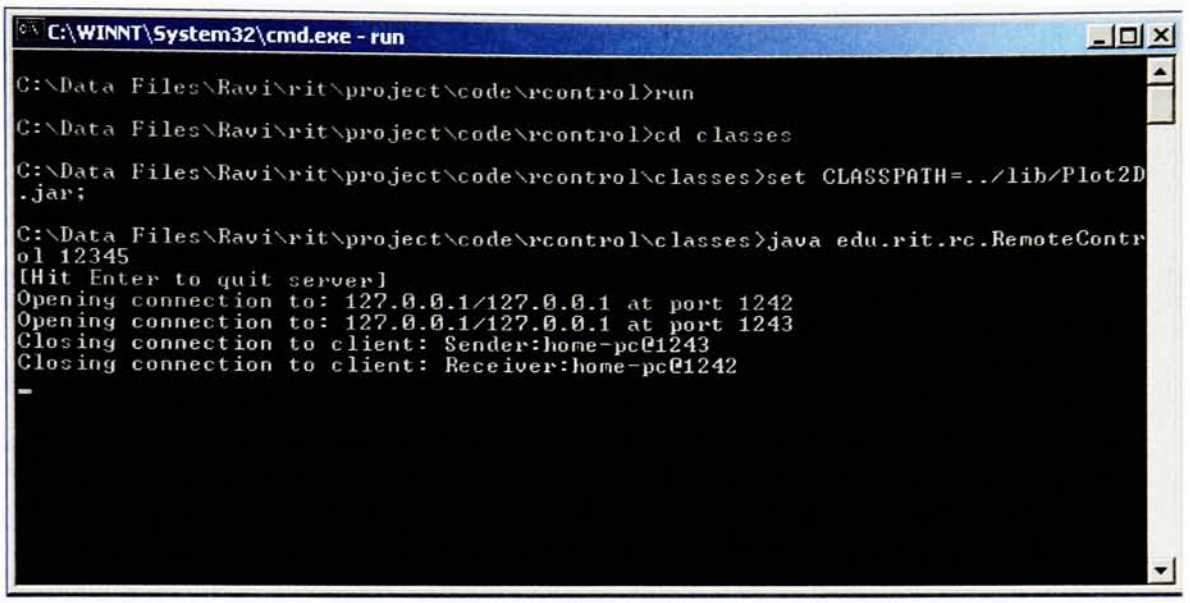

Figure 29: The RemoteControl command window

7.2.2 Communication Watcher

Once the RemoteControl server is running, the communication watcher clients can be started up. The clients could be configured to run in *sender-only, receiver-only* or *sender-receiver* modes. The m2mp.jar needs to be in the classpath for the communication watcher to be able to use the protocol. The command-line parameters needed are the host name and the port on which the RemoteControl server is listening for clients, and the mode in which the application should run (sender-only, receiver-only or sender-receiver).

The application would have to explicitly specify the inclusion of the throughput channel into the pipeline. The code that adds the throughput channel to the pipeline is shown below:

```
theChannel = new IPMulticastChannel (new IPChannelAddress
        (InetAddress.getByName(theIPMulticastAddress), thePort));
```

tpChannel = new TPChannel(theChannel, RCHostName, RCHostPort, appType);

The batch file to execute the application would look something like:

```
set CLASSPATH=../lib/m2mp.jar;
java commwatcher.CommWatcher -receiverOnly -host:Localhost
-port:12345
```

The *output* for this application is mainly the GUI that draws squares to indicate messages being sent or received. If the application is started as a sender-only, then the GUI is as shown in Figure 19. The sender has a text field wherein the user can specify the delay between packets being sent, the default being 1000 milliseconds. The GUI for the receiver looks exactly like the sender except that it does not have the text field for the delay to be specified.

7.3 Program Limitations

The throughput layer design was finalized after many iterations of prototype coding. A lot of limitations in the earlier versions were worked out of the system so that the M2MP internals are not changed, and yet, the system is robust enough for throughput testing.

One of the most important limitations of this project is the usage of the queue data structure. The queue is created to temporarily hold incoming messages. When the queue is instantiated, it has an initial capacity of ten. When more messages have to be queued up, the queue size grows. This is performed by copying arrays, and hence can be very inefficient for large queue sizes.

Another limitation is that there is no upper bound to the number of messages that can be queued. The queue grows in size until the program runs out of memory. This could be changed so that once the upper bound is reached; the queue could start losing packets.

8 Glossary

Acronym	Definition
ABR	Associativity-Based Routing.
AODV	Ad-Hoc On-Demand Distance Vector Routing.
AP	Access Point, a bridge between LAN and WLAN.
CDMA	Code Division Multiple Access.
CDPD	Cellular Digital Packet Data.
CGSR	Clusterhead Gateway Switch Routing.
CSMA / CA	Carrier Sense Multiple Access / Collision Avoidance
CPU	Central Processing Unit.
DSDV	Destination-Sequenced Distance-Vector Routing.
DSR	Dynamic Source Routing.
FEC	Forward Error Correction.
FreeBSD	FreeBSD is an advanced operating system for Intel ia32 compatible, DEC Alpha, and PC-98 architectures. It is derived from BSD UNIX, the version of UNIX developed at the University of California, Berkeley.
FTP	File Transfer Protocol.
GPRS	General Packet Radio Service.
GSM	Global system for Mobile Communications.
GUI	Graphical User Interface. Any window that provides feedback to the user and collects user input is known as a GUI.
J2SE	Java 2 Standard Edition
JAR	JAVA Archive.
LAN	Local Area Network, a group of computers located in close vicinity and hooked up to communicate.
M2MI	Many-to-Many Invocation.
M2MP	Many-to-Many Protocol. This protocol is an experimental protocol to be tested in small hand-held devices.
M2MRMI	Many-to-Many Remote Method Invocation. Similar to Remote Method Invocation, but using the M2MP.

Acronym	Definition
MANET	Mobile Ad hoc NETwork. A MANET is a WLAN in which network devices are participating only when they are in the vicinity of the network.
MVC	Model-View-Controller, a observer-observable kind of pattern commonly used to develop GUI applications.
OS	Operating System.
PAN	Personal Area Network.
PDA	Personal Digital Assistant.
PvC	Pervasive Computing.
RMI	Remote Method Invocation.
SSR	Signal Stability Routing.
TORA	Temporally-Ordered Routing Algorithm.
UML	Unified Modeling Language. A "best practice" graphical notation for expressing software analysis and design concepts.
WCDMA	Wideband Code Division Multiple Access.
Wireless WAN	Wireless Wide Area Network, for example, cellular phones.
WLAN	Wireless Local Area Network, a group of computers in close vicinity communicating wirelessly with each other.
WRP	Wireless Routing Protocol.

Missing Page

9 References

[1] Kaminsky, Alan., Bischof, Hans-Peter., "The Many-to-Many Protocol Library,". Electronically available on the Internet at http://www.cs.rit.edu/~anhinga/m2mp.shtml. Downloaded October 23, 2001.

[2] Molta, Dave., "The Survivor's Guide to 2001: Mobile and Wireless Technology," Senior Editor, Network Computing. Electronically published on the Internet at http://www.networkcomputing.com/1125/1125mobile1.html. Read on April 24, 2002.

[3] Kaminsky, Alan., Bischof, Hans-Peter., "Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems," 17th annual ACM conference on OOPSLA (Object Oriented Programming Systems, Languages, and Applications). Scheduled to appear November 2002. Electronically published on the Internet at http://www.cs.rit.edu/~anhinga/publications/m2mi20020716.pdf. Read on July 20, 2002.

[4] Kaminsky, Alan., "Infrastructure for Distributed Applications in Ad Hoc Networks of Small Mobile Wireless Devices," Technical Report, Information Technology Laboratory, Rochester Institute of Technology. Electronically published on the Internet at http://www.cs.rit.edu/~anhinga/publications/AnhingaPaper20010522.pdf. Read on November 23, 2001.

[5] Kaminsky, Alan., "Many-to-Many Invocation: A New Paradigm for Ad Hoc Collaborative Systems," Technical Report, Information Technology Laboratory, Rochester Institute of Technology. Electronically published on the Internet at http://www.cs.rit.edu/~anhinga/publications/m2mi20020206.pdf. Read on May 10, 2002.

[6] Coulouris, George., Dollimore, Jean., and Kindberg, Tim., "Distributed Systems, Concepts and Design," Addison Wesley, 2001.

[7] Main, Micheal., "Data structures and other objects using JAVA," Addison Wesley, 1998.

[8] Huwaldt, Joseph., "The plotting libraray," LGPL licensed software that was available free of cost on the internet. Available for download at http://www.morphon.com/software/madymo/Plot2D1.2.jar. Downloaded on October 23, 2001.

[9] "What is a Wireless LAN?" White paper, Proxim, Inc., Mountain View, California, USA. Electronically published on the Internet at http://www.proxim.com/learn/library/whitepapers/pdf/whatwlan.pdf. Read on June 1, 2002.

[10] Geier, Jim., "Benefits of the IEEE 802.11 Standard," White paper, Wireless-nets consulting services. Electronically published on the Internet at http://www.wireless-nets.com/articles/whitepaper_benefits_80211.htm. Read on June 18, 2002.

[11] Zyren, Jim., Petrick, Al., "IEEE 802.11 Tutorial," Technical Report, Wireless Ethernet Compatibility Alliance. Electronically published on the Internet at http://www.weca.net/downloads/IEEE_80211_Primer.pdf. Read on June 18, 2002.

[12] Rizzo, Luigi., "Dummynet: a simple approach to the evaluation of network protocols," University of Pisa, Pisa, Italy. ACM Computer Communication Review. Electronically published on the Internet at http://citeseer.nj.nec.com/rizzo97dummynet.html. Read on June 25, 2002.

[13] Rizzo, Luigi., "IP Dummynet," University of Pisa, Pisa, Italy. Electronically published on the Internet at http://info.iet.unipi.it/~luigi/ip_dummynet. Read on June 25, 2002.

[14] Donahoo, Michael., "DummyNet, a free BSD system," Department of Computer Science, Baylor University, Waco, Texas, USA. Electronically published on the Internet at http://cs.baylor.edu/~donahoo/tools/dummy. Read on June 25, 2002.

[15] Ni, Sze-Yao., Tseng, Yu-Chee., Chen, Yuh-Shyan., Sheu, Jang-Ping., "The Broadcast Storm Problem in a Mobile Ad Hoc Netowrk," National Central University, Department of Computer Science and Information Engineering, Chung-Li, Taiwan. Electronically published on the Internet at http://www.cs.cornell.edu/People/egs/615/p151-ni.pdf. Read on July 1, 2002.

[16] Basagni, Stefano., "On the broadcast and clustering problems in peer-to-peer networks," Doctoral Dissertation, Department of Information Technology, University of Milan and Torino, Italy. Electronically published on the Internet at http://www.sigmobile.org/phd/1998/theses/basagni.pdf. Read on July 14, 2002.

[17] Royer, Elizabeth., Toh, C-K., "A review of current routing protocols for Ad-Hoc mobile wireless networks," White paper, Department of Electrical and Computer Engineering, University of California at Santa Barbara and Georgica Institute of Technology at Atlanta. Electronically published on the Internet at http://www.ee.surrey.ac.uk/Personal/G.Aggelou/PAPERS/Adhoc_Review.pdf. Read on July 14, 2002.

[18] Perkins, Charles., Bhagwat, Pravin., "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," T.J. Watson Research Center, IBM, Hawthrone, NY. Electronically published on the Internet at http://www.cise.ufl.edu/~helal/6930F01/papers/DSDV.pdf. Read on August 25, 2002.

[19] Murthy, S., Garcia-Luna-Aceves., "An Efficient Routing Protocol for Wireless Networks (CGSR)," ACM Mobile Networks and App. J., Special issue on Routing in Mobile Communication Networks, Oct 1996, pp. 183-97.

[20] Ishibashi, Brent., "Routing in Ad Hoc Networks," Presentation, Broadband Communications Research Group, University of Waterloo, Canada. Electronically published on the Internet at http://bbcr.uwaterloo.ca/~rboutaba/cs756M/Adhoc-routing.pdf. Read on August 25, 2002.

[21] Park, VD., Corson, MS., "A highly adaptive distributed routing algorithm for mobile wireless networks," Proceedings of INFOCOM'97, April 1997. Electronically published on the Internet at http://www.ics.uci.edu/~atm/adhoc/paper-collection/corson-adaptive-routing-infocom97.pdf. Read on August 25, 2002.

[22] Dube, R., "Signal Stability based adaptive routing for Ad Hoc Mobile Networks," IEEE Pers. Comm., Feb. 1997, pp. 36-45. Electronically published on the Internet at http://www.cs.umd.edu/projects/mcml/papers/pcm97.ps. Read on August 25, 2002.

[23] Savarese, Daniel., "Whatever happened to Jini?," Discussion, JavaPro, A Fawcette Technical Publication, August 2002, Vol. 6, No. 8, pp. 70,71.

[24] Vichr, Roman., Malhotra, Vivek., "Jini and PvC: Problems and promise," JAVA technology articles, IBM developer works. Electronically published on the Internet at http://www-106.ibm.com/developerworks/wireless/library/wi-jini.html?loc=dwmain. Read on August 26, 2002.

[25] Corson, S., Macker, J., "MANET: Routing Protocol Performance Issues and Evaluation Considerations," Request For Comments (RFC) 2501. Electronically published on the Internet at http://www.ietf.org/rfc/rfc2501.txt. Read on July 16, 2002.

[26] Nielsen, Henrik., Gettys, Jim., Baird-Smith, Anselm., Prud'hommeau, Eric., Lie, Hakon., Lilley, Chirs., "Network Performance Effects of HTTP/1.1, CSS1, and PNG". Electronically published on the Internet at http://www.w3c.org/Protocols/HTTP/Performance/Pipeline.html. Read on July 16, 2002.

[27] Xie, Peter Ping., "Network Protocol Performance Evaluation of Ipv6 for Windows NT," Thesis, California Polytechnic State University, San Luis Obispo. Electronically published on the Internet at http://www.ee.calpoly.edu/3comproject/masters-thesis/Xie-Peter.pdf. Read on July 16, 2002.

[28] Ely, David., Savage, Stefan., and Wetherall, David., "Alpine: A User-Level Infrastructure for Network Protocol Development," White paper, Department of Computer Science and Engineering, University of Washington, Seattle, WA. Electronically published on the Internet at http://citeseer.nj.nec.com/ely01alpine.html. Read on July 16, 2002.

[29] Fiuczynski, M., Bershad, B., "An extensible protocol architecture for application-specific networking," USENIX 1996 Annual Technical Conference, San Diego, California, January 1996.

[30] Burbeck, Steve., "Applications Programming in Smalltalk – 80: How to use MVC". Electronically published on the Internet at http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. Read on December 10, 2001.

[31] "Your patterns library", Patterns homepage. Electronically published on the Internet at http://hillside.net/patterns. Read on December 10, 2001.

[32] Baray, Cristobal., "The MVC design Pattern," Indiana University. Electronically published on the Internet at http://www.cs.indiana.edu/~cbaray/projects/mvc.html. Read on December 10, 2001.

[33] Sundaram, Karthik., "Whither Wireless," An article featuring the future of wireless, SiliconIndia magazine, July 2002.

[34] Garg, Gaurav., "Mobile Wireless – boldly going to places where no man has gone before," An article featuring the reality of wireless technological development, SiliconIndia magazine, July 2002.

[35] Kaminsky, Alan., Bischof, Hans-Peter., "Anhinga Project Downloads," Downloads for M2MI, M2MP, RIT class file etc. Electronically available on the Internet at http://www.cs.rit.edu/~anhinga/downloads/downloads.shtml.

[36] "The Anhinga Project Home Page," Department of computer science, Rochester Institute of Technology. Accessible on the web at http://www.cs.rit.edu/~anhinga.