UNIVERSITY OF CINCINNATI

Date: 02/28/2006

I, Akash Bhatia,

hereby submit this work as part of the requirements for the degree of:

Master of Science

in:

Civil Engineering

It is entitled:

A Modified Wireless Token Ring Protocol To Prevent Data Collision in Wireless Traffic Sensors

This work and its defense approved by:

Chair: Dr. Heng Wei

Dr. Prahlad D. Pant

Dr. Yizong Cheng

A MODIFIED WIRELESS TOKEN RING PROTOCOL TO PREVENT DATA COLLISION IN WIRELESS TRAFFIC SENSORS

A thesis submitted to the

Department of Civil & Environmental Engineering

College of Engineering

University of Cincinnati

in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

March, 2006

By

Akash Bhatia

B.E. in Civil Engineering

University of Pune, India, 1998

Committee Chair: Dr. Prahlad D. Pant

Abstract

Intelligent Transportation Systems, or ITS, is about using technology to solve transportation engineering problems. There is a major push by the Departments of Transportation to use wireless sensors all over the highways to provide accurate traffic information in real-time.

Wireless Token Ring Protocol (WTRP), a wireless protocol based loosely on IEEE 802.4 [8] Token Bus Protocol, is an excellent Medium Access Control layer level protocol to prevent data collision in this wireless sensor network on the freeways. The distances between wireless sensors on the freeway were calculated, based on sensor frequency, sensitivity of the receiver and the transmitters, using the Free Space Path Loss Equation. Based on these calculations, it was found that Wireless Token Ring Protocol could fail, as the distances between the sensors was quite large and the data packets got lost, when being transmitted from the last sensor back to the first sensor.

The Wireless Token Ring Protocol was modified such that instead of being uni-directional, transmitting data in one direction only; it became bi-directional, transmitting data in both directions. As such, during simulation, it was found that, the performance of the Modified Wireless Token Ring Protocol was better than that of Wireless Token Ring Protocol. The number of data packets that were lost in transmission or corrupted using Modified Wireless Token Ring Protocol was drastically less than that using Wireless

Token Ring Protocol, over the same node positions. Hence, the Modified Wireless Token Ring Protocol is more effective than the Wireless Token Ring Protocol to prevent data collision and corruption on the field.

Acknowledgements

First and foremost, I would like to thank my advisor Dr. Prahlad Pant for giving me this opportunity to work on my thesis and suggesting this very interesting problem to resolve. I also wish to express my gratitude to Dr. Yizong Cheng and Dr. Heng Wei for agreeing to serve on the committee, and providing me with invaluable feedback.

I would like to thank my sister Neetu and Arpita Majumder, without whose encouragement, this thesis wouldn’t have been completed.

I would like to thank Mustafa Ergen and Duke Lee, the original WTRP developers for providing their insight on making the changes

My special thanks to Nigel Pratt, Ramesh Panchagnula, Sreeram Kumar, J.P Raichura for providing me with invaluable tips and knowledge about networks, allowing me to complete my thesis on time.

Lastly, I would like to thank my parents, who constantly supported and encouraged me as I progressed in my thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

1.1 Introduction

One needs to look no further than new road and bridge construction to see how Intelligent Transportation Systems (ITS) have become an integral part of every new major transportation project. From coast to coast, highway officials and Departments of Transportation are including ITS into project designs not only to lessen the inconvenience to motorists affected by construction, but also to ensure safety before and after the completion of the project.

The goal of ITS is to build a future where people and goods are transported without delay, injury or fatality by integrated systems that are built and operated to be safe, cost effective, efficient and secure. It tries to achieve this by promoting research, deployment and operation of intelligent transportation systems through leadership and partnerships with public, private, educational and consumer stakeholders.

ITS, encompasses a broad range of wireless and wire line communications-based information, control and electronics technologies. When integrated into the transportation system infrastructure, and in vehicles themselves, these technologies help monitor and manage traffic flow, reduce congestion, provide alternate routes to travelers, enhance productivity, and save lives, time and money.

Intelligent Transportation Systems provide the tools for skilled transportation professionals to collect, analyze, and archive data about the performance of the system during the hours of peak use. Having this data enhances traffic operators' ability to respond to incidents, adverse weather or other capacity constricting events.

According to Gartner Dataquest (1), ITS can be used to provide information for homeland security, a subject that is of primary interest, post 9/11. According to Gartner Dataquest, transportation officials could collaborate with public safety officials to monitor transportation infrastructure such as roads, bridges, seaports and airports.

There is an increased focus on remote monitoring through the installation of cameras and sensors at key locations around the city (2). As such, wireless communications will play a huge part in the development of such monitoring capabilities.

This thesis involves researching the best wireless protocol that will enable various wireless sensors transmit data without the problem of data collision and partial connectivity due to large distances on the field. It also involves development of a program that will enable the data to be captured in a database on a computer.

1.2 Objective

The objective of this thesis is to research Wireless Token Ring Protocol for traffic sensors that will enable them to transmit data without the issue of data collision as well as modify it so that it can be used over large distances on the field and develop a Java program that will enable the data transmitted, to be inserted in a database on a computer.

1.3 Layout of thesis

This thesis can be divided into four sections

- Study on various wireless technologies available today.
- Research on Wireless Token Ring Protocol (WTRP) – A Medium Access Control (MAC) layer protocol to minimize data collision.
- Modification to WTRP for implementation on the field.
- Development of a J2ME program that will reside on the sensors and transmit data to a database.

Chapter 2

Literature Review

Past research in works have addressed the problems of collecting traffic data from construction zones and accident sites on a freeway. The problem of real time data collection has been extensively examined ever since Intelligent Transportation Systems was born.

Extensive research has been done at the University of California, Berkeley in Intelligent Transportation Systems, particularly in the field of data collision prevention on the field. Duke Lee, et al, have researched the use of Wireless Token Ring Protocol for ITS applications, which forms the basis for the protocol aspect of this thesis (3). Mustafa Ergen, et al, provide a good comparison in performance between Wireless Token Ring Protocol (WTRP) and IEEE 802.11(4). Mustafa Ergen's Masters Thesis at the University of California at Berkeley provides an excellent insight into the working of WTRP and also provides some extensions, one of which has been utilized in this thesis (5). IEEE 802.4 protocol, on which WTRP is based on, is explained in detail in the International Standards paper (6).

Research on various technologies for highway traffic monitoring was conducted by Mark P. Gardner in the Committee on Highway Traffic monitoring, led by David L. Huft, South Dakota Department of Transportation (2). States like Maryland piloted and adopted the state-of-art lane closure technology (7). Armand J. Ciccarelli III studied the impact of wireless technologies on Traffic Data Collection, Dissemination and Use (8). Cheung, Coleri, et al, have done extensive research in the field of traffic measurement and vehicle classification with a single magnetic sensor (9).

Business Week has mentioned about the various wireless technologies available today and their feasibility (10). Qusay Mahmoud has researched the use of Java in WAP (Wireless Application Protocol) (11). Ken Noblitt of Cambridge Silicon Radio has researched the similarities and differences in Bluetooth and IEEE 802.11 Technologies (12). The consortium for Bluetooth development provides the technical information and possible applications for Bluetooth (13). J. Bray and C.F. Sturman provide developers with the technical specifications for Bluetooth development, complete with all facets of the technology, beginning with the antenna, through the protocol stack and its interfaces, to the applications and the user interface that drives them (14). Peter Rysavy, of Rysavy Research, describes the technology capabilities of General Packet Radio Service (GPRS), Enhanced Data Rates for Global Evolution (EDGE), Universal Mobile Telecommunications Systems/Wideband Code Division Multiple Access (UMTS/WCDMA), the deployment of these technologies, the market adoption and the challenges involved with these technologies giving a great insight into the world of 3G wireless (15). http://java.sun.com introduces various wireless technologies that are compatible with Java, and also throws light on the concept of Code Division Multiple Access/Time Division Multiple Access (CDMA/TDMA) for effective data transmission with minimum collisions (16).

Michael Juntao Yuan has explained in detail about developing mobile applications on the Java platform (17). http://java.sun.com is an excellent resource for developers looking to develop applications for the wireless world (18). Jonathan Knudsen, in his book, gives insight into J2ME, the stripped down version of J2EE (19).

Goldsmith and Wicker explain a great deal of information on wireless sensor networks, in the IEEE magazine of August 2002 (20). The concept of Directed Diffusion in sensor networks is detailed by Chalermek in Directed Diffusion for Wireless Sensor Networking (21). Another solution to the problem of data collision in the form of Data Funneling is proposed by Dragan Petrović, et al (22). Y.C.Tae, from the National University of Singapore provides a solution to the problem of data collision by proposing CSMA (23).

The CDMA group proposes CDMA as the technology to minimize data collision (24) as does Qualcomm CDMA Technologies (25). International Engineering Consortium (IEC) (26) and Nortel Networks propose TDMA as the solution to the data collision problem (27).

The Oracle Technology Network supplies a lot of information on Oracle Spatial, and how an application can be built around the GPS navigation unit in the car (28). Oracle Spatial user's guide provides a valuable insight into the working of Oracle Spatial, a key component in this application (29). Liujian Qian, in his white paper on Oracle Spatial, gives a good example of how to develop a GPS based application using Oracle Spatial and Mapviewer (30).

Chapter 3

Wireless Technologies

There are a number of wireless technologies today, each presenting a certain level of efficiency, cost-effectiveness and viability. An attempt is made to present some of the leading wireless technologies that exist today. Eventually, the technology will be selected based upon its usability and viability, with an eye on the future direction of technology.

3.1 Bluetooth

Bluetooth is a computing and telecommunications industry specification that describes how wireless devices (mobile phones, PDAs and in our case, the central computer and the sensors) can easily interconnect with each other using a short-range wireless connection. Bluetooth devices can easily be synchronized with information in a desktop or a laptop, initiate the sending and receiving of faxes, and in general allow mobile devices to be totally coordinated. The technology requires that a low-cost transceiver chip be included in each device.

Bluetooth can support piconets[1] of up to eight devices, with a maximum of three synchronous-connection oriented (SCO) links. It also supports asynchronous connection links (ACLs) used to exchange data in non time-critical applications. It uses frequency hopping spread spectrum (FHSS) technique, and hops at a rate of 1600 hops/sec and uses Guassian frequency shift keying (GFSK) modulation (14).

[1] Piconet – network of slaves and master in a Bluetooth architecture

Bluetooth protocol specification suggests setting up of a small network, or piconets, of Bluetooth enabled devices. This piconet consists of a master and up to seven slave devices. At any given moment only one of these eight devices can transmit. If master has something to transmit to one of the slave it will do that and if it doesn't have anything to transmit it will poll all the slaves one by one and ask them to transmit if they have any information. The slave can respond to time slot with information or NULL (i.e. it has no data to transmit). This scheme avoids interference among the devices in the piconet, thus avoiding data collision (14).

3.2 802.11b Wireless Local Area Networking (Wi -Fi)

In wireless LAN (W LAN) technology, 802.11 refer to a family of specifications developed by a working group of the Institute of Electrical and Electronics Engineers (IEEE). There are three specifications in the family: 802.11, 802.11a, and 802.11b.
The 802.11 and 802.11b specifications apply to wireless Ethernet LANs and operate at frequencies in the 2.4-GHz region of the radio spectrum. Data speeds are generally 1 Mbps or 2 Mbps for 802.11 and 5.5 Mbps or 11 Mbps for 802.11b, although speeds up to about 20 Mbps are realizable with 802.11b.

802.11b supports true multipoint networking with such data types as broadcast, multicast, and uni-cast packets. The standard practice is approximately one access point (AP) to every 10-20 stations, but the MAC[2] address built in every device allows for virtually unlimited number of devices to be active in a given network. Carrier Sense Multiple

[2] MAC – Media Access Control (on the Ethernet LAN, it is the same as Ethernet Address)

Access with Collision Avoidance (CSMA/CA) is used to control the common channel and avoid collisions. (12)

	802.11b	***Bluetooth***
Typical usage	• Wireless version of the Ethernet • Long Range wireless network access	• Personal cable replacement • Mid-range wireless data exchange and network access
Bandwidth Effective Bandwidth	• 11Mbps, shared • 2 to 3 Mbps with WEP enabled	• 1 Mbps, shared • 700 Kbps, worse due to interference
Interference	Other RF devices, building material, equipment	Other RF devices, building material, equipment
Security	Unsecured unless well protected. Subject to network sniffing, session hi-jacking and unauthorized access. Rely on application level authorization and encryption	Less Secure
Supporting Devices	• Some new laptops have it built-in • PDA: Requires external hardware	• Few Laptops have them built in**.** • Require external hardware
Real –time network access	User does not have to be near an access point	User has to be within 10 meters of an access point
Range	100 meters	10 meters
Component Cost	~ $25.00	~ $ 20.00. Expected to fall to $ 5.00 in a few years
Application Support	TCP/IP	TCP/IP, OBEX[3]
Number of device access at the same time	Multiple, Shared	Up to 8, Shared

Table 3.1: Comparison of 802.11b and Bluetooth (12)

[3] OBEX: Object Exchange – A session protocol that can be described as a binary HTTP. It is optimized for ad-hoc wireless links and can be used to exchange files, pictures, calendar entries (vCal) and business cards (vCard) – Source : http://www.irda.org

As can be seen from the comparison above, Bluetooth supports a very short range (approximately 10m) and relatively low bandwidth (1MBPS). In practice, Bluetooth networks PDAs or cell phones with PC's but is not used widely for general-purpose wireless networking. The very low cost of manufacturing of Bluetooth makes it a viable solution for small wireless devices like PDAs and cell phones. Since this thesis is involved with sensors and central computers that will be placed at a considerable distance from each other, Bluetooth is not a technology that will help us in our quest. Similarly, the limitation of the range of 802.11b (100 m) and its bandwidth (11MBPS) is also not suitable for our project.

3.3 Wireless Cellular Networks

The wireless market has experienced a phenomenal growth since the first second-generation (2G) digital cellular networks, based on global system for mobile communications (GSM) technology, were introduced in the early 1990's. Since then, GSM has become the dominant global 2G radio access standard. Almost 80% of today's new subscriptions take place in one or more than 400 cellular networks that use GSM technology. This growth has taken place simultaneously with the large experienced expansion of access to the Internet and its related multimedia services. (15)

Faced with a challenge to evolve their networks to efficiently support the demand for wireless Internet based multimedia services, cellular operators are performing a rapid technology evolution rolling out new third-generation (3G) radio-access technologies, capable of delivering such services in a competitive and cost-effective way.

There are multiple recognized 3G technologies, which can be summarized into two main 3G evolution paths. The first one to be conceived and developed, and the more widely supported, is UMTS (Universal Mobile Telecommunications System), and the second one is CDMA2000. (15)

3.3.1 2G Mobile Systems

Compared to the first generation systems, second generation (2G) use digital multiple access technology, such as TDMA (time division multiple access) and CDMA (Code

Division Multiple Access). Global System for Mobile Communications, or GSM uses TDMA technology to support multiple users. (15)

3.3.2 2.5G Mobile Systems

The move into the 2.5G world began with General Packet Radio Service (GPRS). GPRS is a radio technology for GSM networks that adds packet switching protocols, shorter setup time for ISP connections, and the possibility to charge by the amount of data sent, rather than connection time. Packet switching is a technique whereby the information (voice or data) to be sent is broken up into packets, of at most a few Kbytes each, which are then routed by the network between different destinations based on addressing data within each packet. Use of network resources is optimized, as the resources are needed only during the handling of each packet. (15)

3.3.3 3G Mobile Systems

Third Generation mobile systems are faced with several challenging technical issues, such as the provision of seamless services across both wired and wireless networks and universal mobility. In Europe, there are three evolving networks under investigation:

1. UMTS (Universal Mobile Telecommunications Systems)
2. MBS (Mobile Broadband Systems)
3. WLAN (Wireless Local Area Networks)

The use of hierarchical cell structures is proposed. The overlaying of cell structures allows different rates of mobility to be serviced and handled by different cells. Advanced multiple access techniques are also being investigated, and two promising proposals have evolved, one based on wideband CDMA and other that uses a hybrid TDMA/CDMA/FDMA approach. (15)

3.4 Medium Access Protocols

In the ALOHA scheme, first used in the 1970's at the University of Hawaii, a node simply transmits a message when it desires. If it receives an acknowledgement, all is well. If not, the node waits a random time and re-transmits the message.

In *Frequency Division Multiple Access* (FDMA), different nodes have different carrier frequencies. Since frequency resources are divided, this decreases the bandwidth available for each node. FDMA also requires additional hardware and intelligence at each node.

In *Code Division Multiple Access* (CDMA), a unique code is used by each node to encode its messages. This increases the complexity of the transmitter and the receiver. (24)

In *Time Division Multiple Access* (TDMA), the Radio Frequency (RF) link is divided on a time axis, with each node being given a predetermined time slot it can use for communication. This decreases the sweep rate, but a major advantage is that TDMA can be implemented in software. All nodes require accurate, synchronized clocks for TDMA

CDMA and TDMA are two competing technologies that differ in the manner in which users share the common resource. TDMA does it by chopping up the channel into sequential time slices. Each user of the channel takes turns transmitting and receiving in a round-robin fashion. In reality, only one person is actually using the channel at any given moment, but he or she only uses it for short bursts. He then gives up the channel momentarily to allow the other users to have their turn. This is very similar to how a computer with just one processor can seem to run multiple applications simultaneously.

CDMA, on the other hand really, does let everyone transmit at the same time. Conventional wisdom would lead you to believe that this is simply not possible. Using conventional modulation techniques, it most certainly is impossible. What makes CDMA work is a special type of digital modulation called "Spread Spectrum". This form of modulation takes the user's stream of bits and splatters them across a very wide channel in a pseudo-random fashion. The "pseudo" part is very important here, since the receiver must be able to undo the randomization in order to collect the bits together in a coherent order. (24)

Imagine a room full of people, all trying to carry on one-on-one conversations. In TDMA each couple takes turns talking. They keep their turns short by saying only one sentence at a time. As there is never more than one person speaking in the room at any given moment, no one has to worry about being heard over the background din. In CDMA each couple talks at the same time, but they all use a different language. Because none of the listeners understand any language other than that of the individual to whom they are listening, the background din doesn't cause any real problem (24).

3.5 How TDMA works

TDMA relies upon the fact that the audio signal has been digitized; that is, divided into a number of milliseconds-long packets. It allocates a single frequency channel for a short time and then moves to another channel. The digital samples from a single transmitter occupy different time slots in several bands at the same time as shown in the figure below:

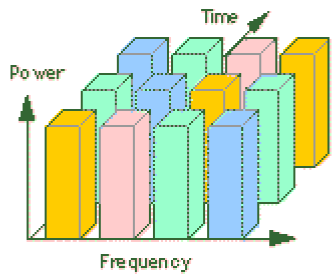


Figure 3.1: TDMA (27)

TDMA is the access technique used in the European digital standard, GSM, and the Japanese digital standard, personal digital cellular (PDC). The reason for choosing TDMA for all these standards was that it enables some vital features for system operation in an advanced cellular or PCS environment. Today, TDMA is an available, well-proven technique in commercial operation in many systems (53).

To illustrate the process, consider the following situation. The following figure shows four different, simultaneous conversations occurring.

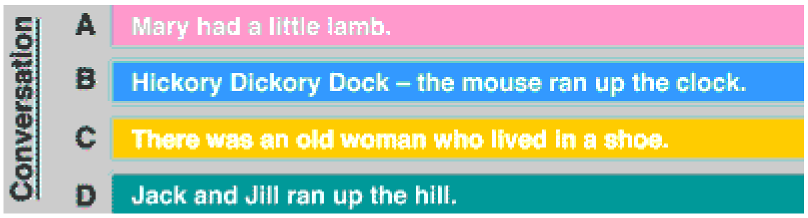

Conversation
A
Mary had a little lamb.
B
Hickory Dickory Dock – the mouse ran up the clock.
C
There was an old woman who lived in a shoe.
D
Jack and Jill ran up the hill.

Figure 3.2: Four Conversations – Four Channels (27)

A single channel can carry all four conversations if each conversation is divided into relatively short fragments, is assigned a time slot, and is transmitted in synchronized timed bursts as in the figure below. After the conversation in time-slot four is transmitted, the process is repeated.

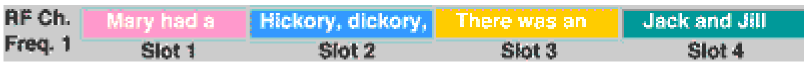


Figure 3.3: Four Conversations – One Channel (27)

3.6 Will TDMA solve the problem on the field?

Cellular systems work on the basis of a common clock residing on the carrier's computer. As a result TDMA is hugely successful for cellular phones. But sensor networks on the freeways are not clock synchronized, i.e. they don't have a single common clock or a single server for synchronization, and as such, time division method of preventing data collision is irrelevant. As a result, TDMA will not be applicable to this topology.

3.7 Traditional Radio

Radio is a simple technology. With just a couple of electronic components, a simple radio transmitter and receiver can be built. Radio waves transmit data invisibly through the air, over large distances.

All radios today, use continuous sine waves to transmit information. Continuous sine waves are used because there are thousands of devices and people using the radio waves at the same time. Each different radio signal uses a different sine wave frequency and that is how they are all separated.

A radio set up has two parts, the transmitter and the receiver. The transmitter takes the data, encodes it onto the sine wave, and transmits it with the radio waves. The receiver receives the radio waves and decodes the message from the sine wave it receives. Both the transmitter and the receiver use antennas to capture the radio signal.

This technology seems to be the simplest, and the sensors on the field use this technology to transmit the data. But with a number of sensors transmitting at the same time, what is needed is a protocol that will reduce the data collision. This thesis studies a protocol (Wireless Token Ring Protocol) that can be used by this traditional method of transmitting data, to transmit without data collision.

Chapter 4

4.1 Methodology

The methodology for this thesis is based on the increasing use of wireless communications in every sector, including Transportation Engineering. Traffic sensors have come a long way from loop detectors to the wireless sensors that are currently being implemented by the Departments of Transportation. With the combination of high-speed mobile access with Internet Protocol (IP) based services, the potential of wireless in transportation engineering is limitless.

Already, various organizations like the CALTRANS (California Department of Transportation) are incorporating wireless technologies to reduce congestion on freeways, most particularly, Highway 101, running through the heart of Silicon Valley.

There are many different wireless technologies available today, that present us with a number of options to choose from, and sometimes even confusing us over what technology is best suited for the problem at hand. But the problem on the field is unique, that about the ad-hoc topology of the sensor network as well as the lack of a clock for time coordination of the sensors. An effort is made to address this issue and research a wireless protocol that is best suited for these conditions.

The Wireless Token Ring Protocol (WTRP) developed at the University of California at Berkeley, is a very promising protocol satisfying most of our conditions. This protocol will be thoroughly studied, researched and modified to be able to implement it on the freeways. Some of the technologies that could be used are also studied. Some of the technologies that are available today are:

Bluetooth

GSM/GPRS/CDMA/TDMA (all 3G services)

Wi-Fi (802.11x)

Traditional Radio

The instruments of data collection for this study include:

Manuals on wireless technologies

World Wide Web

Wireless Communications Alliance

Sun Microsystems' Java for Wireless Applications

Past theses on use of wireless technologies in Transportation Engineering

4.2 Actual Research

The research for this thesis consists mainly of understanding how the Wireless Token Ring Protocol works and to modify it, so that it can be implemented on the field. The modification to the protocol should be such that the inherent advantages of the parent protocol (WTRP) are retained by the modified protocol, while making the protocol robust on the field.

The research for the thesis is carried out in four phases as in the layout:

Phase 1. Study of Wireless Token Ring Protocol (WTRP) and its working.
Phase 2. Modify WTRP, so it can be implemented on the field, with sensors placed at huge distances on the highway
Phase 3. Research various wireless technologies and protocols that are available today.
Phase 4. Develop a code that captures the data sent out by the sensors wirelessly, and displays it on a website and can be displayed on the Variable Message Signs (VMS).

Chapter 5

Background

Traffic data collection and sensor networks

Efficient management of highways and urban streets require accurate and timely information about traffic flow. Traditionally, traffic measurements have been extensively carried out using inductive loop detectors that are difficult to install and maintain. Also, closing lanes to cut loops into the pavement is also very disruptive.

Of late, more and more Departments of Transportation are utilizing magnetic or video sensors over Inductive Loop Detectors (ILD). In addition to vehicle count, occupancy and speed, the sensors yield traffic information (such as vehicle classification) that cannot be obtained from loop data. Practical measurements have concluded that by using sensors, the vehicle detection rate is 99% better than ILD's and estimates of vehicle length and speed appear to be better than 90%. Moreover, the measurements also give inter-vehicle spacing or headways, which reveal interesting phenomena as platoon formation downstream of a traffic signal or an incident zone. But these sensors need to be installed on the freeway and at intersections, and they would require a physical connection to a computer. This would involve a lot of cable infrastructure and would make the entire setup bulky and unmanageable. Hence, there is a move towards wireless sensors, that can be set up instantly, and almost anywhere on the freeway or near an intersection.

Extensive research is being carried out in the field of wireless sensors to obtain traffic information. In fact, these sensors can be set up practically anywhere to obtain crucial traffic information, be it at intersections, around an accident on a highway, or around a construction zone, etc, where the authorities can monitor traffic and hence reduce the occurrence of collisions and traffic backup.

Once this traffic data is collected, it is of utmost importance to process this data and have it available as soon as possible to vehicle owners. Traditionally, this data is collected by the Traffic Management Centers and then processed and displayed at the Variable Message Signs (VMS) or at traffic websites.

The problem of data collision

Sensors transmit data in the form of packets. These packets are accepted by the receiving station, which can be used to store in a database, display on the Variable Message Signs (VMS) or even transmit it to another sensor.

The problem arises when two or more sensors transmit data at the same time. A packet collision occurs causing the data packet to be either discarded or sent back to the originating station to be re-transmitted. Packet collisions can result in a loss of packet integrity or can impede the performance of the network.

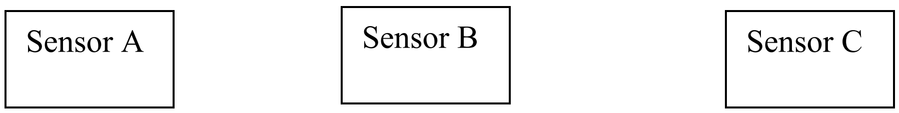
Chapter 6

Wireless Token Ring Protocol (WTRP)

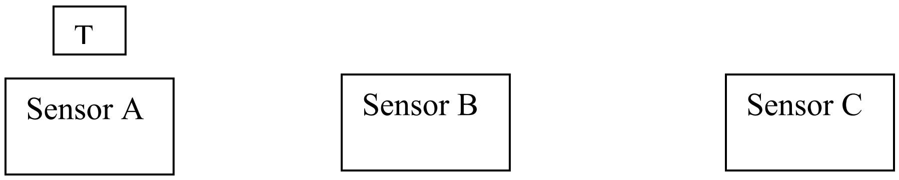
It is a distributed Medium Access Protocol for ad-hoc networks. There is no need for clock synchronization in this protocol. It is conceived specifically for ITS applications. Its distributed architecture, where the nodes need not be connected with a master, or a central server, makes it robust against single node failure. It is inspired by the IEEE 802.4[8] Token Bus Protocol, in which each station takes a turn to transmit and release its turn when the reserved time expires.

WTRP guarantees bounded delay and a share of the bandwidth to all the stations in the network. The consistency of token rotation time, regardless of the number of simultaneous transmissions is key to bounding the medium access latency. This perhaps is the most valuable feature of WTRP, since it is critical in real-time applications like the one discussed in this thesis.

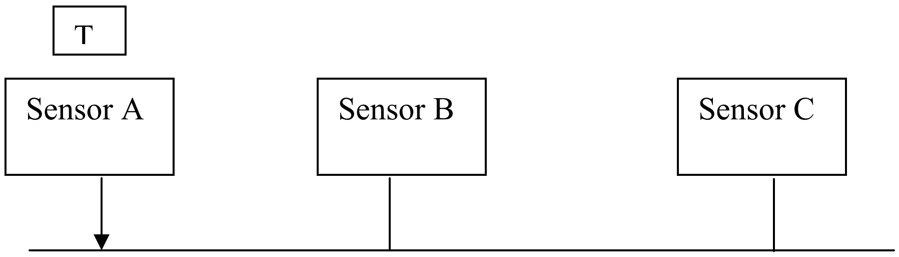
WTRP works in the following manner:



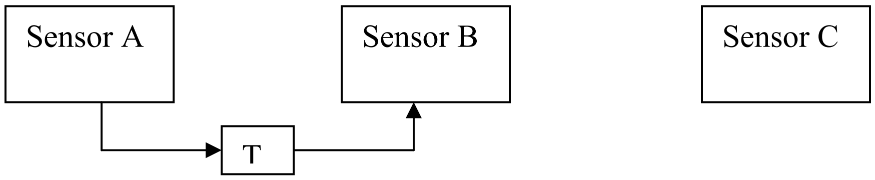
The three sensors A, B and C are in a network, and transmit data to the VMS.



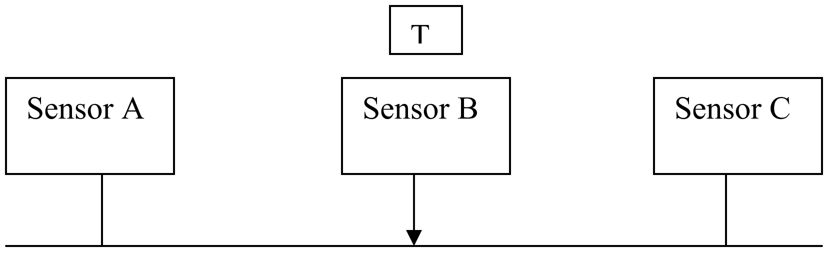
Sensor A has the right to transmit data (has the token).



Only A has the right to transmit data to any point or node in the network.



Sensor A now transmits the token to its successor, Sensor B.



Now, Sensor B has the right to transmit data to any point/sensor/node in the network.

Figure 6.1: Wireless Token Ring Protocol

In the WTRP, the successor and the predecessor fields of each node in the ring define the ring and the transmission order. A station receives the token from its predecessor, transmits data and passes the token to its successor.

Let us now study Wireless Token Ring Protocol in detail. Duke Lee, et al(35) have explained the protocol in details. The following are excerpts from their paper.

Wireless Token Ring Protocol (WTRP) is inspired from the IEEE 802.4 Token Bus protocol. WTRP is a distributed MAC protocol for ad-hoc networks. By distributed, it is meant that there is no central station manning the other stations and as such, each station is independent or partially connected to another station. This makes the protocol quite robust against a single point of failure. WTRP is designed to recover from multiple simultaneous failures (3).

The Token

Let us now look at the critical component of this protocol, and that is the token. Following is a depiction of the token frame.

FC	RA	DA	SA	NoN	GenSeq	Seq

Bytes

1	6	6	6	2	4	4

Figure 6.2: Token Frame and size (3)

In the figure above,

- FC stands for Frame Control and it identifies the packet, such as Token, Solicit Successor, Set Predecssor, etc
- RA stands for Ring Address, and it refers to the ring to which the token belongs
- DA is the Destination Address
- SA is the Source Address
- NoN is the number of nodes in the ring and it is calculated by taking the difference of sequence numbers in one rotation
- GenSeq is the Generation Sequence number. It is initialized to zero and incremented at every rotation of the token by the creator of the token
- Seq is the Sequence Number and it is initialized to zero. It is incremented by every station that passes the token

Problem of Partial Connectivity (3)

One of the biggest challenges that the WTRP overcomes is partial connectivity. To overcome the problem of partial connectivity, management, special tokens and new fields in the tokens are introduced into the protocol. When a node joins a ring, it is mandatory that the joining node be connected to a prospective predecessor node and a successor node.

So how does a node join a ring?

In WTRP, each node has a connectivity manager that monitors the transmissions from its own ring, as well as other rings in the vicinity. The connectivity manager builds a connectivity table, which is nothing but an ordered list of stations in its rings, by monitoring the sequence number of the transmitted tokens, which we will get to in sometime.

Hence, to join a ring, the joining node obtains this information by looking up its connectivity table. When a node leaves a ring, the predecessor of the leaving node finds the next available node to close the ring by looking up its connectivity table.

Here is a diagrammatic representation of the connectivity table and the joining of the ring is shown below:

Connectivity table of station E

Seq = 1	Address = F
Seq = 2	Address = A
Seq = 3	Address = unknown
Seq = 4	Address = unknown
Seq = 5	Address = D

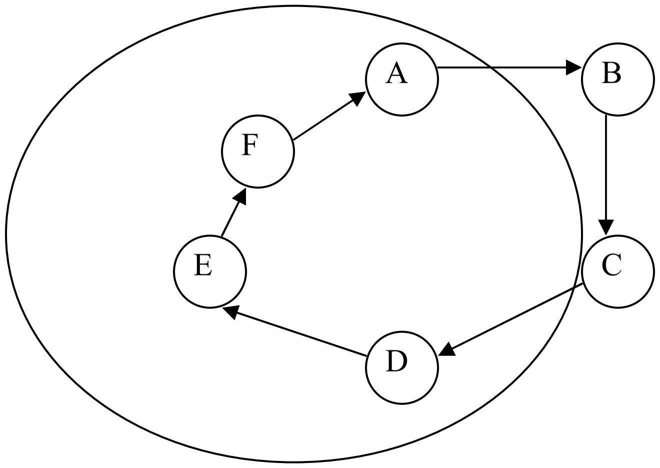


Figure 6.3: Connectivity Table And Node Joining The Ring (3)

In the figure, Station E monitors the successive token transmission from F to D before the token comes back to E. At time 0, E transmits the token with sequence number 0, at time 1, F transmits the token with sequence number 1 and so on. E will not hear the transmission from B and C, but when it hears the transmission from D, E will notice that the sequence number has been increased by 3 instead of 1. This indicates to E that there were two stations it could not hear between A and D

In a partially connected network, simply dropping the token whenever a station hears another transmission is not sufficient. A unique priority assignment scheme for tokens has been developed that will delete tokens that the station is unable to hear. WTRP also has algorithms that keep each ring address unique, to enable the operation of multiple rings in proximity (3).

Architecture of WTRP (35)

The WTRP was designed to be deployed in the automated highway project of the University of California at Berkeley, wherein, a platoon of vehicles traveling along a highway communicate with each other, through the sensors located on them. As a result, the architecture of WTRP is more attuned to take care of some issues evolving out of the automated highway project.

The figure below describes the overall system architecture. In addition to the connectivity table, there is a Mobility Manager, Channel Allocator, Management Information Base (MIB), and Admission Control Manager. It is assumed that multiple channels are available, and that different rings are on different channels.

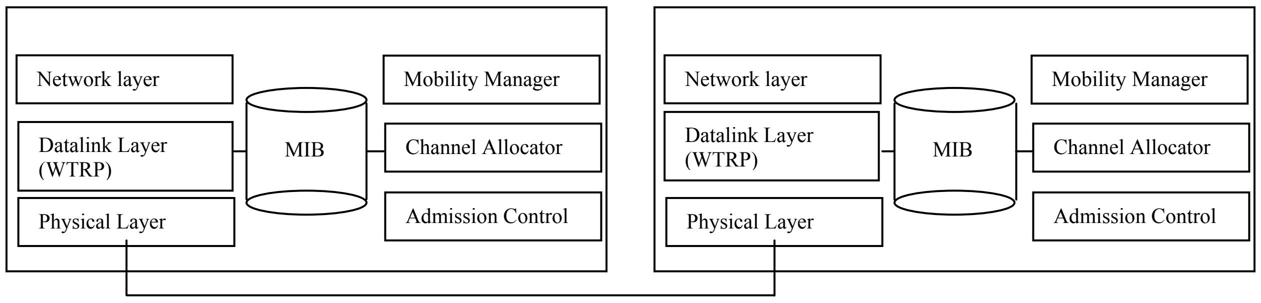


Figure 6.4: Architecture of WTRP (35)

Medium Access Control

Medium Access Control (MAC) enables multiple nodes to transmit on the same medium. This is where WTRP is located. The main function of MAC is to control the timing of the transmissions by different nodes to increase the chance of successful transmission.

In WTRP, the MAC layer performs ring management and timing of the transmissions. Ring Management involves:

- Ensuring that each ring has a unique ring address
- Ensuring that one and only one token exists in a ring
- Ensuring the propriety of the ring
- Managing the joining and leaving operations

Mobility Manager

The Mobility Manager decides when a station should join or leave a ring. The problem that the Mobility Manager has to solve is similar to the mobile hand-off problem. When a mobile node is drifting from a ring and into the vicinity of another ring, at some threshold the Mobility Manager decides to move to the next ring.

Admission Control

To ensure the level of Quality of Service (QoS), the Admission Control manager limits the number of stations that can transmit on the medium. There is an Admission Control Manager in each ring. It may move with the token but does not have to move every time the token moves. It periodically solicits other stations to join if there are resources available in the ring. To define a resource, let

MAX_MTRT be the minimum of the maximum latency that each station in the ring can tolerate.

RESV_MTRT be the sum of the token holding time (THT) of each station.

MAX_NoN be the maximum number of nodes (NoN) that are allowed in the ring.

The Admission Control Manager has to ensure that

RESV_MTRT < MAX_MTRT

And

NoN < MAX_NoN

Only if these are satisfied, the Admission Control Manager solicits another station to join.

Management Information Base (MIB)

The Management Information Base holds all the information that each management module needs to manage the MAC module. Majority of this information is collected by the MAC module and stored there.

Channel Allocator

Generally, the Channel Allocator chooses the channel on which the station should transmit. If a large number of token rings exist in proximity, achieving spatial reuse through sensible channel allocation can increase their efficiency.

Finding the globally optimal solution for channel allocation, an allocation that maximizes the capacity of the network, is a challenging problem in any large deployment of many mobile nodes. This is due to

1. Collecting and maintaining channel allocation information can be difficult and burdensome
   This is because the collection and maintenance of information may involve frequent packet transmission
2. The optimal allocation computation is complex

In this project, a distributed solution is considered. The Channel Allocator is local to each station, and the channel allocator can access the network topology through the MIB. Each node decides on which channel to join in a distributed manner using the information collected from the token structure, which contains the number of nodes (NoN) in the ring. If NoN reaches the maximum value, this is an indication for the nodes out of the ring to shift to the next channel and search for another ring.

Acknowledgement of token transmissions

The Ring Owner is the station that has the same MAC address as the ring address. A station can claim to be the ring owner by changing the ring address of the token that is being passed around. Stations rely on implicit acknowledgments to monitor the success of their token transmissions. An implicit acknowledgement is any packet heard after token transmission that has the same ring address as the station. Another acceptable implicit acknowledgement is any transmission from a successive node regardless of the ring address in the transmission. A successive node is a station that was in the ring during the last token rotation, i.e. the succession stations are those present in the local connectivity table.

Each station resets its IDLE_TIMER whenever it receives an implicit acknowledgement. If the token is lost in the ring, then no implicit acknowledgement will be heard in the ring, and the IDLE_TIMER will expire. When the IDLE_TIMER expired, the station generates a new token, thereby becoming the owner of the ring.

Multiple tokens

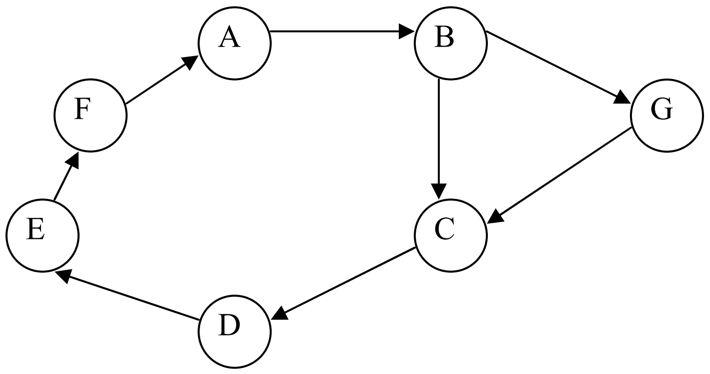
To resolve multiple tokens (that is, to delete all tokens but one), the concept of priority is used. The generation sequence number and the ring address define the priority of a token. A token with a higher generation sequence number has higher priority. When the generation sequence numbers of tokens are the same, ring addresses of each token are used to break the tie. The priority of a station is the priority of the token that the station accepted or generated. When a station receives a token with a lower priority than itself, it deletes the token and notifies its predecessor without accepting a token. With this scheme, the protocol deletes all multiple tokens in a single token rotation provided no more tokens are being generated.

Ring recovery Mechanism

The ring recovery mechanism is invoked when the monitoring node decides that its successor is unreachable. In this case, the station tries to recover from the failure by reforming the ring. The strategy taken by the WTRP is to try and reform the ring by excluding as small a number of nodes as possible. Using the Connectivity Manager, the monitoring station is able to quickly find the next connected node in the transmission order. The monitoring station then sends the SET_PREDECESSOR token to the next connected node to close the ring.

Joining the ring (35)

WTRP allows nodes to join a ring dynamically, one at a time, if the token rotation time (sum of token holding times per node, plus overhead such as token transmission times) would not grow unacceptably with the addition of the new node.



Joining G in the ring

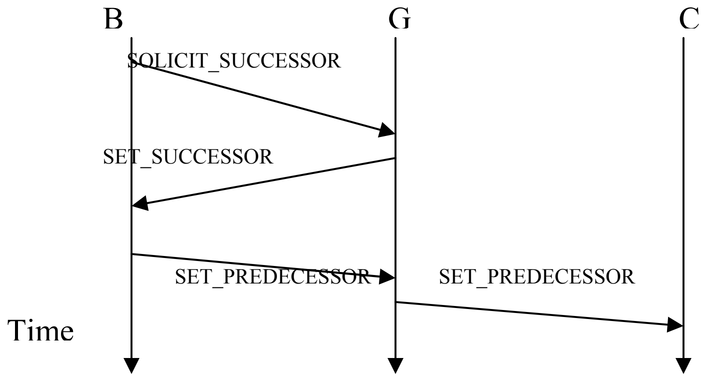
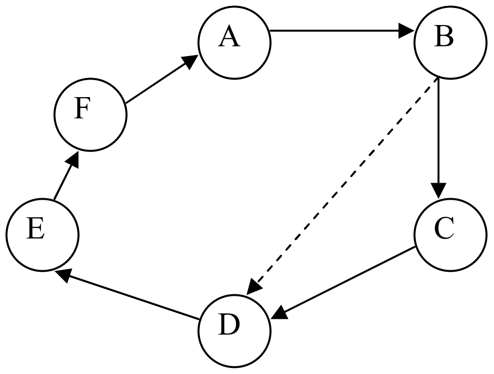


Figure 6.5: Joining The Ring by Solicit Successor and Set Predecessor (35)

In the above figure, suppose station G wants to join the ring. Let us also say that the Admission Control Manager on station B invites another node to join the ring by sending out a SOLICIT_SUCCESSOR. The Admission Control Manager waits for the duration of the response window for the interested nodes to respond. The response window represents the window of opportunity for a new node to join the ring. The response window is divided into slots of the duration of the SET_SUCCESSOR transmission time. When a node that wants to join the ring, such as G in the above example, hears a SOLICIT_SUCCESSOR token, it picks a random slot and transmits a SET_SUCCESSOR token. When the response window passes, the host node, B can decide among the slot winners. Suppose that G wins the contention, then the host node passes the SET_PREDECESSOR token to G, and G sends the SET_PREDECESSOR token to node C, the successor of the host node B. The joining process concludes.

Leaving the ring



C leaving the ring

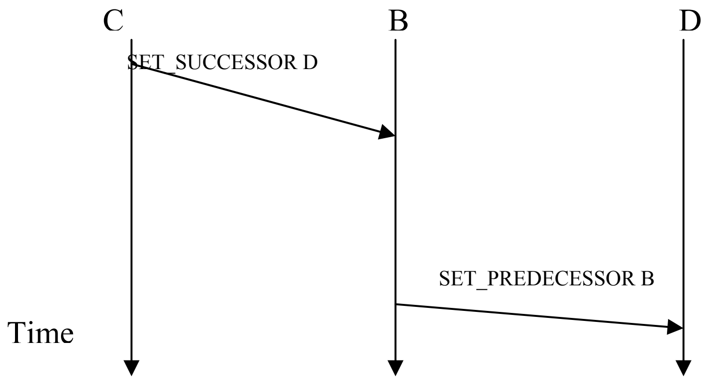


Figure 6.6: Leaving The Ring by Set Successor and Set Predecessor (3)

As shown in the above figure, suppose station C wants to leave the ring. First, C waits for the right to transmit. Upon receipt of the right to transmit, C sends the SET_SUCCESSOR packet to its predecessor B with the MAC address of its successor, D.

If B can hear D, B tries to connect with D by sending a SET_PREDECESSOR token. If B cannot hear D, B will find the next connected node, in the transmission order, and send it the SET_PREDECESSOR token.

Interference (35)

Interference is eliminated by including NoN (Number of Nodes) in the token packet. When a station detects a ring, it examines the NoN value in the token. If NoN is set to maximum, the station changes its channel and searches for another ring. Otherwise, the station either waits to become a ring member or changes its channel to search for another ring. If the station waits, it suspends transmission and waits for a SOLICIT_SUCCESSOR token. As a result, a newcomer station never interferes with the ring.

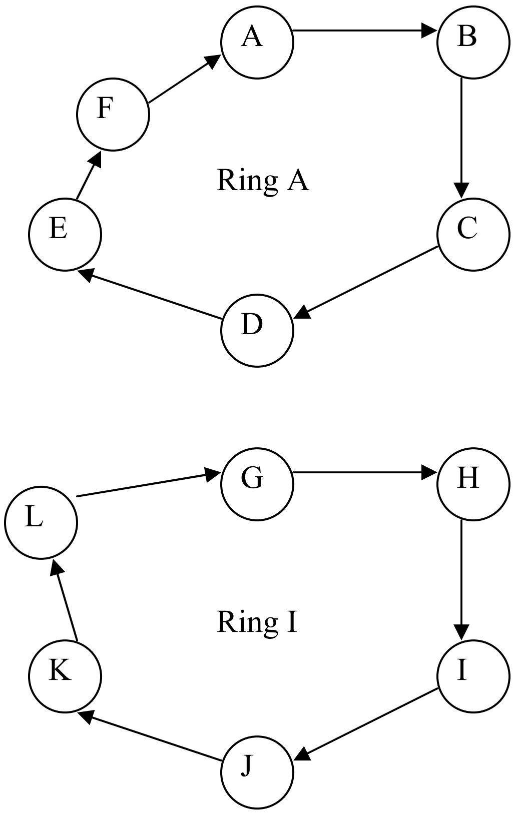


Figure 6.7: Multiple Rings (35)

In the above figure, we can see that the ring address of a ring is the address of one of the stations in the ring, which is called the owner of the ring. In the example, the owner of ring A is station A. Because it is assumed that the MAC address of each station is unique, the ring address is also unique. The uniqueness of the address is important, since it allows the stations to distinguish between messages coming from different rings.

To ensure that the ring owner is present in the ring, when the ring owner leaves the ring, the successor of the owner claims the ring address and becomes the ring owner. The protocol deals with the case where the ring owner leaves the ring without notifying the rest of the stations in the ring as follows. The ring owner updates the generation sequence number of the token every time it receives a valid token. If a station receives a token without its generation sequence number updated, it assumes that the ring owner is unreachable and it elects itself to be the ring owner.

Thus, Wireless Token Ring Protocol manages the medium by creating multiple rings. Rings are identified by an ID that is the MAC address of the node in the ring. Each station has a successor and a predecessor. When a station receives the token from the predecessor, it transmits for a fixed time and passes the token to its successor. After transmitting, the station looks for an implicit acknowledgement that is a transition from its successor to the successor of its successor. Stations monitor token transitions and create a connectivity table (an ordered list of stations in their ring). A token has a priority that increases each time it is transmitted. If a station gets two different tokens, it chooses the one with the higher priority and notifies its predecessor.

Each station monitors the token rotation time and the number of nodes in the ring, and sends the invitation to the nodes outside the ring, if there is room. The joining process is a handshake mechanism between the inviting station, joining station and the successor of the inviting station. When a station leaves the ring willingly, it notifies its successor to its predecessor. The predecessor tries to connect to the successor of the station, and if it is unsuccessful, the predecessor closes the ring with the next station in its connectivity table.

If there is a failure in a station, its predecessor detects it when passing the token, and tries to connect with the next station in its connectivity table.

Let us see how WTRP can be utilized in our problem. As per our problem, we have sensors placed along a highway work zone or for that matter, any part of the freeway, as part of the ITS initiative to provide real-time information to users. The figure below gives a visual description of the problem.

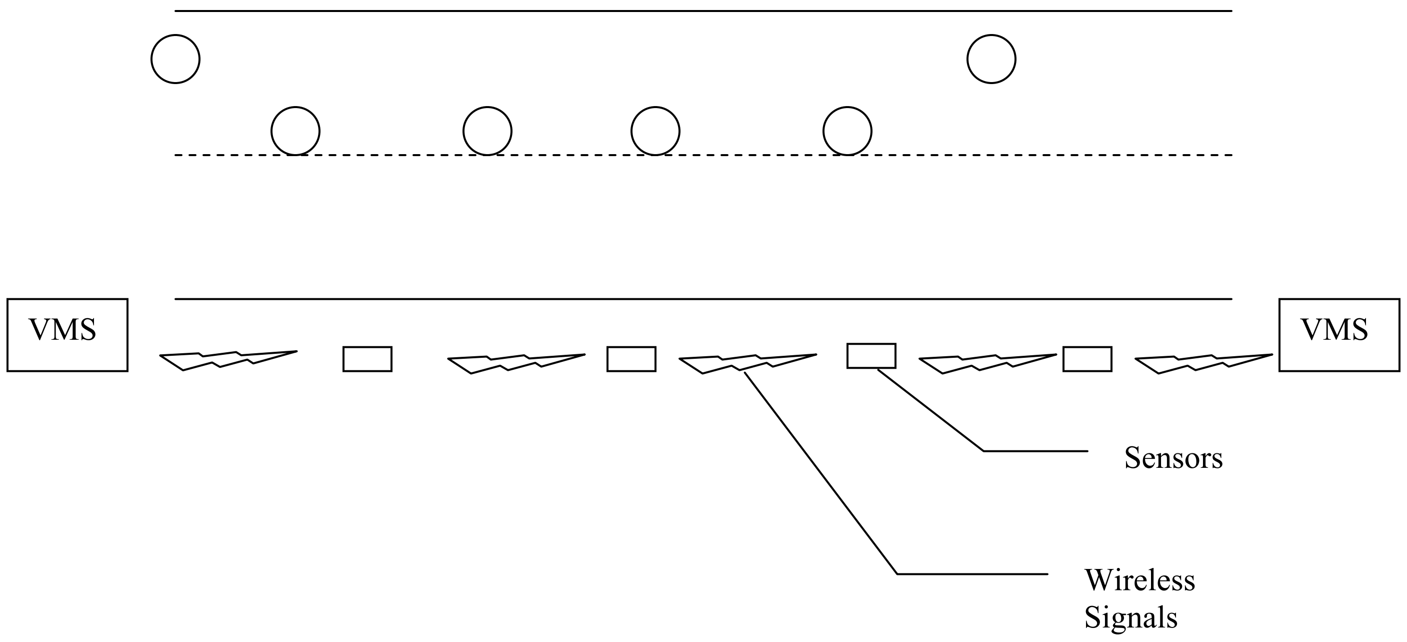


Figure 6.8: Highway Work Zone

Let the sensors, the VMS and the computer be the nodes in the ring, as per the Wireless Token Ring Protocol.

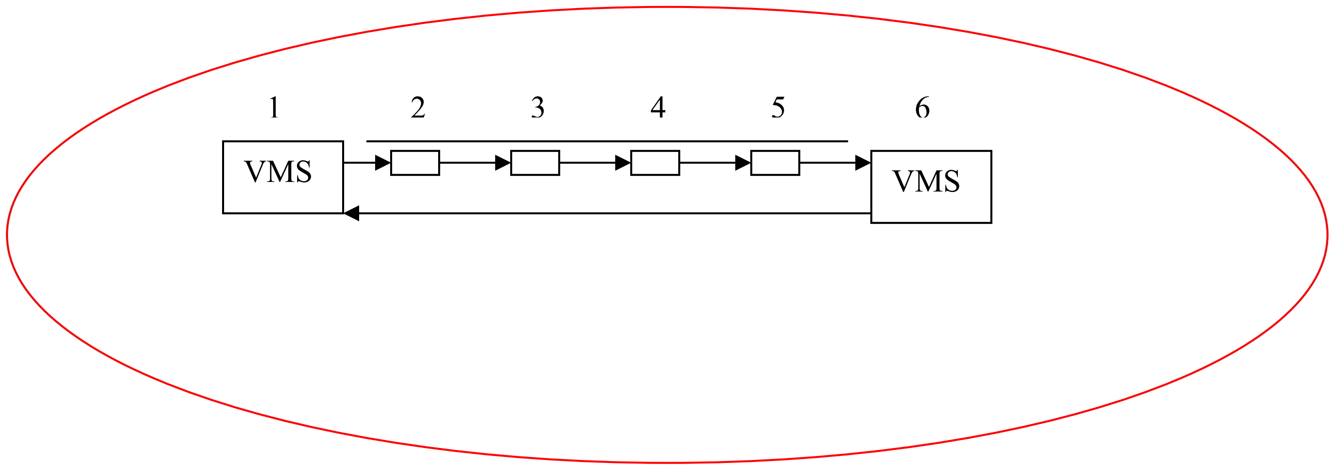


Figure 6.9: Wireless Token Ring Adaptation of the Highway Work Zone

Let the VMS, sensors and the computer be labeled Station 1, Station 2, etc.

Connectivity table for Station 1

Seq = 1	Address = 2
Seq = 2	Address = 3
Seq = 3	Address = 4
Seq = 4	Address = 5
Seq = 5	Address = 6

In the figure, Station 1 monitors the successive token transmission from Station 2 to Station 6 before the token comes back to Station 1. At time 0, Station 1 transmits the token with sequence number 0, at time 1, Station 2 transmits the token with sequence number 1 and so on. Station 1 will not hear the transmission from Station 3 to Station 5, but when it hears the transmission from Station 6, Station 1 will notice that the sequence number has been increased by 3 instead of 1. This indicates to Station 1 that there were three stations it could not hear between Station 2 and Station 6.

Thus, it can be seen that the problem of data collision in sensor networks on the freeway can be minimized using Wireless Token Ring Protocol or WTRP, discussed above. If needed, more nodes or stations can be added to this ring, as per the mechanism discussed above. The problem of multiple tokens is also resolved using the concept of priority. This protocol recovers quickly from a single node failure by quickly reforming the ring, as discussed above. The Connectivity Manager provides the ability to the monitoring station to quickly find the next node in the transmission order.

Chapter 7

Communication Range for Protocol

On the field, a wide range of problems can crop up. One of the most common problems on the freeway is the lack of Line of Sight for the sensor nodes. Range of Communication is a key factor in the wireless world. As radio waves propagate in free space, power falls off as a square of the range.

This effect of the spreading of the wave propagated is calculated using the Free Space Path Loss Propagation equation.

PL (Path Loss) = $20 \times \log_{10}(4\times\prod\times d\times f / c)$

Where,

d =distance between the transmitter and the receiver

f = frequency of operation = 460MHz

c = speed of light in m/sec= $3\text{x}10^8$

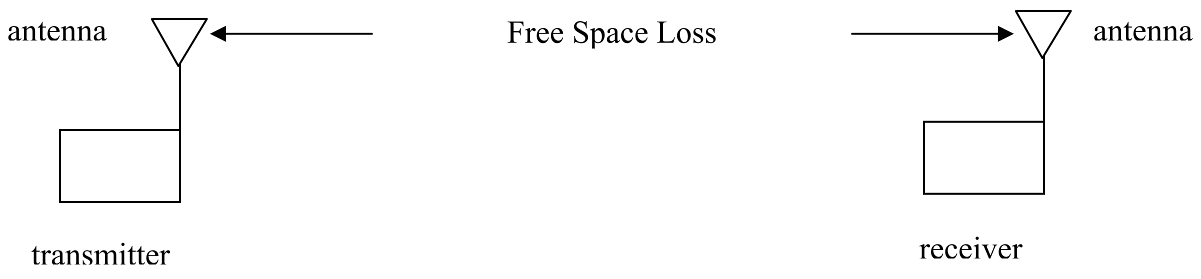


Figure 7.1: Free Space Path Loss

Sensors on the field usually have a frequency of operation of 460MHz. We assume this to be the frequency for our calculations.

Assuming 10dBi antenna gains at the Transmitter and the Receptor with 0 dB cable loss.

Sensitivity at the receiver = -80 dBm and the transmitter and the receiver are placed 2 Km or 1.2 miles apart

Plugging these in the Path Loss Equation mentioned above gives us

PL = -92 dB

The Effected (isotropic) Radiated Power (EiRP) = 30dBm

Hence, the Receiver level = 30dBM – 92dB + 10dBi = -52 dBm

Link Margin = Receiver level – Sensitivity

= -52 dBm – (-80dBm)

= 28 dB

Hence, we have a 28dB link margin that gives us the cushion due to fading conditions and interference.

Thus, we have a high quality RF link because of such a high link margin.

As stated in the details for the Wireless Token Ring Protocol, the protocol allows the nodes to transmit, one at a time, by passing the transmission token, one at a time. Hence, no two nodes talk at the same time. This allows the transmitters and the receivers to use only one channel in the 460MHz band. These transmitters and receivers can then re-use the frequencies after every transmission and reception. Hence, when only one channel is available, then the frequencies can be re-used. Based on the calculations for the Free Space Path Loss Equation, we come to the conclusion that the maximum range for the sensors in a single channel bandwidth will be 10Km or 6.25 miles. At this distance, the path loss is 105dB and that takes the received level to reach –79.8dBm. Beyond this distance, the signal will surely drop.

So this brings us to a new problem, and that is, on the field, the sensors might not be in a ring and might be placed linearly, alongside the freeway.

Let us look at our scenario.

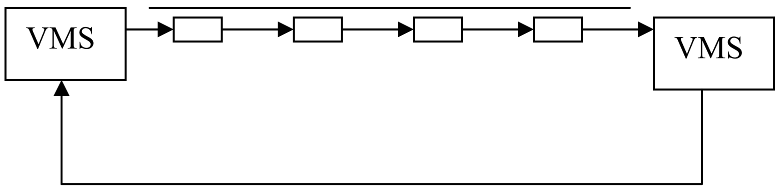


Figure 7.2: Alignment of Sensors on field

According to our calculations, the maximum distance, at which the transmission will definitely drop is 10KM or 6.2 miles. Hence if the last node in the network wants to send information to the first node, the distance from the first node to the last node is enormous, and there is a likelihood of the transmission being dropped. As a result, this protocol needs to be modified so that the data transmitted from the last node to the first node does not get lost.

Chapter 8

Modification to the Wireless Token Ring Protocol

As per the wireless token ring protocol, the nodes pass the token ring to the successor nodes after a specified time interval, in a ring. Mustafa Ergen (5) has suggested an extension called 'Token Chain", where he suggests that the ring be bi-directional as opposed to uni-directional.

To extend the reach of the nodes, the protocol can be modified in the following way.

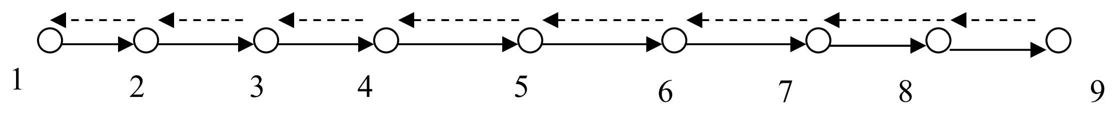


Figure 8.1: Modification of Wireless Token Ring Protocol (35)

As is evident from the figure above, the protocol is modified to be bi-directional, instead of unidirectional. This enables us to extend the reach of the protocol far more than the traditional Wireless Token Ring Protocol, which is unidirectional. So instead of a long hop of the token from node 9 to node 1, the token is simply passed from node 9 to node 8, and reversed back to node 1.

This is achieved by modifying the protocol in the following manner.

In the original protocol, each node maintains a connectivity table that contains the predecessor and the successor of the node. Another connectivity table is created for each node, in which, the successor and the predecessor nodes are reversed. Hence, the predecessor node in the original connectivity table becomes the successor node in the new connectivity table. (35)

Mustafa Ergen (5) has also proposed this as the best approach to make the token bi-directional. This bi-direction flow of the table is achieved in the following manner.

As mentioned above, each node has a connectivity table that provides it with information about adjoining nodes. When the node receives the token, it transmits the data, and looks up the connectivity table for the next node in the ring, and passes the token to the next node. Thus, the connectivity table is key to this protocol. We make use of this connectivity table to achieve bi-directional flow of token. We also make use of the token frame, which consists of information regarding the Ring Address, the Source Address, the Destination Address and so on.

Initially, the token is forward moving, as per the original protocol. The token passing and acknowledgement occurs as per WTRP. For the final node, the direction is always forward moving, and as such, the Source Address (SA) and the Destination Address (DA) is the same for it. Hence, when it transmits the data, it “forwards” the token to the Destination Address (DA) (Same as the Source Address (SA)). (35)

The middle nodes, on receiving the token, compare the token frame with the connectivity table. If the Source Address (SA) on the token frame is the same as the Predecessor node in the connectivity table, then the node realizes that the token is forward moving, and passes the token to the Successor node.

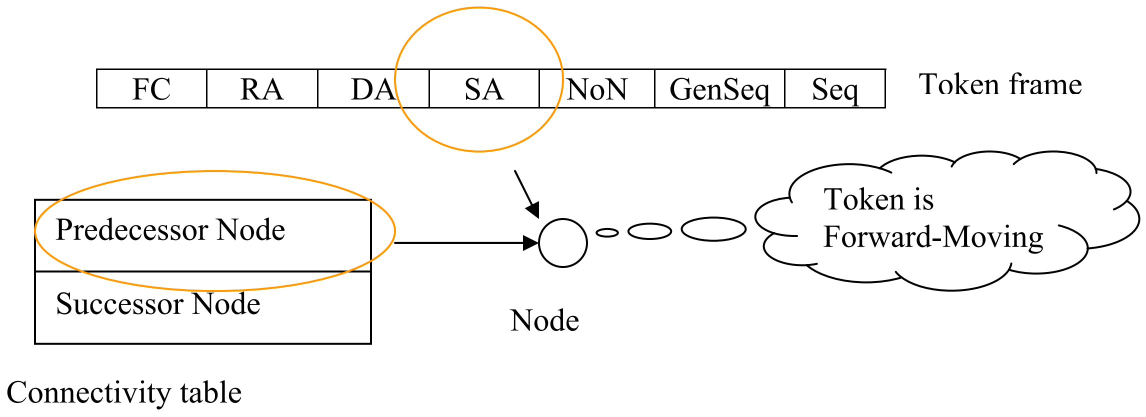


Figure 8.2: Token is Forward-Moving

If the Source Address (SA) on the token frame is the same as the Successor node, then the node realizes it is a backward moving token, and hence passes the token to the predecessor.

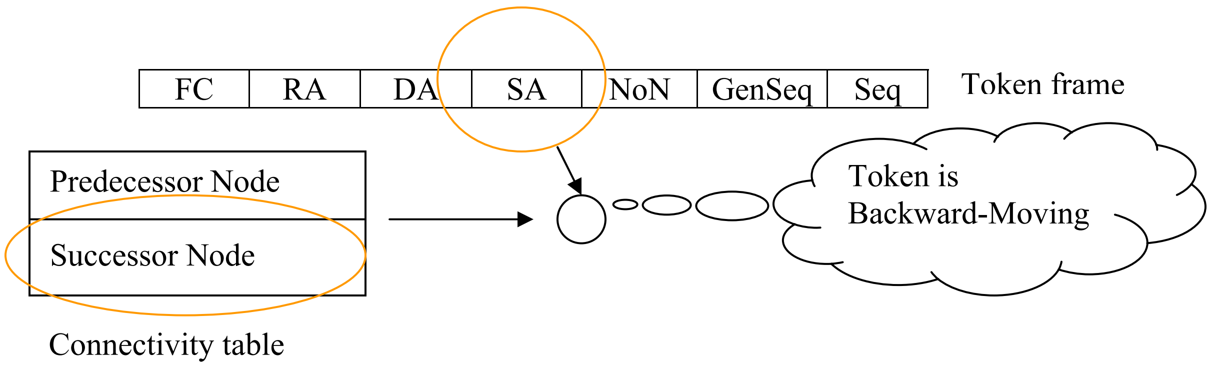


Figure 8.3: Token is Backward-Moving

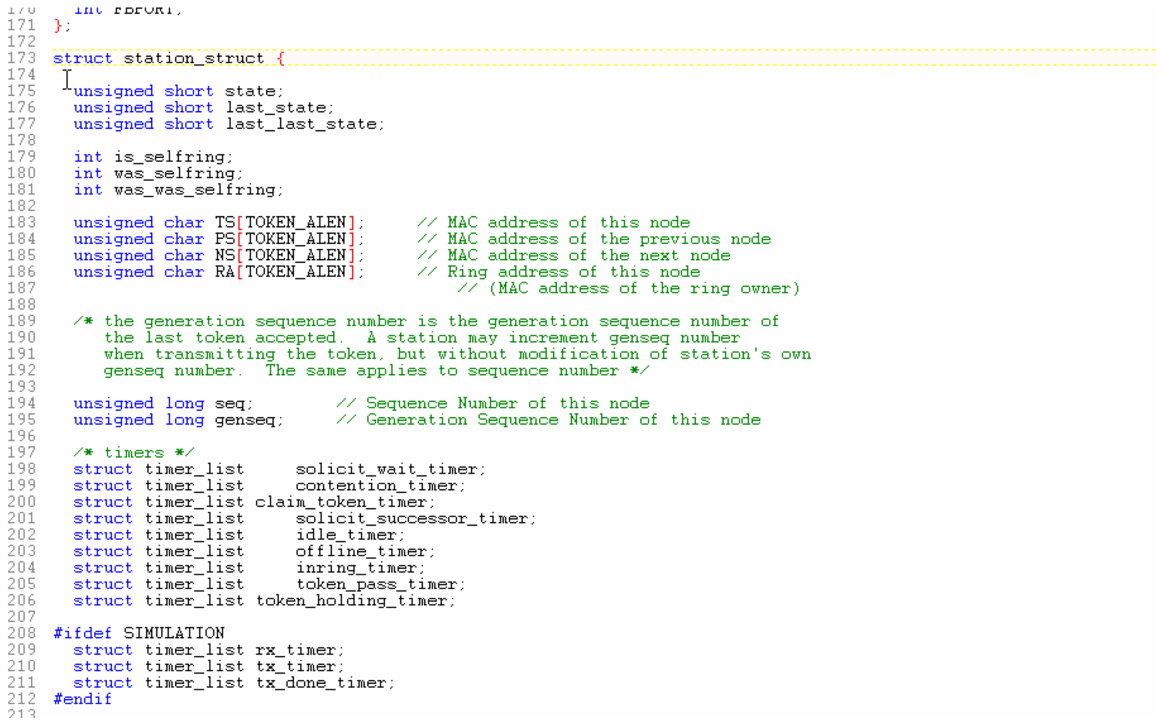
The WTRP source code has a lot of files containing code that can get very complicated. Some of the files that needed to be modified in order to achieve the modification in the protocol are:

Tokenring.h: (this is the header file of the most important file *tokenring.c* in the protocol code)

Tokenring.h contains some of the data structures required for the protocol.

The structure ***station_struct*** is the main structure that drives the protocol.

Below is a snapshot of the structure *station_struct* in the original code.

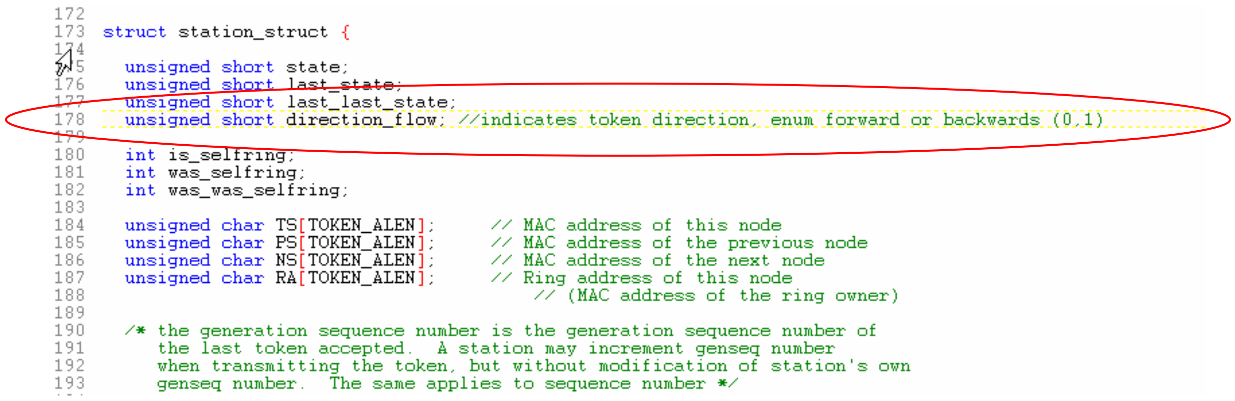


```
};

struct station_struct {

  unsigned short state;
  unsigned short last_state;
  unsigned short last_last_state;

  int is_selfring;
  int was_selfring;
  int was_was_selfring;

  unsigned char TS[TOKEN_ALEN];     // MAC address of this node
  unsigned char PS[TOKEN_ALEN];     // MAC address of the previous node
  unsigned char NS[TOKEN_ALEN];     // MAC address of the next node
  unsigned char RA[TOKEN_ALEN];     // Ring address of this node
                                        // (MAC address of the ring owner)

  /* the generation sequence number is the generation sequence number of
     the last token accepted.  A station may increment genseq number
     when transmitting the token, but without modification of station's own
     genseq number.  The same applies to sequence number */

  unsigned long seq;        // Sequence Number of this node
  unsigned long genseq;     // Generation Sequence Number of this node

  /* timers */
  struct timer_list     solicit_wait_timer;
  struct timer_list     contention_timer;
  struct timer_list claim_token_timer;
  struct timer_list     solicit_successor_timer;
  struct timer_list     idle_timer;
  struct timer_list     offline_timer;
  struct timer_list     inring_timer;
  struct timer_list     token_pass_timer;
  struct timer_list token_holding_timer;

#ifdef SIMULATION
  struct timer_list rx_timer;
  struct timer_list tx_timer;
  struct timer_list tx_done_timer;
#endif
```

Figure 8.4: Structure *station_struct* in the tokenring.h file

Currently, the protocol is unidirectional. To make it bi-directional, we need to create a variable in structure station_struct

A variable *direction_flow* is created which indicates whether the token is taking the forward or the backward direction

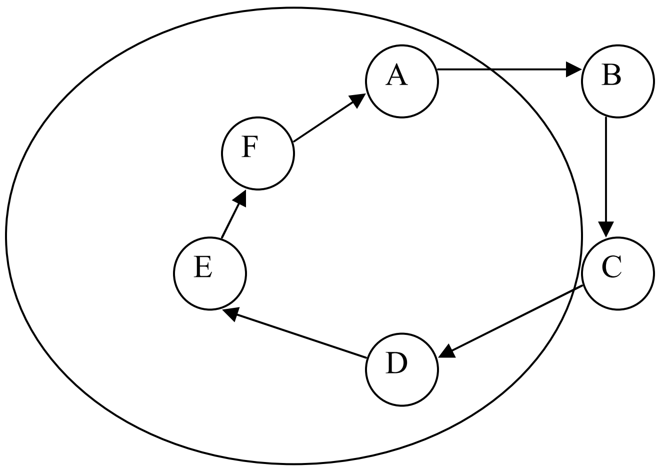


```
struct station_struct {

  unsigned short state;
  unsigned short last_state;
  unsigned short last_last_state;
  unsigned short direction_flow; //indicates token direction, enum forward or backwards (0,1)

  int is_selfring;
  int was_selfring;
  int was_was_selfring;

  unsigned char TS[TOKEN_ALEN];     // MAC address of this node
  unsigned char PS[TOKEN_ALEN];     // MAC address of the previous node
  unsigned char NS[TOKEN_ALEN];     // MAC address of the next node
  unsigned char RA[TOKEN_ALEN];     // Ring address of this node
                                        // (MAC address of the ring owner)

  /* the generation sequence number is the generation sequence number of
     the last token accepted.  A station may increment genseq number
     when transmitting the token, but without modification of station's own
     genseq number.  The same applies to sequence number */
```

Figure 8.5: Modification to structure *station_struct* in the tokenring.h file

The Connectivity Table

As mentioned in the protocol details, each station maintains a connectivity table that holds information of the transmission order of its own ring. The connectivity tables reside in the file *ctable.c*. The following figure will explain the connectivity table formation for station E.



Reception Range of E

Seq = 1	Address = F
Seq = 2	Address = A
Seq = 3	Address = unknown
Seq = 4	Address = unknown
Seq = 5	Address = D

Figure 8.6: Connectivity Table

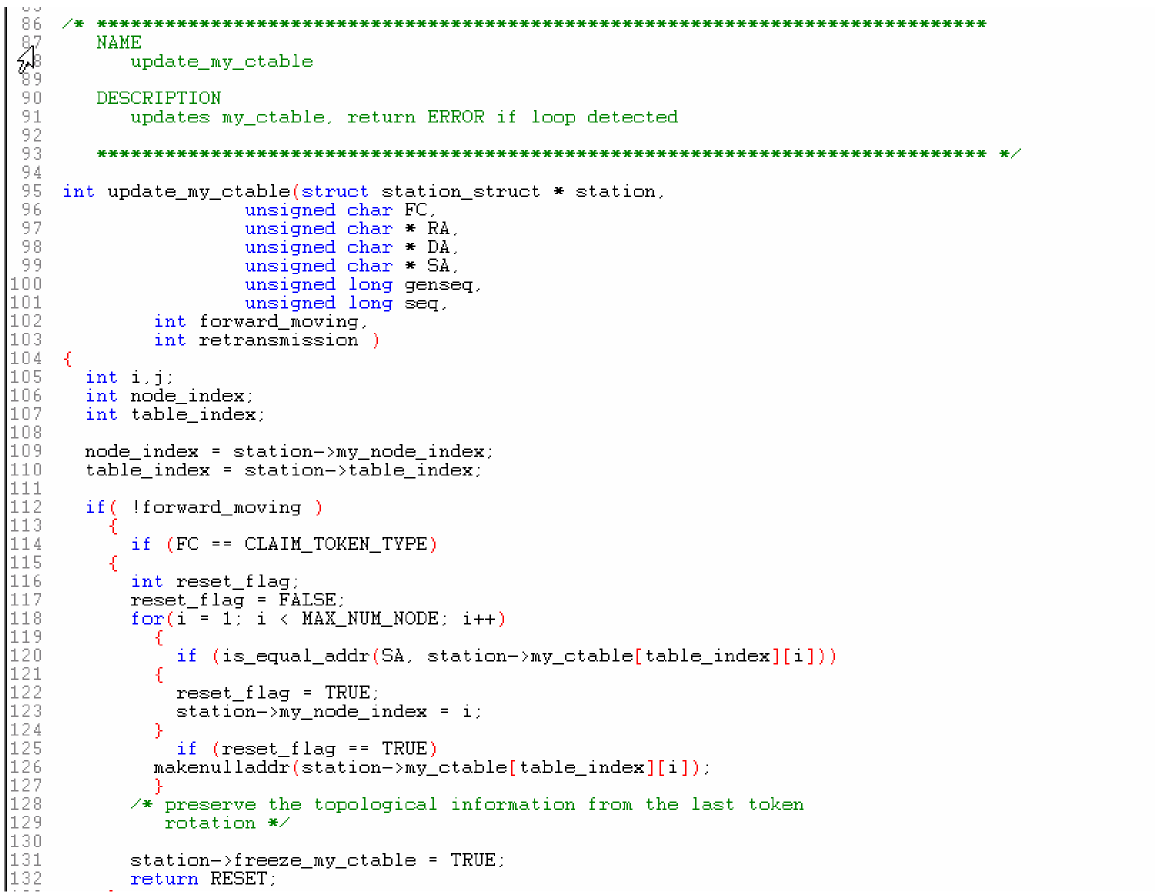
The file *ctable.c* consists of information and code for the connectivity tables.

The connectivity table *my_ctable* contains the information about the transmission order of its own ring. The other table *other_ctable* holds information about all the transmissions in the reception range.

The WTRP has the provision of freezing, or storing the connectivity table. But it can refer only to the last frozen table for history. The table in history helps populate the new table.

The *my_ctable* table is where some of the modifications were carried out.

The function *update_my_ctable* updates the connectivity table *my_ctable*



```c
/* ******************************************************************************
   NAME
      update_my_ctable

   DESCRIPTION
      updates my_ctable, return ERROR if loop detected

   ****************************************************************************** */

int update_my_ctable(struct station_struct * station,
                unsigned char FC,
                unsigned char * RA,
                unsigned char * DA,
                unsigned char * SA,
                unsigned long genseq,
                unsigned long seq,
        int forward_moving,
        int retransmission )
{
  int i,j;
  int node_index;
  int table_index;

  node_index = station->my_node_index;
  table_index = station->table_index;

  if( !forward_moving )
    {
      if (FC == CLAIM_TOKEN_TYPE)
    {
      int reset_flag;
      reset_flag = FALSE;
      for(i = 1; i < MAX_NUM_NODE; i++)
        {
          if (is_equal_addr(SA, station->my_ctable[table_index][i]))
        {
          reset_flag = TRUE;
          station->my_node_index = i;
        }
          if (reset_flag == TRUE)
        makenulladdr(station->my_ctable[table_index][i]);
        }
      /* preserve the topological information from the last token
         rotation */

      station->freeze_my_ctable = TRUE;
      return RESET;
```

Figure 8.7: Function *update_my_ctable* in ctable.c file

The variable *forward_moving* is used as a Boolean, with 0 meaning Reset and any other value meaning forward moving. This parameter need to be changed to be 0 for reset, 1 for forward moving and 2 for backwards moving. Below are some examples of the code changes to this 100+ line function

1. The variable *forward_moving* is changed from a Boolean to one where 0 is RESET, 1 is FORWARD MOVING and 2 is BACKWARDS MOVING

2. *forward_moving* would then become unintuitive. Hence changing the variable to *direction_moving*
3. The Source Address (SA) and the Destination Address (DA) get reversed when the *direction_moving* gets reversed.

Consider the following example:

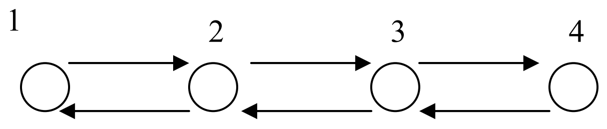
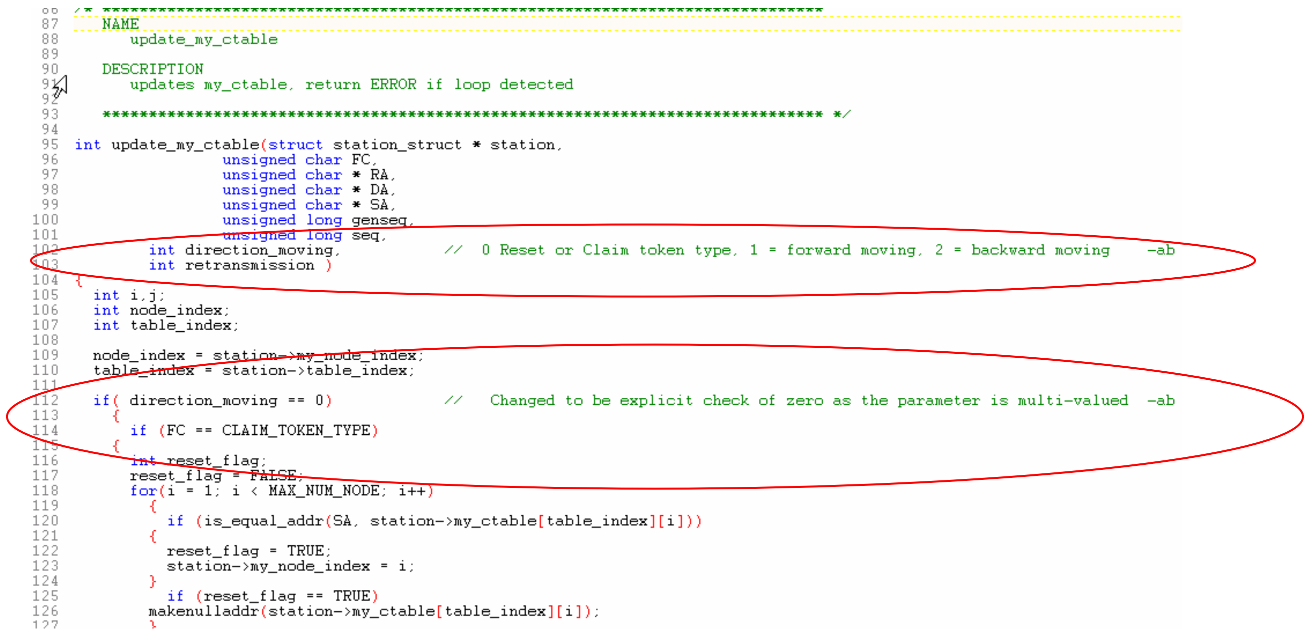
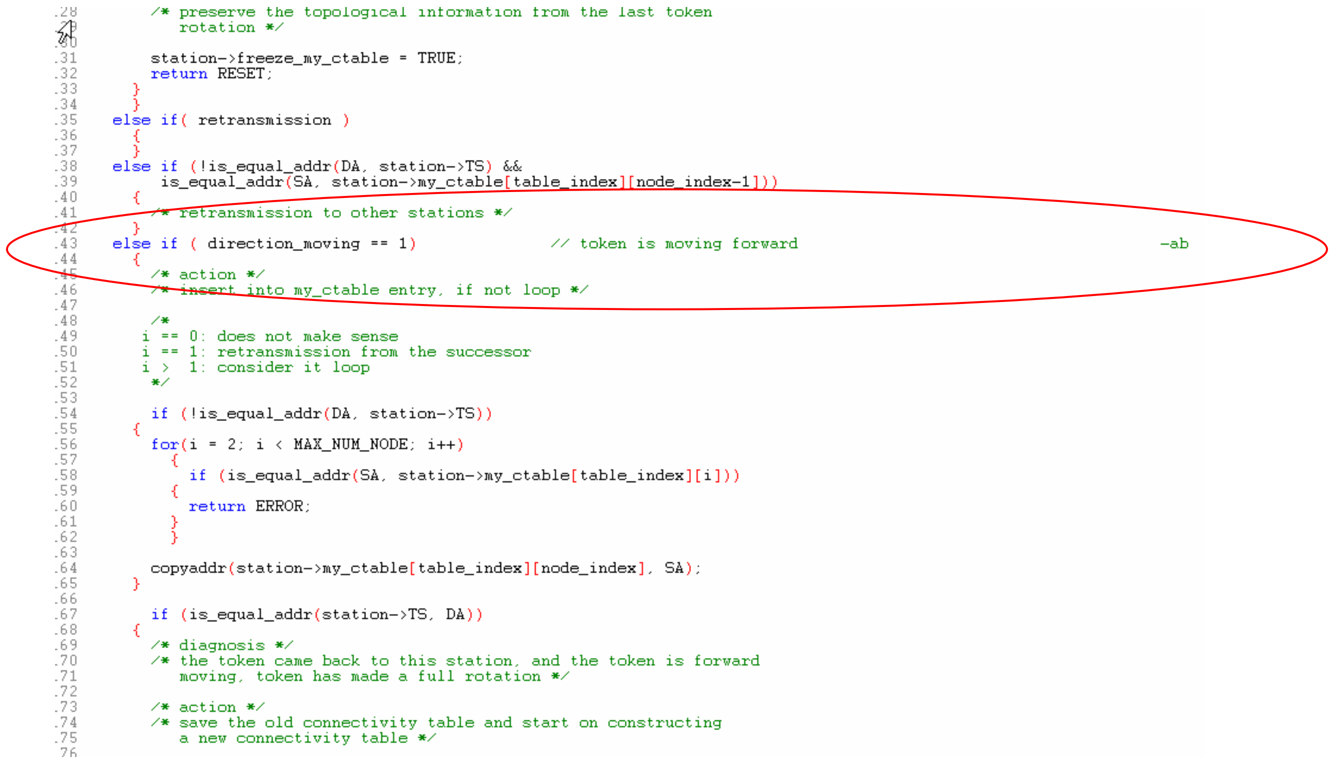


Figure 8.8: Swapping of Source Address and Destination Address on the Reverse Path

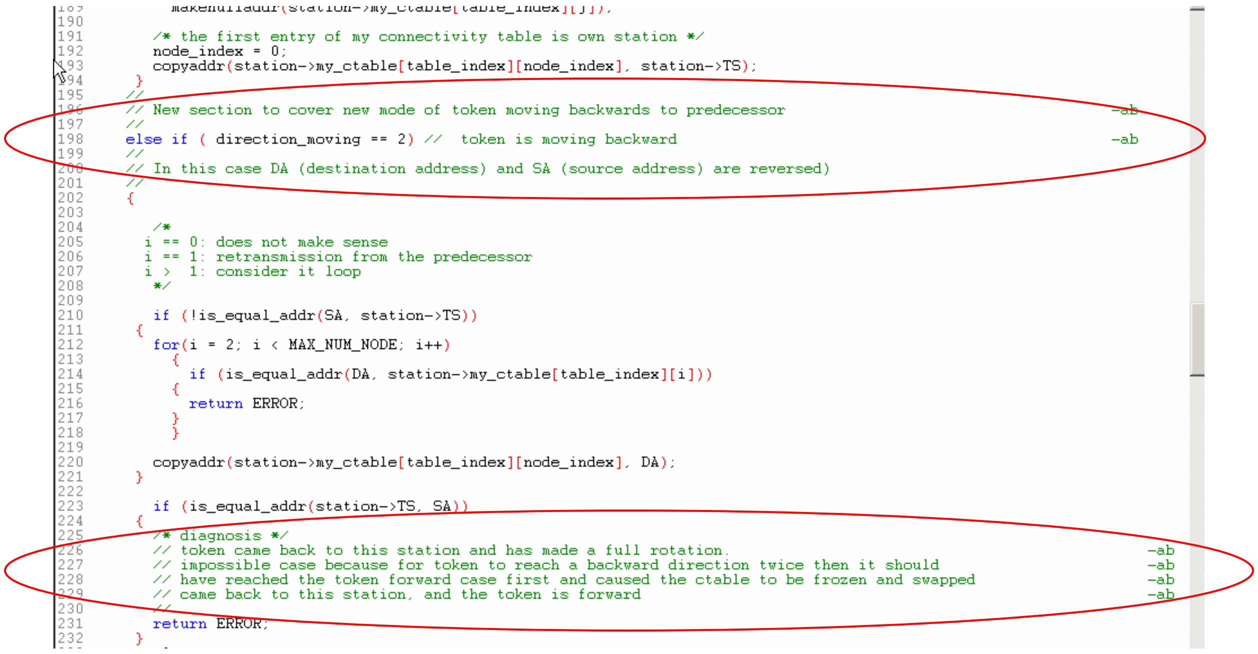
Consider station 3. For 3, the Source Address (SA) for the token in the forward direction is 2 and the Destination Address (DA) is 4. On the reverse direction, for the same node 3, the Source Address (SA) is 4 and the Destination Address (DA) is 2.



```
/* ******************************************************************************
   NAME
      update_my_ctable

   DESCRIPTION
      updates my_ctable, return ERROR if loop detected

   ****************************************************************************** */

int update_my_ctable(struct station_struct * station,
                unsigned char FC,
                unsigned char * RA,
                unsigned char * DA,
                unsigned char * SA,
                unsigned long genseq,
                unsigned long seq,
        int direction_moving,           //  0 Reset or Claim token type, 1 = forward moving, 2 = backward moving    -ab
        int retransmission )
{
  int i,j;
  int node_index;
  int table_index;

  node_index = station->my_node_index;
  table_index = station->table_index;

  if( direction_moving == 0)            //   Changed to be explicit check of zero as the parameter is multi-valued  -ab
    {
      if (FC == CLAIM_TOKEN_TYPE)
    {
      int reset_flag;
      reset_flag = FALSE;
      for(i = 1; i < MAX_NUM_NODE; i++)
        {
          if (is_equal_addr(SA, station->my_ctable[table_index][i]))
        {
          reset_flag = TRUE;
          station->my_node_index = i;
        }
          if (reset_flag == TRUE)
        makenulladdr(station->my_ctable[table_index][i]);
```



```
      /* preserve the topological information from the last token
         rotation */

      station->freeze_my_ctable = TRUE;
      return RESET;
    }
    }
  else if( retransmission )
    {
    }
  else if (!is_equal_addr(DA, station->TS) &&
       is_equal_addr(SA, station->my_ctable[table_index][node_index-1]))
    {
      /* retransmission to other stations */
    }
  else if ( direction_moving == 1)              // token is moving forward                                        -ab
    {
      /* action */
      /* insert into my_ctable entry, if not loop */

      /*
     i == 0: does not make sense
     i == 1: retransmission from the successor
     i >  1: consider it loop
      */

      if (!is_equal_addr(DA, station->TS))
    {
      for(i = 2; i < MAX_NUM_NODE; i++)
        {
          if (is_equal_addr(SA, station->my_ctable[table_index][i]))
        {
          return ERROR;
        }
        }

      copyaddr(station->my_ctable[table_index][node_index], SA);
    }

      if (is_equal_addr(station->TS, DA))
    {
      /* diagnosis */
      /* the token came back to this station, and the token is forward
         moving, token has made a full rotation */

      /* action */
      /* save the old connectivity table and start on constructing
         a new connectivity table */
```



```
        makenulladdr(station->my_ctable[table_index][j]);

      /* the first entry of my connectivity table is own station */
      node_index = 0;
      copyaddr(station->my_ctable[table_index][node_index], station->TS);
    }
   //
   // New section to cover new mode of token moving backwards to predecessor            -ab
   //
   else if ( direction_moving == 2) //  token is moving backward                        -ab
   //
   // In this case DA (destination address) and SA (source address) are reversed)
   //
   {

      /*
     i == 0: does not make sense
     i == 1: retransmission from the predecessor
     i >  1: consider it loop
      */

      if (!is_equal_addr(SA, station->TS))
    {
      for(i = 2; i < MAX_NUM_NODE; i++)
         {
          if (is_equal_addr(DA, station->my_ctable[table_index][i]))
         {
          return ERROR;
         }
         }

      copyaddr(station->my_ctable[table_index][node_index], DA);
    }

      if (is_equal_addr(station->TS, SA))
    {
      /* diagnosis */
      // token came back to this station and has made a full rotation.                     -ab
      // impossible case because for token to reach a backward direction twice then it should  -ab
      // have reached the token forward case first and caused the ctable to be frozen and swapped -ab
      // came back to this station, and the token is forward                                 -ab
      //
      return ERROR;
    }
```

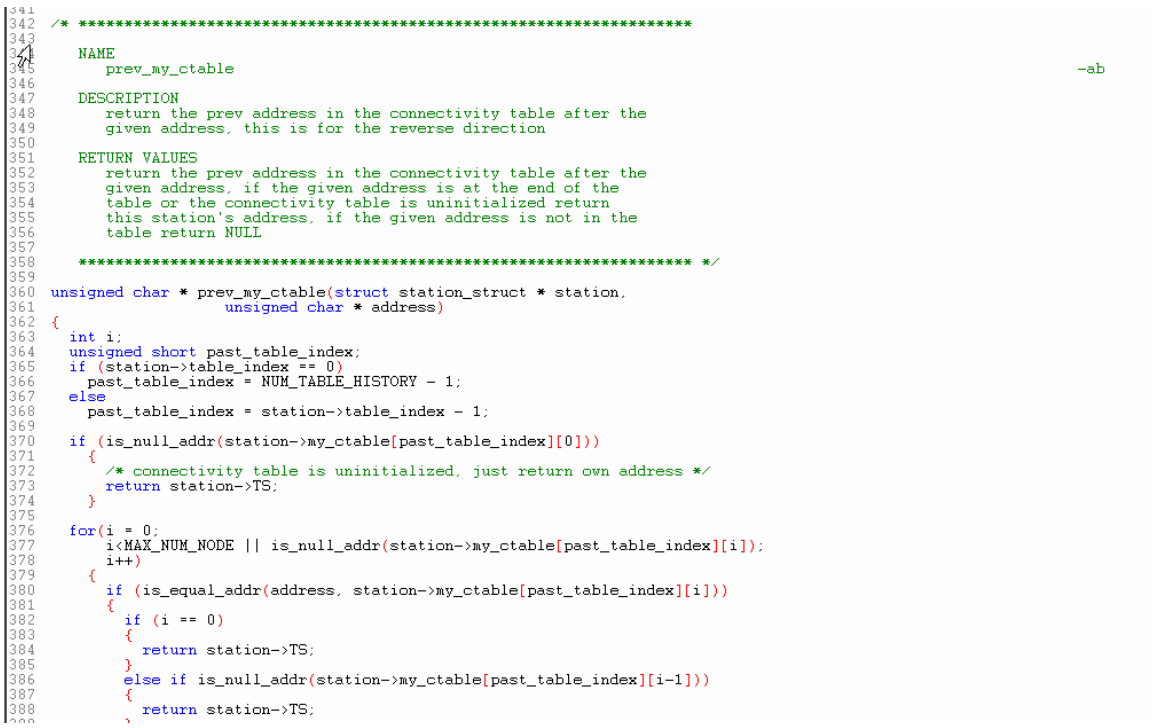
Figure 8.9: Modifications to *update_my_ctable* to recognize the direction of the token

Currently, given the unidirectional nature of token transmission, there is another table *next_my_ctable*. This table returns the next address in the connectivity table for the node.

For the token to traverse back the same path in its reverse direction, another table is created that returns the next address in the connectivity table for the node, in the reverse path. Which means, it is the previous address in the connectivity table. That is, the Predecessor node for a given station becomes the Successor node for the station on the reverse token transmission.

A table *prev_my_ctable is created* that provides the previous node information to the token on its reverse path.

prev_my_ctable

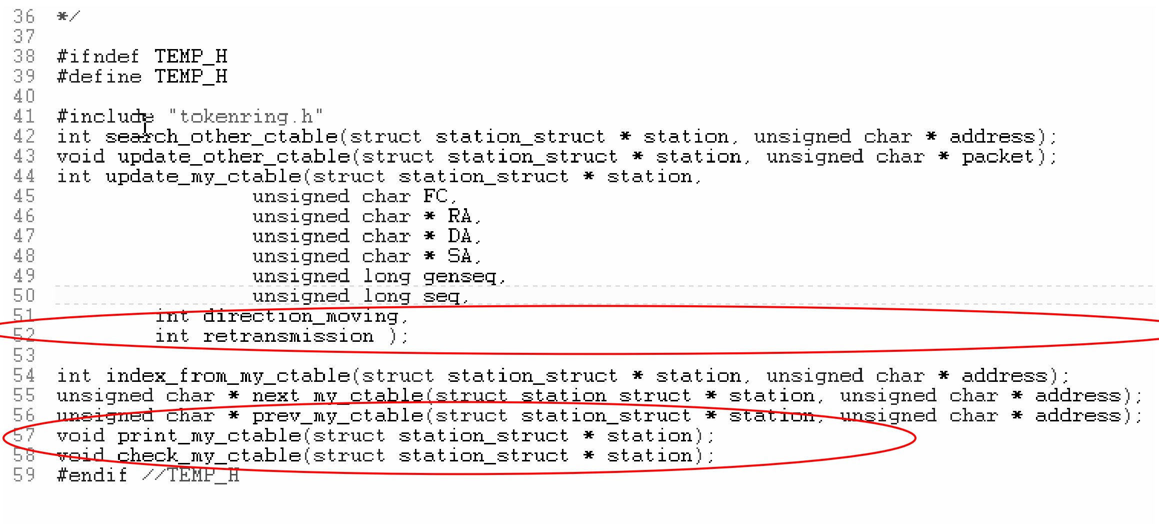


```
/* ******************************************************************

   NAME
      prev_my_ctable                                                         -ab

   DESCRIPTION
      return the prev address in the connectivity table after the
      given address, this is for the reverse direction

   RETURN VALUES
      return the prev address in the connectivity table after the
      given address, if the given address is at the end of the
      table or the connectivity table is uninitialized return
      this station's address, if the given address is not in the
      table return NULL

   ****************************************************************** */

unsigned char * prev_my_ctable(struct station_struct * station,
                   unsigned char * address)
{
  int i;
  unsigned short past_table_index;
  if (station->table_index == 0)
    past_table_index = NUM_TABLE_HISTORY - 1;
  else
    past_table_index = station->table_index - 1;

  if (is_null_addr(station->my_ctable[past_table_index][0]))
    {
      /* connectivity table is uninitialized, just return own address */
      return station->TS;
    }

  for(i = 0;
      i<MAX_NUM_NODE || is_null_addr(station->my_ctable[past_table_index][i]);
      i++)
    {
      if (is_equal_addr(address, station->my_ctable[past_table_index][i]))
      {
        if (i == 0)
        {
          return station->TS;
        }
        else if is_null_addr(station->my_ctable[past_table_index][i-1]))
        {
          return station->TS;
```

Figure 8.10: New Table *prev_my_ctable* created

The ***ctable.h*** file, or the header file for ***ctable***

A change is also made in the *ctable.h* file. It is shown as below:

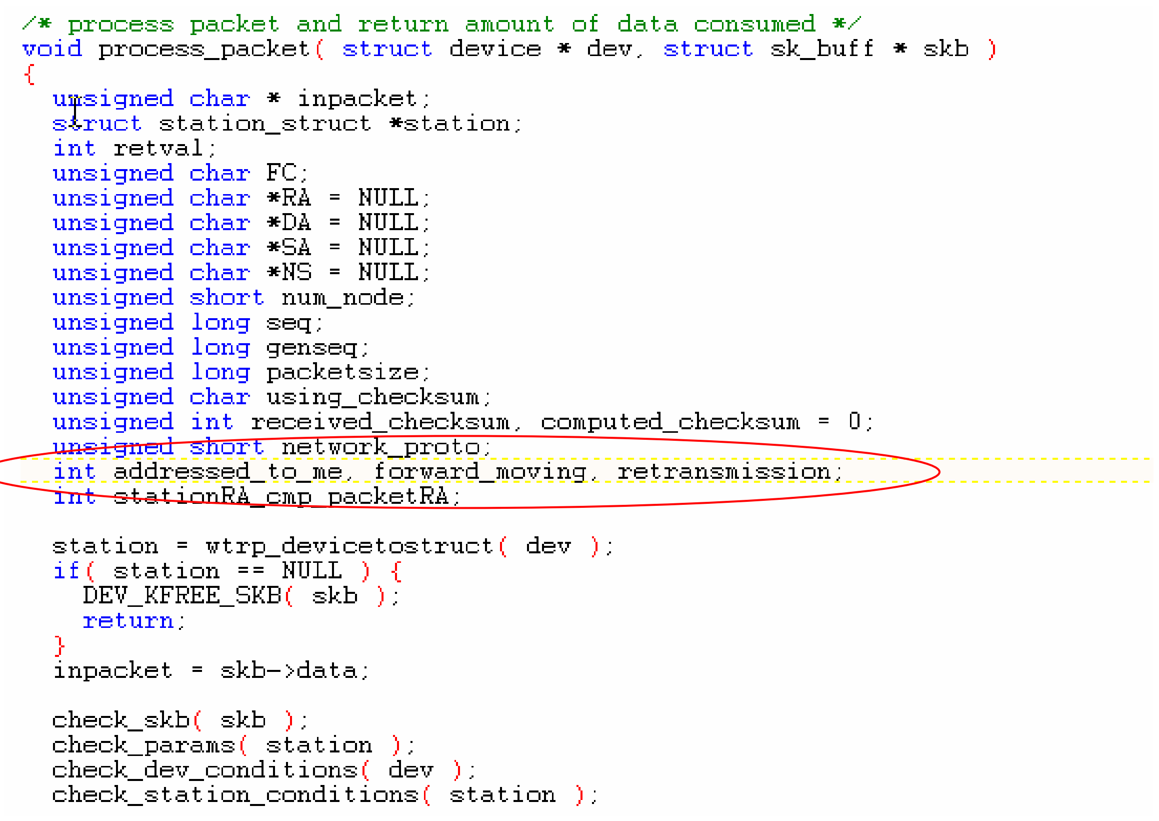


```
*/

#ifndef TEMP_H
#define TEMP_H

#include "tokenring.h"
int search_other_ctable(struct station_struct * station, unsigned char * address);
void update_other_ctable(struct station_struct * station, unsigned char * packet);
int update_my_ctable(struct station_struct * station,
                unsigned char FC,
                unsigned char * RA,
                unsigned char * DA,
                unsigned char * SA,
                unsigned long genseq,
                unsigned long seq,
        int direction_moving,
        int retransmission );

int index_from_my_ctable(struct station_struct * station, unsigned char * address);
unsigned char * next_my_ctable(struct station_struct * station, unsigned char * address);
unsigned char * prev_my_ctable(struct station_struct * station, unsigned char * address);
void print_my_ctable(struct station_struct * station);
void check_my_ctable(struct station_struct * station);
#endif //TEMP_H
```

Figure 8.11: Changes to *ctable.h* file

Tokenring.c

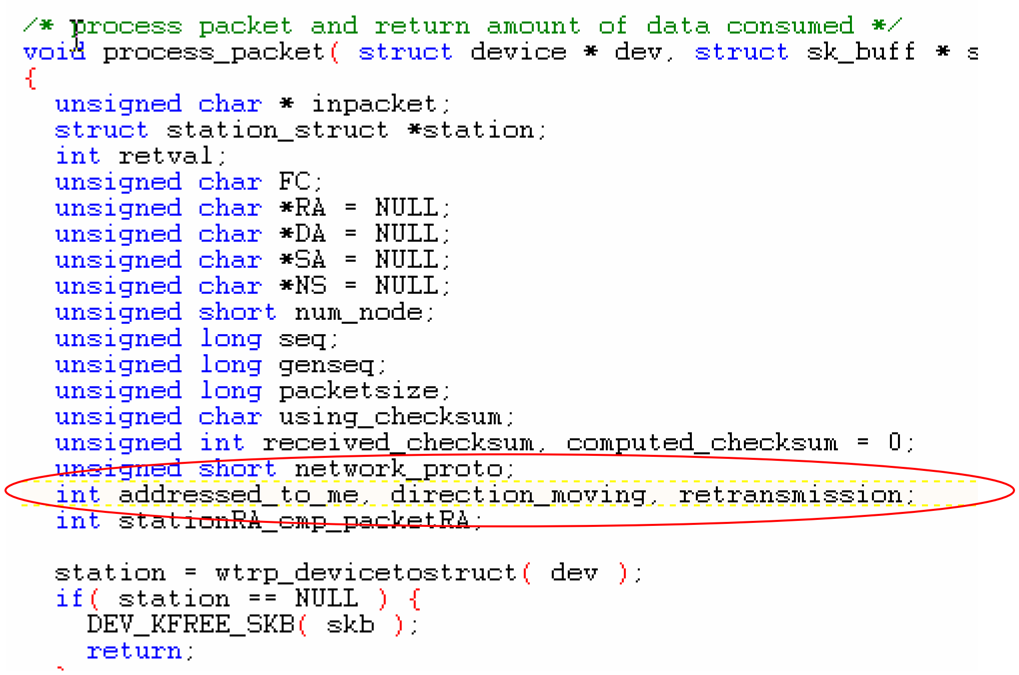
This is the most important file in the protocol code. It contains about 3500 lines of complicated code. Let us look at the code where the packet is processed:



```
/* process packet and return amount of data consumed */
void process_packet( struct device * dev, struct sk_buff * skb )
{
  unsigned char * inpacket;
  struct station_struct *station;
  int retval;
  unsigned char FC;
  unsigned char *RA = NULL;
  unsigned char *DA = NULL;
  unsigned char *SA = NULL;
  unsigned char *NS = NULL;
  unsigned short num_node;
  unsigned long seq;
  unsigned long genseq;
  unsigned long packetsize;
  unsigned char using_checksum;
  unsigned int received_checksum, computed_checksum = 0;
  unsigned short network_proto;
  int addressed_to_me, forward_moving, retransmission;
  int stationRA_cmp_packetRA;

  station = wtrp_devicetostruct( dev );
  if( station == NULL ) {
    DEV_KFREE_SKB( skb );
    return;
  }
  inpacket = skb->data;

  check_skb( skb );
  check_params( station );
  check_dev_conditions( dev );
  check_station_conditions( station );
```

Figure 8.12: *process_packet* in tokenring.c file

The *forward_moving* variable is changed to *direction_moving*

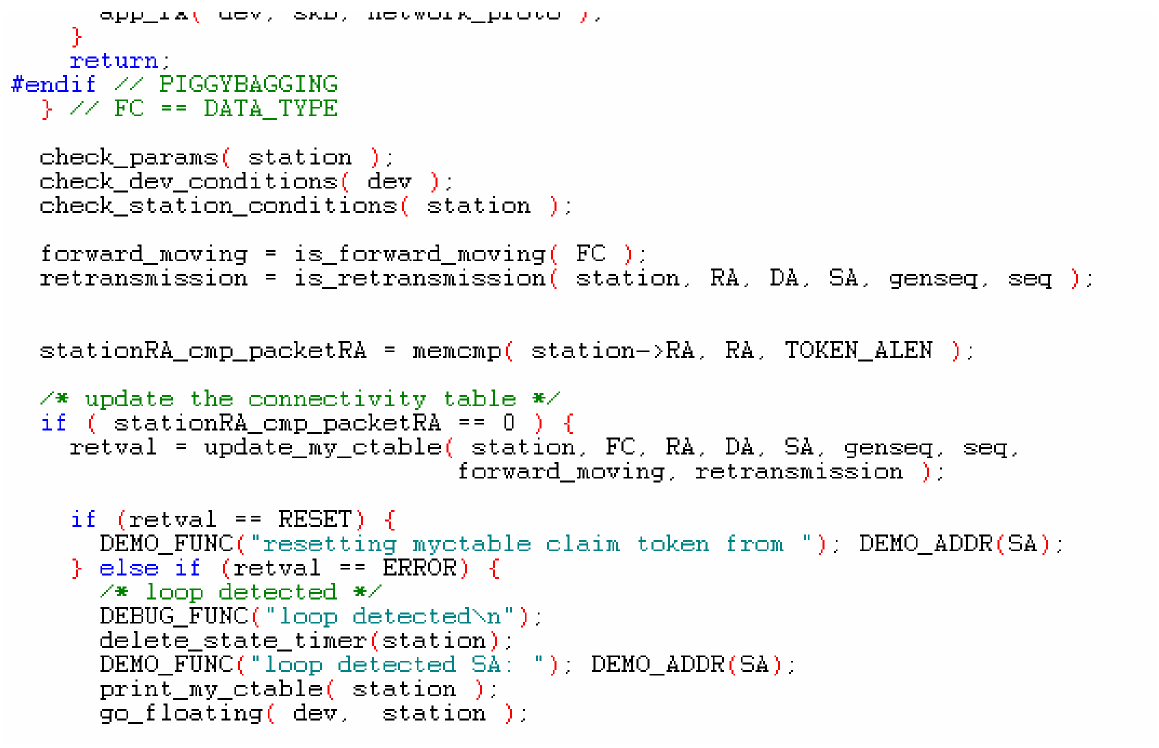


```
/* process packet and return amount of data consumed */
void process_packet( struct device * dev, struct sk_buff * s
{
  unsigned char * inpacket;
  struct station_struct *station;
  int retval;
  unsigned char FC;
  unsigned char *RA = NULL;
  unsigned char *DA = NULL;
  unsigned char *SA = NULL;
  unsigned char *NS = NULL;
  unsigned short num_node;
  unsigned long seq;
  unsigned long genseq;
  unsigned long packetsize;
  unsigned char using_checksum;
  unsigned int received_checksum, computed_checksum = 0;
  unsigned short network_proto;
  int addressed_to_me, direction_moving, retransmission;
  int stationRA_cmp_packetRA;

  station = wtrp_devicetostruct( dev );
  if( station == NULL ) {
    DEV_KFREE_SKB( skb );
    return;
```

Figure 8.13: Change made in *process_packet* in tokenring.c file

Now we need to initialize *direction_moving* and assign values for forward direction and backward direction.

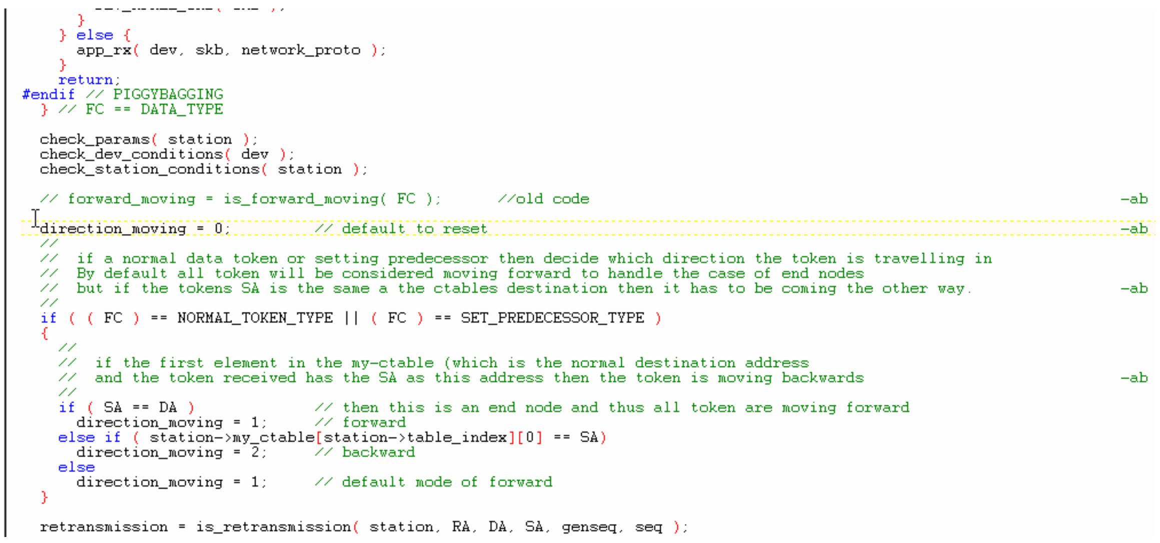
The original code is as shown below.



```
    }
    return;
#endif // PIGGYBAGGING
  } // FC == DATA_TYPE

  check_params( station );
  check_dev_conditions( dev );
  check_station_conditions( station );

  forward_moving = is_forward_moving( FC );
  retransmission = is_retransmission( station, RA, DA, SA, genseq, seq );


  stationRA_cmp_packetRA = memcmp( station->RA, RA, TOKEN_ALEN );

  /* update the connectivity table */
  if ( stationRA_cmp_packetRA == 0 ) {
    retval = update_my_ctable( station, FC, RA, DA, SA, genseq, seq,
                                    forward_moving, retransmission );

    if (retval == RESET) {
      DEMO_FUNC("resetting myctable claim token from "); DEMO_ADDR(SA);
    } else if (retval == ERROR) {
      /* loop detected */
      DEBUG_FUNC("loop detected\n");
      delete_state_timer(station);
      DEMO_FUNC("loop detected SA: "); DEMO_ADDR(SA);
      print_my_ctable( station );
      go_floating( dev,  station );
```

Figure 8.14: Original code for forward moving token

This original piece of code is changed as follows.



```
      }
    } else {
      app_rx( dev, skb, network_proto );
    }
    return;
#endif // PIGGYBAGGING
  } // FC == DATA_TYPE

  check_params( station );
  check_dev_conditions( dev );
  check_station_conditions( station );

  // forward_moving = is_forward_moving( FC );      //old code                                                -ab

  direction_moving = 0;          // default to reset                                                           -ab
  //
  //  if a normal data token or setting predecessor then decide which direction the token is travelling in
  //  By default all token will be considered moving forward to handle the case of end nodes
  //  but if the tokens SA is the same a the ctables destination then it has to be coming the other way.      -ab
  //
  if ( ( FC ) == NORMAL_TOKEN_TYPE || ( FC ) == SET_PREDECESSOR_TYPE )
  {
    //
    //  if the first element in the my-ctable (which is the normal destination address
    //  and the token received has the SA as this address then the token is moving backwards                  -ab
    //
    if ( SA == DA )              // then this is an end node and thus all token are moving forward
      direction_moving = 1;      // forward
    else if ( station->my_ctable[station->table_index][0] == SA)
      direction_moving = 2;      // backward
    else
      direction_moving = 1;      // default mode of forward
  }

  retransmission = is_retransmission( station, RA, DA, SA, genseq, seq );
```

Figure 8.15: Change made to the code to assign direction to the token

We default *direction_moving* to “0”. By default, all tokens will be considered forward moving.

The station or node compares the packet header for the Destination Address (DA) and the Source Address (SA), with the connectivity table it maintains. If the source address on the packet and the destination address on the connectivity table are the same, then the node is an end node, and the direction is forward moving.

Consider the following figure. Let the node be an end node.

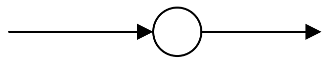


Figure 8.16: End Node is always forward moving

Whereas for the nodes other than the end nodes, the direction will be both forward and backwards.

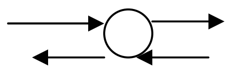
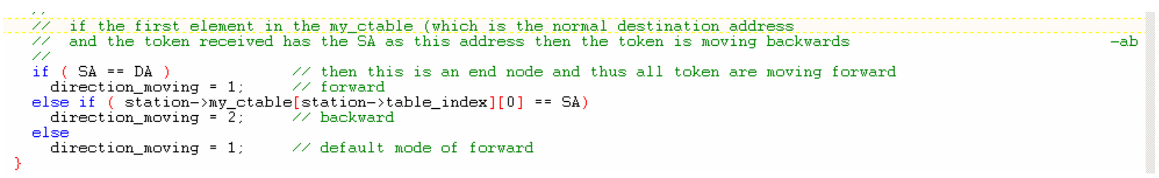


Figure 8.17: Nodes in the ring are bi-directional

For the middle nodes, if the Source Address (SA) of the packet, is the same as the first node in the sequence of its connectivity table, then the node assumes that the token is in the backward direction.



```
  //  if the first element in the my_ctable (which is the normal destination address
  //  and the token received has the SA as this address then the token is moving backwards                  -ab
  //
  if ( SA == DA )               // then this is an end node and thus all token are moving forward
    direction_moving = 1;       // forward
  else if ( station->my_ctable[station->table_index][0] == SA)
    direction_moving = 2;       // backward
  else
    direction_moving = 1;       // default mode of forward
}
```

Figure 8.18: Code Changes for Reverse Propagation of Token

Thus, the bi-directional flow of the token is achieved.

The joining and the leaving of the ring are achieved the same way as in the original protocol.

Chapter 9

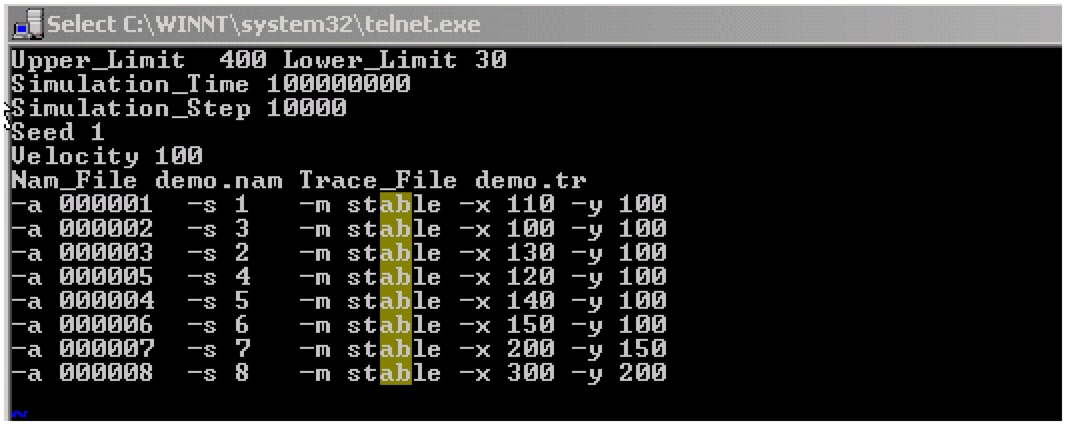
Testing the Modifications

The changes were then ported to the original WTRP code, and installed in a GNU/Kernel 2.6.9-5. The original code was developed for Kernel versions 2.2.19 up to version 2.4

There have been major Kernel changes from version 2.4 to 2.6 and hence, porting the application to the version 2.6.9-5 would involve a lot of man-hours as well as major changes in the code. Hence, Kernel 2.2.19 was used to compile and run the code with the changes. The code compiled properly and following are the figures depicting the results of the run for the simulation. The simulator is designed to run the implementation code and perform some rudimentary analysis. Its primary goal is to simulate the actual implementation in large network models that can scale up to a hundred or thousand nodes.

The simulation is run off the **node** file. In this file, you can specify the parameters for the nodes. You can increase or decrease the number of nodes in the simulation as well as specify the type of nodes and their locations. In this, there can be three types of nodes, viz., Stable, Bounce and Brownian. In *Stable* mode, nodes are fixed to a coordinate. In *Bounce* mode, nodes bounce off the square walls specified by the *Upper_Limit* and *Lower_Limit.* In *Brownian* mode, nodes drift in a random walk.

The figure below shows a snapshot of one of the Node files used in the simulation.



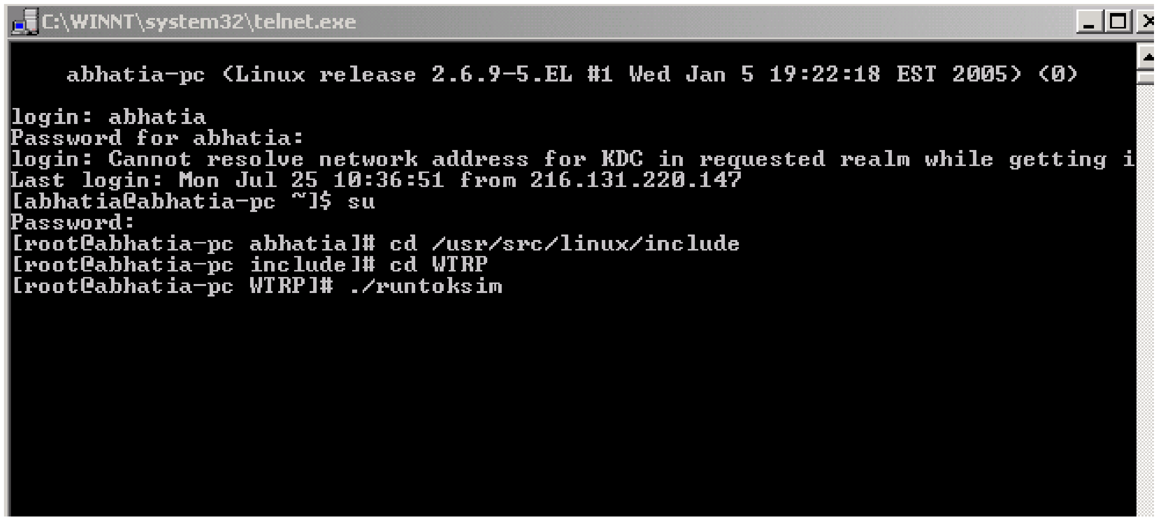
```
Upper_Limit  400 Lower_Limit 30
Simulation_Time 100000000
Simulation_Step 10000
Seed 1
Velocity 100
Nam_File demo.nam Trace_File demo.tr
-a 000001  -s 1   -m stable -x 110 -y 100
-a 000002  -s 3   -m stable -x 100 -y 100
-a 000003  -s 2   -m stable -x 130 -y 100
-a 000005  -s 4   -m stable -x 120 -y 100
-a 000004  -s 5   -m stable -x 140 -y 100
-a 000006  -s 6   -m stable -x 150 -y 100
-a 000007  -s 7   -m stable -x 200 -y 150
-a 000008  -s 8   -m stable -x 300 -y 200
```

Figure 9.1: Node file and its parameters

As stated above, the Upper_Limit and Lower_Limit specify the boundaries for the Bounce mode. But since we are interested only in the stable mode, this parameter is of little importance. The velocity parameter is for the Bounce and Brownian movements and hence, of little importance to us. What are important are the positions of the nodes in the stable mode. These are indicated by the x and y coordinates as shown in the snapshot above.

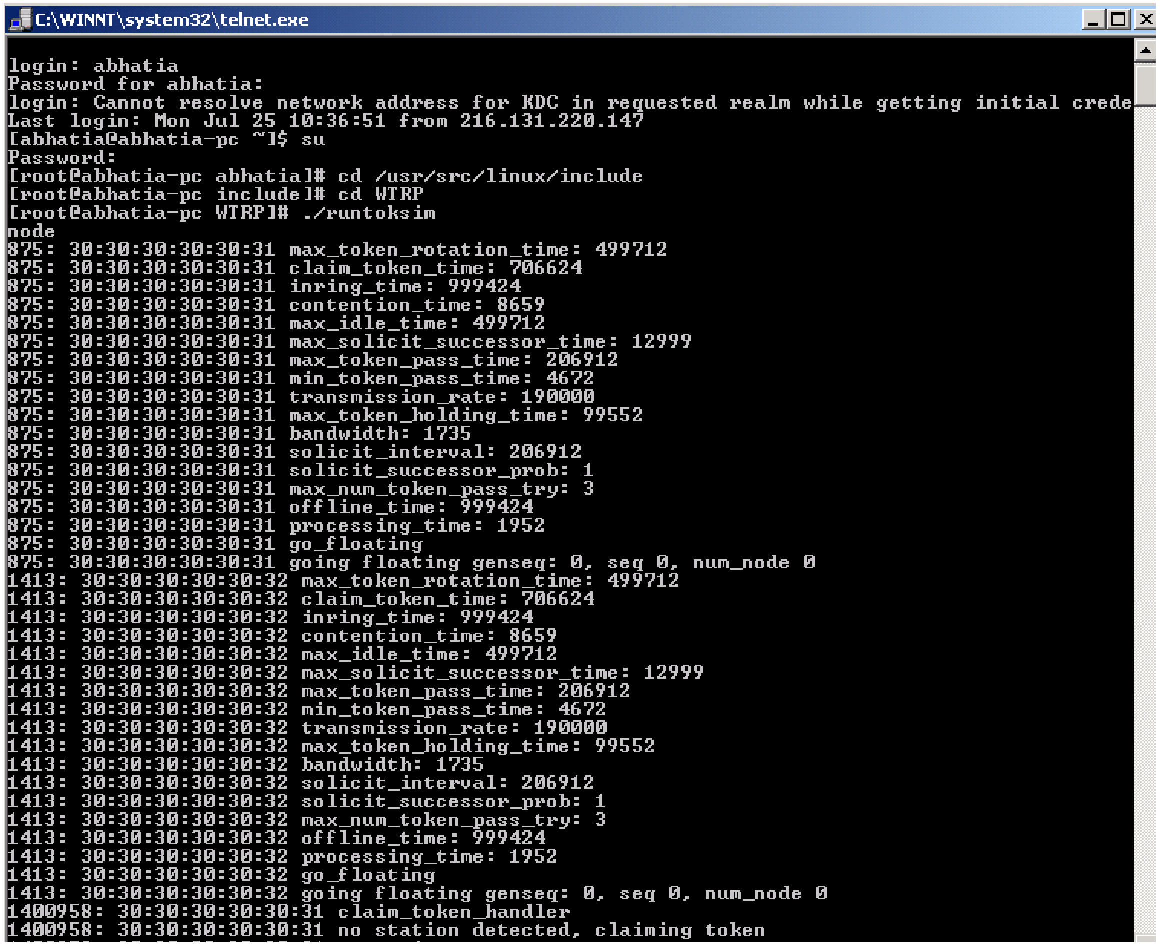
Running the Simulation

The simulation is run using the runtoksim file. The nodes are assigned a MAC address which can be seen in figure 9.3.



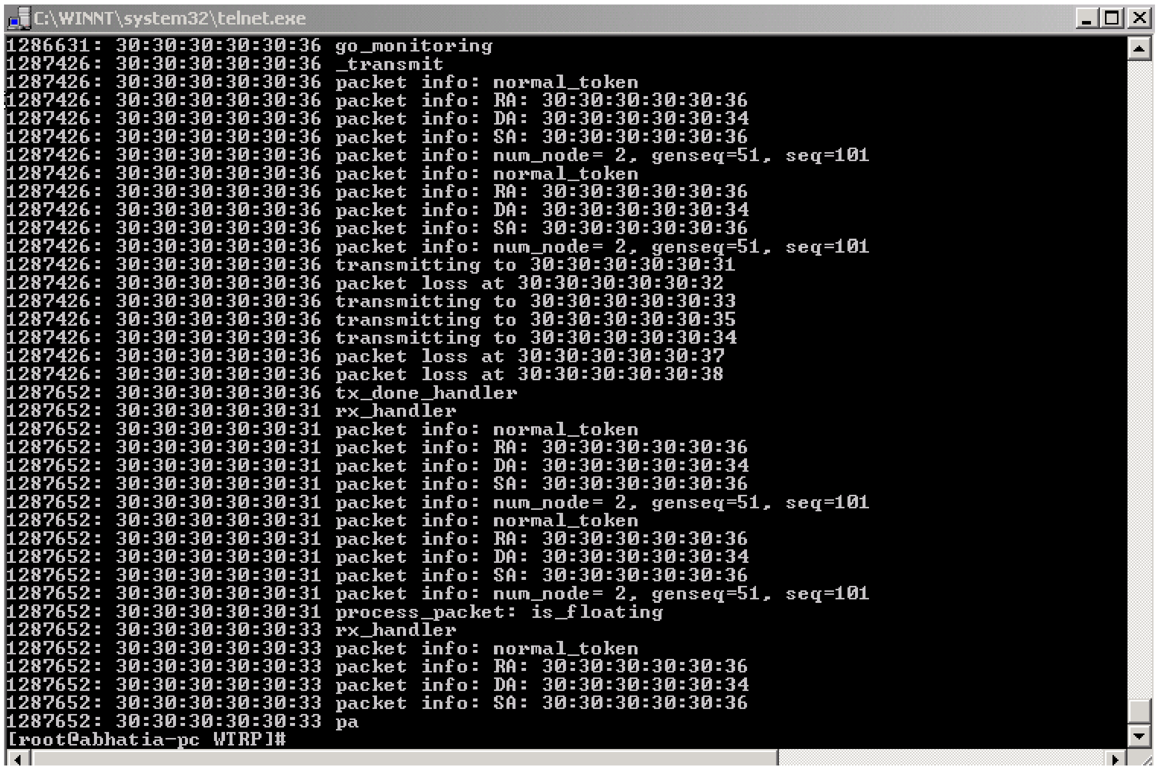
```
C:\WINNT\system32\telnet.exe

    abhatia-pc (Linux release 2.6.9-5.EL #1 Wed Jan 5 19:22:18 EST 2005) (0)

login: abhatia
Password for abhatia:
login: Cannot resolve network address for KDC in requested realm while getting i
Last login: Mon Jul 25 10:36:51 from 216.131.220.147
[abhatia@abhatia-pc ~]$ su
Password:
[root@abhatia-pc abhatia]# cd /usr/src/linux/include
[root@abhatia-pc include]# cd WTRP
[root@abhatia-pc WTRP]# ./runtoksim
```

Figure 9.2: Running the simulation



```
C:\WINNT\system32\telnet.exe
login: abhatia
Password for abhatia:
login: Cannot resolve network address for KDC in requested realm while getting initial crede
Last login: Mon Jul 25 10:36:51 from 216.131.220.147
[abhatia@abhatia-pc ~]$ su
Password:
[root@abhatia-pc abhatia]# cd /usr/src/linux/include
[root@abhatia-pc include]# cd WTRP
[root@abhatia-pc WTRP]# ./runtoksim
node
875: 30:30:30:30:30:31 max_token_rotation_time: 499712
875: 30:30:30:30:30:31 claim_token_time: 706624
875: 30:30:30:30:30:31 inring_time: 999424
875: 30:30:30:30:30:31 contention_time: 8659
875: 30:30:30:30:30:31 max_idle_time: 499712
875: 30:30:30:30:30:31 max_solicit_successor_time: 12999
875: 30:30:30:30:30:31 max_token_pass_time: 206912
875: 30:30:30:30:30:31 min_token_pass_time: 4672
875: 30:30:30:30:30:31 transmission_rate: 190000
875: 30:30:30:30:30:31 max_token_holding_time: 99552
875: 30:30:30:30:30:31 bandwidth: 1735
875: 30:30:30:30:30:31 solicit_interval: 206912
875: 30:30:30:30:30:31 solicit_successor_prob: 1
875: 30:30:30:30:30:31 max_num_token_pass_try: 3
875: 30:30:30:30:30:31 offline_time: 999424
875: 30:30:30:30:30:31 processing_time: 1952
875: 30:30:30:30:30:31 go_floating
875: 30:30:30:30:30:31 going floating genseq: 0, seq 0, num_node 0
1413: 30:30:30:30:30:32 max_token_rotation_time: 499712
1413: 30:30:30:30:30:32 claim_token_time: 706624
1413: 30:30:30:30:30:32 inring_time: 999424
1413: 30:30:30:30:30:32 contention_time: 8659
1413: 30:30:30:30:30:32 max_idle_time: 499712
1413: 30:30:30:30:30:32 max_solicit_successor_time: 12999
1413: 30:30:30:30:30:32 max_token_pass_time: 206912
1413: 30:30:30:30:30:32 min_token_pass_time: 4672
1413: 30:30:30:30:30:32 transmission_rate: 190000
1413: 30:30:30:30:30:32 max_token_holding_time: 99552
1413: 30:30:30:30:30:32 bandwidth: 1735
1413: 30:30:30:30:30:32 solicit_interval: 206912
1413: 30:30:30:30:30:32 solicit_successor_prob: 1
1413: 30:30:30:30:30:32 max_num_token_pass_try: 3
1413: 30:30:30:30:30:32 offline_time: 999424
1413: 30:30:30:30:30:32 processing_time: 1952
1413: 30:30:30:30:30:32 go_floating
1413: 30:30:30:30:30:32 going floating genseq: 0, seq 0, num_node 0
1400958: 30:30:30:30:30:31 claim_token_handler
1400958: 30:30:30:30:30:31 no station detected, claiming token
```

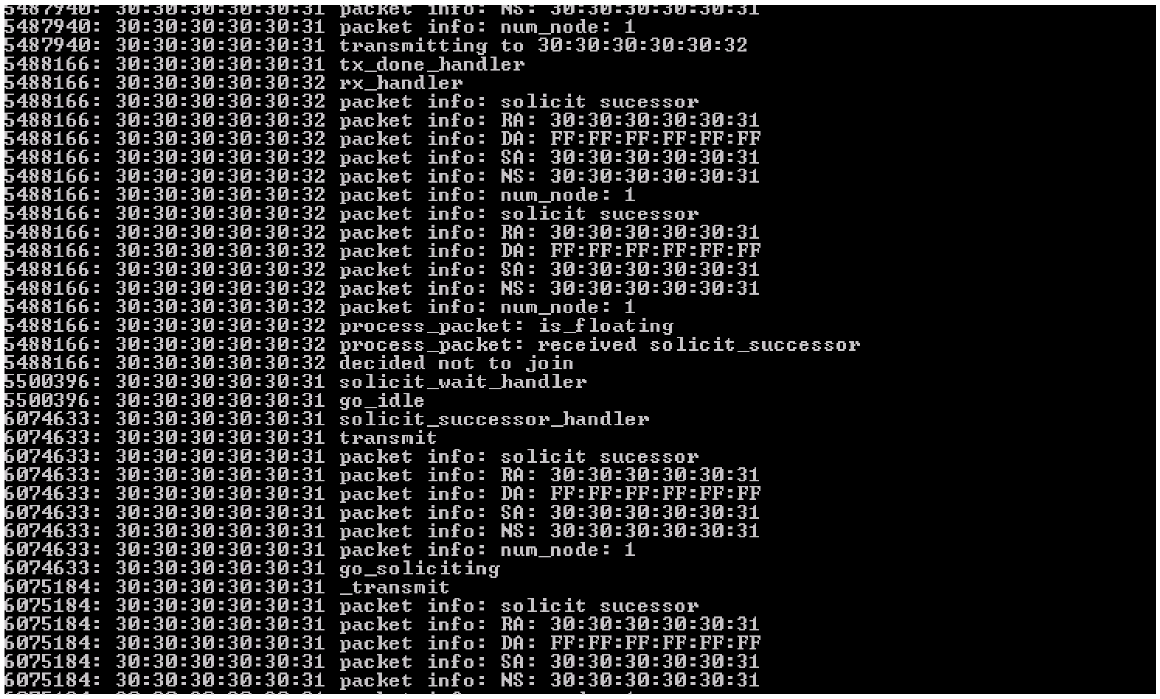
Figure 9.3: Simulation Parameters



```
1286631: 30:30:30:30:30:36 go_monitoring
1287426: 30:30:30:30:30:36 _transmit
1287426: 30:30:30:30:30:36 packet info: normal_token
1287426: 30:30:30:30:30:36 packet info: RA: 30:30:30:30:30:36
1287426: 30:30:30:30:30:36 packet info: DA: 30:30:30:30:30:34
1287426: 30:30:30:30:30:36 packet info: SA: 30:30:30:30:30:36
1287426: 30:30:30:30:30:36 packet info: num_node= 2, genseq=51, seq=101
1287426: 30:30:30:30:30:36 packet info: normal_token
1287426: 30:30:30:30:30:36 packet info: RA: 30:30:30:30:30:36
1287426: 30:30:30:30:30:36 packet info: DA: 30:30:30:30:30:34
1287426: 30:30:30:30:30:36 packet info: SA: 30:30:30:30:30:36
1287426: 30:30:30:30:30:36 packet info: num_node= 2, genseq=51, seq=101
1287426: 30:30:30:30:30:36 transmitting to 30:30:30:30:30:31
1287426: 30:30:30:30:30:36 packet loss at 30:30:30:30:30:32
1287426: 30:30:30:30:30:36 transmitting to 30:30:30:30:30:33
1287426: 30:30:30:30:30:36 transmitting to 30:30:30:30:30:35
1287426: 30:30:30:30:30:36 transmitting to 30:30:30:30:30:34
1287426: 30:30:30:30:30:36 packet loss at 30:30:30:30:30:37
1287426: 30:30:30:30:30:36 packet loss at 30:30:30:30:30:38
1287652: 30:30:30:30:30:36 tx_done_handler
1287652: 30:30:30:30:30:31 rx_handler
1287652: 30:30:30:30:30:31 packet info: normal_token
1287652: 30:30:30:30:30:31 packet info: RA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:31 packet info: DA: 30:30:30:30:30:34
1287652: 30:30:30:30:30:31 packet info: SA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:31 packet info: num_node= 2, genseq=51, seq=101
1287652: 30:30:30:30:30:31 packet info: normal_token
1287652: 30:30:30:30:30:31 packet info: RA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:31 packet info: DA: 30:30:30:30:30:34
1287652: 30:30:30:30:30:31 packet info: SA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:31 packet info: num_node= 2, genseq=51, seq=101
1287652: 30:30:30:30:30:31 process_packet: is_floating
1287652: 30:30:30:30:30:33 rx_handler
1287652: 30:30:30:30:30:33 packet info: normal_token
1287652: 30:30:30:30:30:33 packet info: RA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:33 packet info: DA: 30:30:30:30:30:34
1287652: 30:30:30:30:30:33 packet info: SA: 30:30:30:30:30:36
1287652: 30:30:30:30:30:33 pa
[root@abhatia-pc WTRP]#
```

Figure 9.4: Successful transmission of data packet along with a few loss of packets

In the above figure, the node with the MAC address 30:30:30:30:30:36 gets the token, on which the Ring Address (RA) is the MAC address 30:30:30:30:30:36. This means that the token was generated at this node. The Destination Address (DA) is the node with the MAC address 30:30:30:30:30:34 and the Source Address (SA) is the node with the MAC address 30:30:30:30:30:36, which means that the token was generated at this node. Since this node has the token, it transmits the data packets to all other nodes. As can be seen, the packets get transmitted to other nodes, and there are a few packet losses at some nodes.



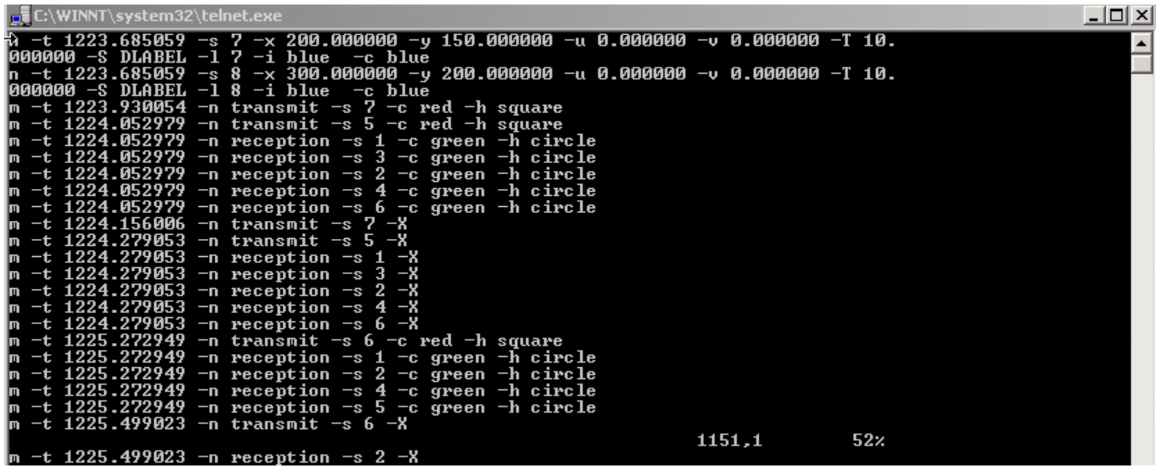
```
5487940: 30:30:30:30:30:31 packet info: NS: 30:30:30:30:30:31
5487940: 30:30:30:30:30:31 packet info: num_node: 1
5487940: 30:30:30:30:30:31 transmitting to 30:30:30:30:30:32
5488166: 30:30:30:30:30:31 tx_done_handler
5488166: 30:30:30:30:30:32 rx_handler
5488166: 30:30:30:30:30:32 packet info: solicit sucessor
5488166: 30:30:30:30:30:32 packet info: RA: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: DA: FF:FF:FF:FF:FF:FF
5488166: 30:30:30:30:30:32 packet info: SA: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: NS: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: num_node: 1
5488166: 30:30:30:30:30:32 packet info: solicit sucessor
5488166: 30:30:30:30:30:32 packet info: RA: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: DA: FF:FF:FF:FF:FF:FF
5488166: 30:30:30:30:30:32 packet info: SA: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: NS: 30:30:30:30:30:31
5488166: 30:30:30:30:30:32 packet info: num_node: 1
5488166: 30:30:30:30:30:32 process_packet: is_floating
5488166: 30:30:30:30:30:32 process_packet: received solicit_successor
5488166: 30:30:30:30:30:32 decided not to join
5500396: 30:30:30:30:30:31 solicit_wait_handler
5500396: 30:30:30:30:30:31 go_idle
6074633: 30:30:30:30:30:31 solicit_successor_handler
6074633: 30:30:30:30:30:31 transmit
6074633: 30:30:30:30:30:31 packet info: solicit sucessor
6074633: 30:30:30:30:30:31 packet info: RA: 30:30:30:30:30:31
6074633: 30:30:30:30:30:31 packet info: DA: FF:FF:FF:FF:FF:FF
6074633: 30:30:30:30:30:31 packet info: SA: 30:30:30:30:30:31
6074633: 30:30:30:30:30:31 packet info: NS: 30:30:30:30:30:31
6074633: 30:30:30:30:30:31 packet info: num_node: 1
6074633: 30:30:30:30:30:31 go_soliciting
6075184: 30:30:30:30:30:31 _transmit
6075184: 30:30:30:30:30:31 packet info: solicit sucessor
6075184: 30:30:30:30:30:31 packet info: RA: 30:30:30:30:30:31
6075184: 30:30:30:30:30:31 packet info: DA: FF:FF:FF:FF:FF:FF
6075184: 30:30:30:30:30:31 packet info: SA: 30:30:30:30:30:31
6075184: 30:30:30:30:30:31 packet info: NS: 30:30:30:30:30:31
```

Figure 9.5: Joining Process – Node decides not to join

This simulation was achieved with only two nodes, in a stable mode. A self ring was formed with the node with MAC Address 30:30:30:30:30:31. Hence the node 30:30:30:30:30:31 sends a solicit_successor token to the node 30:30:30:30:30:32. Since the Destination Address (DA) has not yet been assigned, as its in solicit successor mode, the DA shows FF:FF:FF:FF:FF:FF, or the Ethernet Broadcast address, as the Simulation is run off a machine connected to the Ethernet.

The simulation run generates two files, the demo.nam and the demo.tr files. Both these files provide us with data that can be analyzed to study the protocol.

Since the demo.nam file is extremely big, certain sections of the file are being shown here.

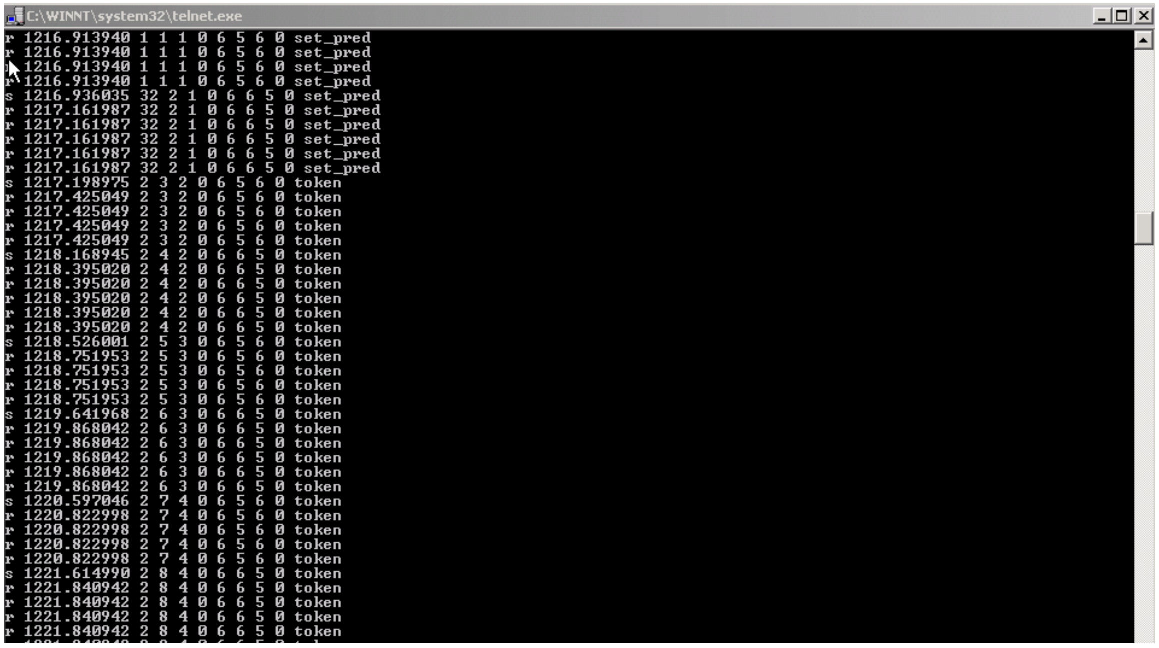


```
C:\WINNT\system32\telnet.exe
n -t 1223.685059 -s 7 -x 200.000000 -y 150.000000 -u 0.000000 -v 0.000000 -T 10.
000000 -S DLABEL -l 7 -i blue  -c blue
n -t 1223.685059 -s 8 -x 300.000000 -y 200.000000 -u 0.000000 -v 0.000000 -T 10.
000000 -S DLABEL -l 8 -i blue  -c blue
m -t 1223.930054 -n transmit -s 7 -c red -h square
m -t 1224.052979 -n transmit -s 5 -c red -h square
m -t 1224.052979 -n reception -s 1 -c green -h circle
m -t 1224.052979 -n reception -s 3 -c green -h circle
m -t 1224.052979 -n reception -s 2 -c green -h circle
m -t 1224.052979 -n reception -s 4 -c green -h circle
m -t 1224.052979 -n reception -s 6 -c green -h circle
m -t 1224.156006 -n transmit -s 7 -X
m -t 1224.279053 -n transmit -s 5 -X
m -t 1224.279053 -n reception -s 1 -X
m -t 1224.279053 -n reception -s 3 -X
m -t 1224.279053 -n reception -s 2 -X
m -t 1224.279053 -n reception -s 4 -X
m -t 1224.279053 -n reception -s 6 -X
m -t 1225.272949 -n transmit -s 6 -c red -h square
m -t 1225.272949 -n reception -s 1 -c green -h circle
m -t 1225.272949 -n reception -s 2 -c green -h circle
m -t 1225.272949 -n reception -s 4 -c green -h circle
m -t 1225.272949 -n reception -s 5 -c green -h circle
m -t 1225.499023 -n transmit -s 6 -X
                                                        1151,1        52%
m -t 1225.499023 -n reception -s 2 -X
```

Figure 9.6: Demo.nam file

A nam file gets generated which provides the visual aspect of the simulation. Since it requires further configuration and addition of software, we analyze the raw data in the demo.nam file. You can see, when node 6 is transmitting and nodes 1,2,3,4,5 are in the reception mode, that is, receive data from 6. But nodes 7 and 8 don't receive the data, and hence the data packet gets lost. Currently, the transmission station is highlighted red with the start_transmission, and the receiving stations are highlighted green by the start_reception function.

The demo.tr file gives us the following output.



```
C:\WINNT\system32\telnet.exe
r 1216.913940 1 1 1 0 6 5 6 0 set_pred
r 1216.913940 1 1 1 0 6 5 6 0 set_pred
r 1216.913940 1 1 1 0 6 5 6 0 set_pred
r 1216.913940 1 1 1 0 6 5 6 0 set_pred
s 1216.936035 32 2 1 0 6 6 5 0 set_pred
r 1217.161987 32 2 1 0 6 6 5 0 set_pred
r 1217.161987 32 2 1 0 6 6 5 0 set_pred
r 1217.161987 32 2 1 0 6 6 5 0 set_pred
r 1217.161987 32 2 1 0 6 6 5 0 set_pred
r 1217.161987 32 2 1 0 6 6 5 0 set_pred
s 1217.198975 2 3 2 0 6 5 6 0 token
r 1217.425049 2 3 2 0 6 5 6 0 token
r 1217.425049 2 3 2 0 6 5 6 0 token
r 1217.425049 2 3 2 0 6 5 6 0 token
r 1217.425049 2 3 2 0 6 5 6 0 token
s 1218.168945 2 4 2 0 6 6 5 0 token
r 1218.395020 2 4 2 0 6 6 5 0 token
r 1218.395020 2 4 2 0 6 6 5 0 token
r 1218.395020 2 4 2 0 6 6 5 0 token
r 1218.395020 2 4 2 0 6 6 5 0 token
r 1218.395020 2 4 2 0 6 6 5 0 token
s 1218.526001 2 5 3 0 6 5 6 0 token
r 1218.751953 2 5 3 0 6 5 6 0 token
r 1218.751953 2 5 3 0 6 5 6 0 token
r 1218.751953 2 5 3 0 6 5 6 0 token
r 1218.751953 2 5 3 0 6 5 6 0 token
s 1219.641968 2 6 3 0 6 6 5 0 token
r 1219.868042 2 6 3 0 6 6 5 0 token
r 1219.868042 2 6 3 0 6 6 5 0 token
r 1219.868042 2 6 3 0 6 6 5 0 token
r 1219.868042 2 6 3 0 6 6 5 0 token
r 1219.868042 2 6 3 0 6 6 5 0 token
s 1220.597046 2 7 4 0 6 5 6 0 token
r 1220.822998 2 7 4 0 6 5 6 0 token
r 1220.822998 2 7 4 0 6 5 6 0 token
r 1220.822998 2 7 4 0 6 5 6 0 token
r 1220.822998 2 7 4 0 6 5 6 0 token
s 1221.614990 2 8 4 0 6 6 5 0 token
r 1221.840942 2 8 4 0 6 6 5 0 token
r 1221.840942 2 8 4 0 6 6 5 0 token
r 1221.840942 2 8 4 0 6 6 5 0 token
r 1221.840942 2 8 4 0 6 6 5 0 token
```

Figure 9.7: Demo.tr file

The record_transmission and record_reception functions are used by the demo.tr file to keep a log of the packet events. These functions create a log of information of the packet header with the time stamp.

Chapter 10

Simulation Results

Since implementation of the Modified WTRP code in the field is beyond the scope of this thesis, we used the simulation to verify that the modifications we make to the WTRP code and test for the robustness of the protocol over long distances.

The developers of Wireless Token Ring Protocol created a simulator – Wireless Token Ring Simulator (WTRS) (3). The purpose of building this simulator was to simulate the actual implementation code in large network models that could scale up to a hundred or thousand nodes. The design of this simulator eliminated the need for a separate code or program for simulation as well as deployment of the protocol. This simulator was used in the present study.

The Simulator consists of two parts: The WTRP Module (or the Modified WTRP Module) and the Simulator Module. The WTRP module consists of the entire code for the Wireless Token Ring Protocol. The Simulator module is a code that provides an interface for scheduling and data packet manipulation to the WTRP module. It consists of data structures that hold the node information, a basic data application that sends periodic packets to the WTRP Core module, and functions to implement event queues, add event and delete event. Hence, the Simulator is designed for the purpose of simulating the actual code in a wireless environment with multiple nodes. It facilitates the implementation code and allows us to do some basic analysis. We use the same Simulator Module to analyze and compare the WTRP module and the Modified WTRP module. We specify the position of the nodes, and their topology to the Simulator, along with the Simulation Time. The specified nodes are then assigned a MAC address by the simulator, and the basic data application Constant Bit Rate (CBR) sends periodic packets to these nodes, to be transmitted. From there on, the WTRP (or Modified WTRP) code takes over and data transmission is then carried out using the actual code that was modified. Hence, we can adequately test our modified protocol using this simulation and studying the packet loss and corruption from the output.

The Simulator captures the effects of Signal to Interference ratio, free space propagation, thus modeling a simple Direct Sequence Spread Spectrum. It calculates packet loss on the basis of distance and power, and packet corruption based on Signal to Interference Ratio

The protocol was tested and simulated for an increasing number of nodes, with different node types, and different x and y coordinates. The protocol shows the same robustness of the parent protocol and similar characteristics when handling the joining and the leaving of the ring. Both the protocols (WTRP and Modified WTRP) were run, for a number of nodes as well as a number of different positions (co-ordinates) for the nodes. The nodes were increased from 3 nodes initially to 8 nodes. Each time, the results were analyzed and compared. Since the protocol is simulated over Ethernet, it inherits all the overheads associated with Ethernet protocol.

The demo.nam file gets created when the simulation is run. It shows the node movements, transmissions, receptions and ring address of each node. The packet losses at each node were identified. The packet losses are based on the distance between the nodes and packet corruption is based on Signal to Interference Ratio.

Below are the results for each of the node positions.

Node	Type	X Coordinate	Y Coordinate
000001	Stable	90	100
000002	Stable	100	100
000003	Stable	110	100

Table 10.1: Node Parameters for First Simulation

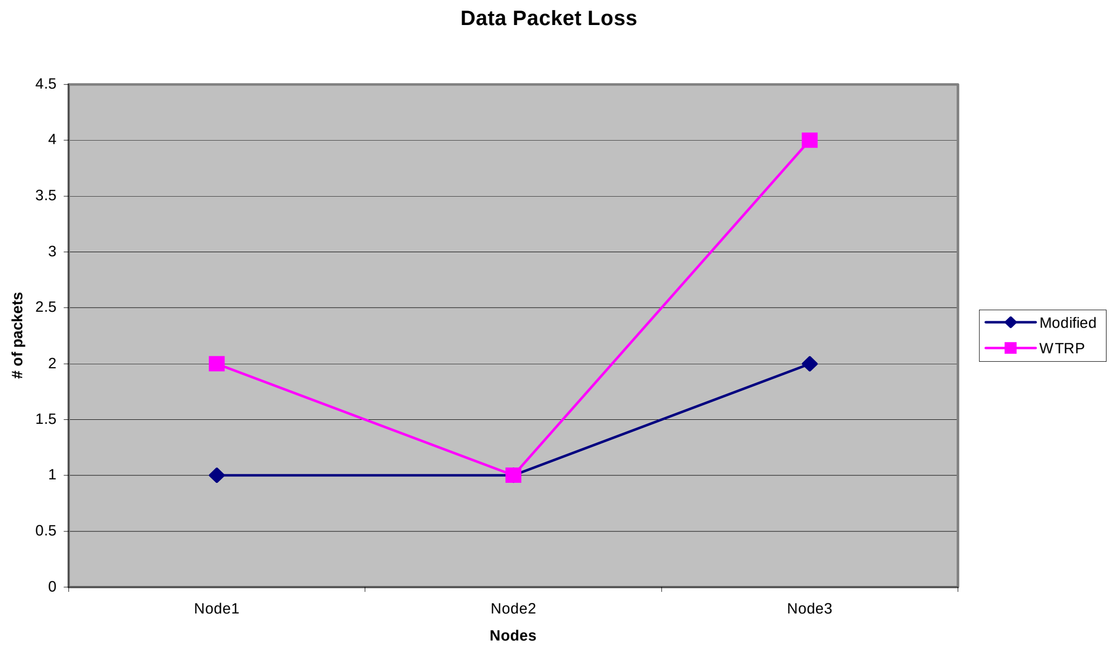


Figure 10.1: Data Packet Loss in First Simulation

Node	Type	X Coordinate	Y Coordinate
000001	Stable	80	100
000002	Stable	90	100
000003	Stable	100	100
000004	Stable	110	100
000005	Stable	120	100

Table 10.2: Node Parameters for Second Simulation

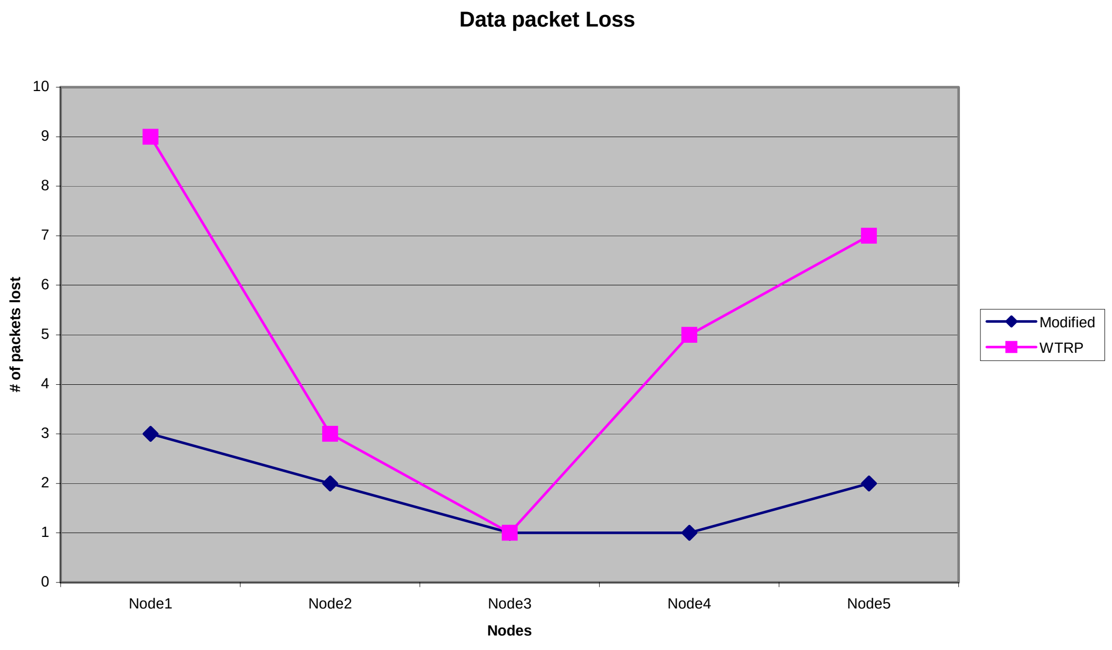


Figure 10.2: Data Packet Loss in Second Simulation

Node	Type	X Coordinate	Y Coordinate
000001	Stable	40	100
000002	Stable	50	100
000003	Stable	70	100
000004	Stable	80	100
000005	Stable	100	100
000006	Stable	120	100
000007	Stable	140	110
000008	Stable	170	130

Table 10.3: Node Parameters for Third Simulation

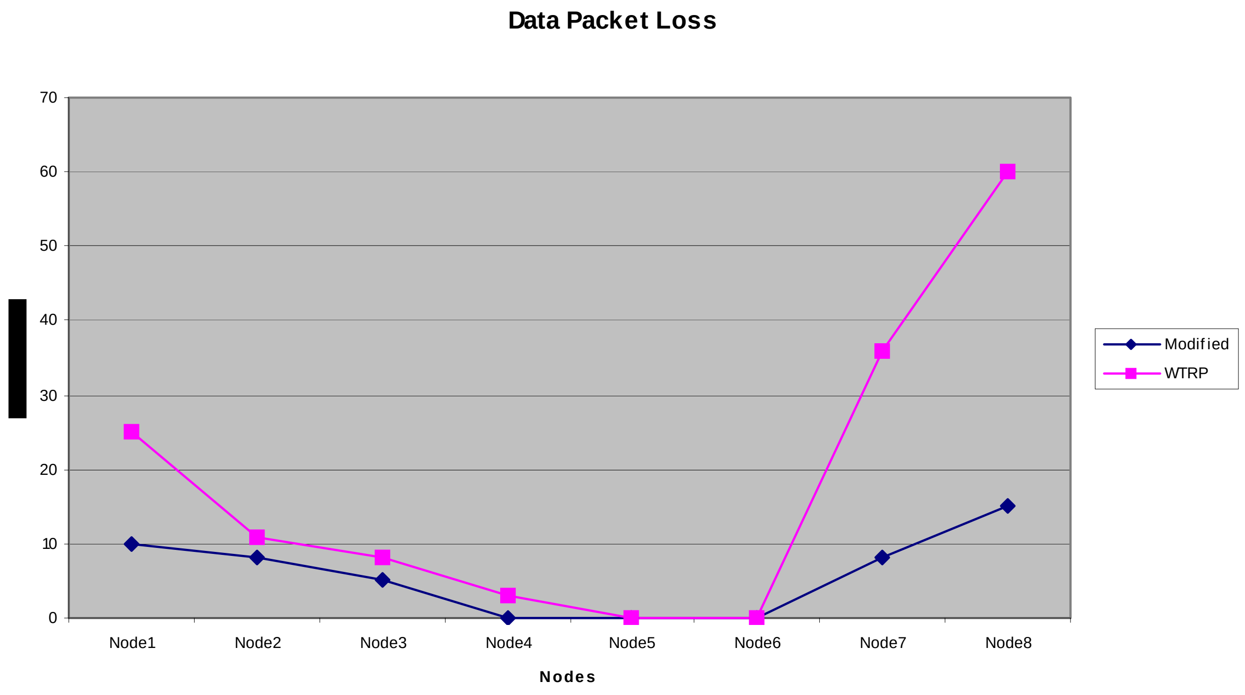


Figure 10.3: Data Packet Loss in Third Simulation

As can be seen from the graph above, the loss of packets in the Modified WTRP protocol is far less than that in the original protocol (WTRP). As the distance between the nodes increases, packets get increasingly lost in the WTRP, as compared to our modified protocol.

This shows that in cases of really long distances, like those on the field, the Modified Wireless Token Ring Protocol can be effectively used, without too much loss of data.

Chapter 11

11.1 About the Program

The program attached in the appendix, for the present problem, has the following aspects to it:

- Resides on the central computer
- Collects the data transmitted by the sensors placed at the highway work zone
- Displays this data on a website with as little a time lag as possible
- Allows for archiving of data for future references

The program is written in Java.

Why Java?

The advantages of Java Technology over other technologies are well known. They are documented as follows:

Java is..

Cross Platform: This is very important in the diversified mobile device market. In a heterogeneous enterprise environment, the ability to develop and maintain a single client for all devices results in huge savings.
For the project at hand, no matter what the Operating System of the computer, or that of the sensors, Java can be implemented on any machine.

Robust: Since Java applications are completely managed, the byte-code is verified before execution, and memory leaks are reclaimed by garbage collectors. Even if a Java application does crash, it is contained within the virtual machine. It will not affect other sensitive applications or data on the device.

Secure: The Java runtime provides advanced security features through a domain-based security manager and standard security Application Programming Interfaces (API)

Object oriented: The Java language is a well-designed, object–oriented language with vast library support. There is a vast pool of existing Java developers.
Hence, any extensions to this project can be developed by almost anyone familiar with Java

Wide adoption at the back end: It is relatively easy to make Java clients work with Java application servers and messaging servers. Due to the wide adoption of Java 2 Enterprise Edition (J2EE) on the server side, mobile Java is the leading candidate for front-end applications.

Technical merits as those above are not the only factors that determine the success of the technology. Other factors such as the business values are just as important.
The most significant aspect of development in Java is its Open Standards approach.
As a result of all the above reasons, Java was selected as the technology of choice for this thesis.

11.2 The Application

The application, like every other application, consists of a server side and a client side. The requirements for developing the application were as follows:

- End user enters the username and a text message from the cell phone to be stored on the server
- Allow a J2ME client to store a text message on the server
- The username is a way to identify who the message came from
- Have the ability to view all the messages in a desktop browser
- The messages should be displayed in the order of time received with the latest messages appearing first
- Have the ability to allow the J2ME client to see all the messages stored on the database
- This will allow the end user to see all the messages stored on the server

These requirements make sure that there is a to-and-fro communication between the cell phone and server, which in our case, becomes the sensor and the server.

Consider the following flow diagram that indicates how the application components will interact with each other.

.

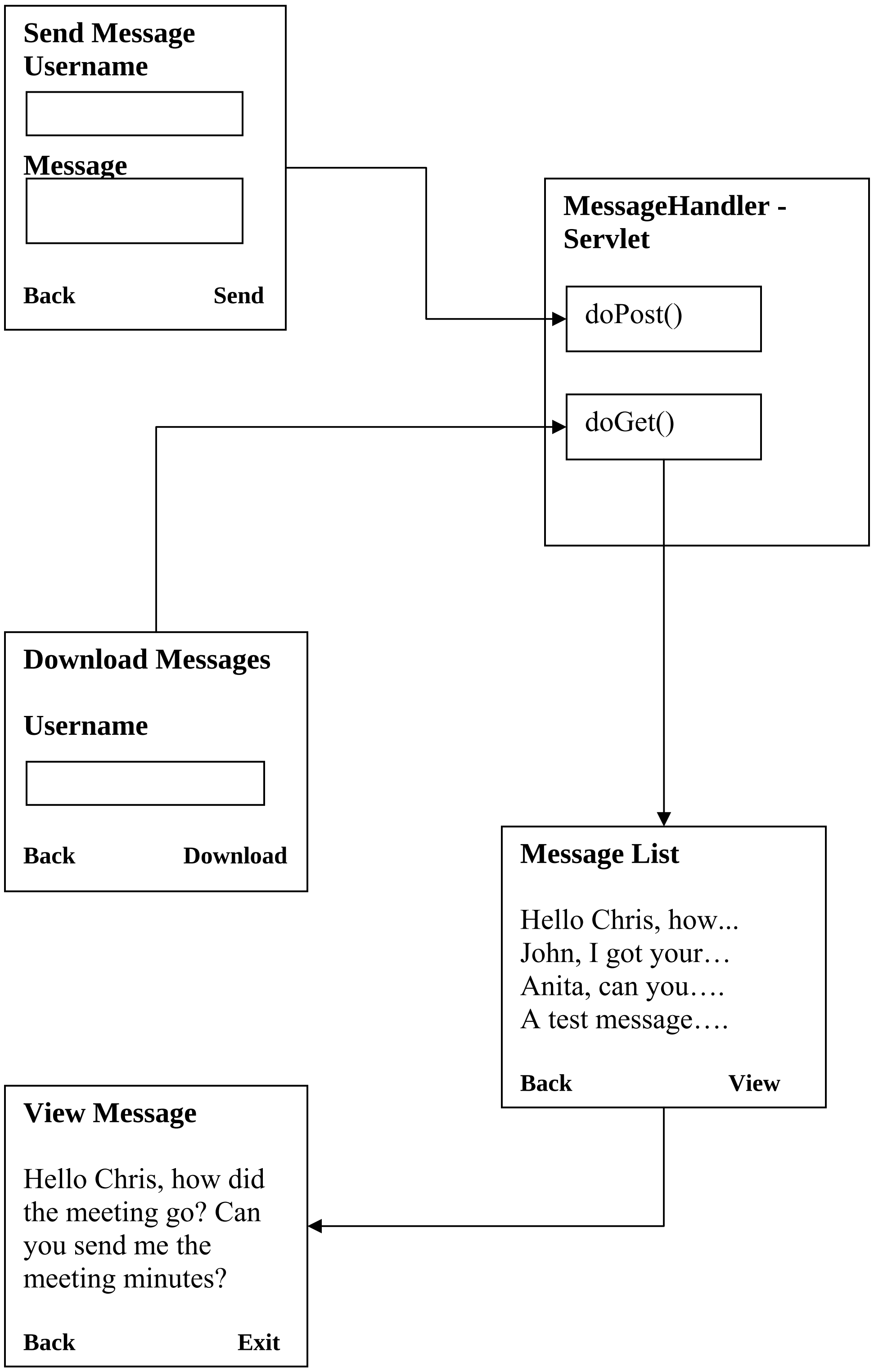


Figure 11.1: Flow Diagram of J2ME application

There are two components that interact with each other.

- J2ME client

  It is the application running on the cell phone (or in our case, the sensors). All the boxes above, except the Message Handler – Servlet, represent the various screens/UI that are a part of the J2ME client.

  - **Send Message**:

    This captures user's input in the form of a username and a text message. The username can be 15 chars long and the message can be maximum 255 characters long. This Midlet sends an 'http' POST message to a URL on which the MessageHandler servlet is configured.

  - **Download Messages**:

    This downloads messages sent by a given user. It sends an 'http' GET request to the MessageHandler servlet. The servlet responds by sending back a list of messages for that given username. The messages retrieved are shown in the next screen that is shown as Message List in figure 12.1.

- **View Message**:

  It is a screen that shows the complete message. Note that Message List only shows partial message (may be first 20 characters or so). The user can then select a particular message in MessageList screen and then click on View to see the actual message in View Message screen.

- MessageHandler Servlet

  It is responsible for processing the J2ME client requests that come in shape of **doGet** for retrieving all messages for a given user and a **doPost** request for saving a message.

Save Message

Saves a text message received for a given user. The table structure to save messages in is given below.

```
CREATE TABLE MESSAGES(
        username   varchar(15),
        message    varchar(50),
        dateReceived datetime)
```

Download Messages

For this request the servlet queries the database for all messages for a given username and send them back in ‘http’ Response.

- showAllMessages.jsp (JSP Page to view all messages)

  This is a simple but elegant JSP page that queries and displays all the messages in the Messages table. The messages are sorted by Date in a descending order.

The Database Setup

The database used in this application is a **Pointbase** Database.

A table is created with the following structure:

```
CREATE TABLE MESSAGES(
        username   varchar(50),
        message    varchar(255)
)
```

The Web Application

The web application consists of servlets, deployment descriptors, JSP pages, JDBC drivers for Pointbase database, XML files, etc.

The application consists of the following components:

- **MessageProcessor.java**: A Servlet that handles the SendMessage and RetrieveMessage requests from the J2ME client
- **PersistenceException.java**: An exception that is used by MessageProcessor.java internally to indicate database related errors, if any
- **Web.xml**: Deployment Descriptor for the web application
- **showAllMessages.jsp**: A JSP page that displays all the messages in the table

- **pbclient42RE.jar**: JDBC Driver for Pointbase database
- **build.xml**: Ant based build file to build the server project
- **MessengerServer.war**: Ready to deploy web application containing Servlet and JSP pages mentioned above.

Chapter 12

Potential Applications

One major potential application is using the J2ME application, with the protocol, along with Oracle Spatial database and MapViewer to create an application that conveys precise traffic information to the vehicle GPS systems.

The infrastructure that can be envisioned for using the real time traffic information collected by the sensors and wirelessly transmitted to the central computer or server is as follows:

- The waypoints from the United States Geological Survey (USGS) database are entered in the Oracle Spatial database
- The waypoints for the location of the sensors is entered in the database
- As the vehicle carrying the GPS navigation unit transmits its location to the GPS satellites, the information is relayed to the central computer wirelessly.
- The sensors transmit the information to the central computer wirelessly
- This information, along with the information from the GPS system and the waypoints stored in the database, can be used to calculate the vehicle location and the traffic information for the waypoints on the route the vehicle is traveling, using complex SQL (Structured Query Language) statements
- This information can be relayed wirelessly to the GPS navigation unit in the vehicle (54)

The key components to this infrastructure are Oracle Spatial database and Oracle Map Viewer.

Oracle Spatial is an integrated set of functions and procedures that enable spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle9*i* database. The location features in Oracle Database provide a platform that supports a wide range of applications – from automated mapping/facilities management and geographic information systems (GIS), to wireless location services and location-enabled e-business. (52)
Map Viewer is a programmable tool for visualizing spatial data managed by Oracle Spatial. It is basically a Java Servlet that runs inside a J2EE container, in this case, Oracle's Application Server that awaits user's map request through HTTP, and sends back a map response after processing and generating a map. The maps generated by Map Viewer are highly customizable, with all the mapping metadata such as map symbols and styling rules stored in the database and managed through a graphical user interface. Map Viewer works directly with Oracle Spatial through Java JDBC. (55)

With GPS, we can locate the coordinates of the vehicles, and these coordinates can be transmitted to the central server. Similarly, the coordinates of the traffic sensors are also recorded in a database table. The traffic sensors also transmit traffic data, say for example, traffic volume, along with the timestamp to the database.

These coordinates can then be matched with the coordinates of the planned route.

Consider the following figure:

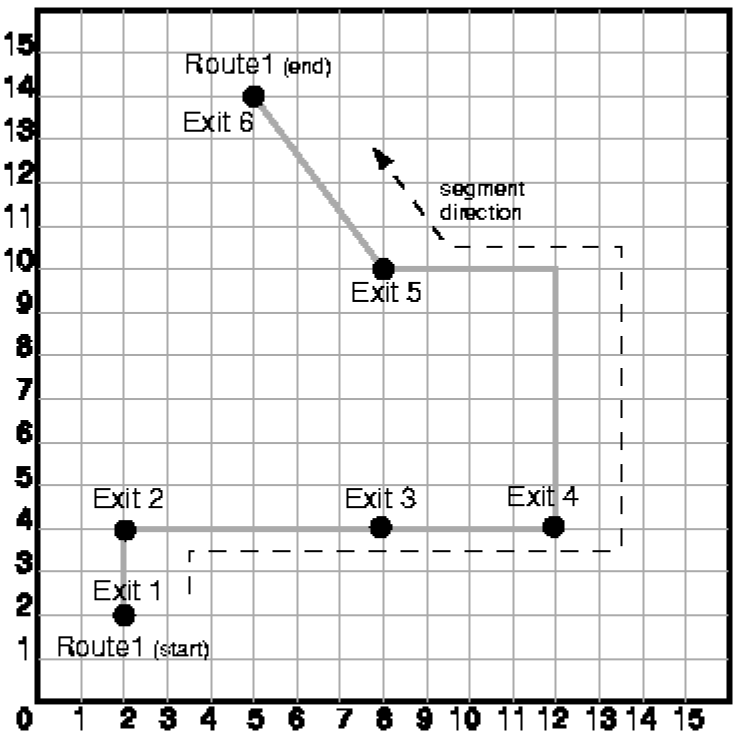


Figure 12.1: Coordinates of a vehicle along a route (53)

Assume that the car follows the route as shown. Also assume that the traffic sensors are placed along Exits 2, 3, 4,5 and 6. When the route of the car is planned, the coordinates of the route are inserted by the SQL Query mentioned above. The sensors transmit traffic data to the database.

When the car traverses the planned route, its location gets updated in the database in real-time and then using another Query that joins both the car location tables and the sensor tables, the traffic volume for the sensor nearest to the car is calculated.

This data is then transmitted back to the car where it is updated on the GPS screen.

The SQL for creating the table is:

```
TABLE sensor_locations (sensor_id           number(10),
                    Time            timestamp(3),
                    Traffic_Volume       number(10),
                    Location        SDO_GEOMETRY)
```

```
INSERT INTO sensor_locations VALUES(abcd,
      to_timestamp('11-JUN-2004 14:01:13.00', ' dd-mon-yyyy hh24:mi:ss.ff'),
      30,
      mdsys.sdo_geometry(2001, 8265,
                              mdsys.sdo_point_type(-140.2491, 32.8835, null),
                                    null, null)
```

```
SELECT  sdo_geom.relate(a.location, 'ANYINTERACT', b.route,0.5)
        From car_locations a, car_planned_routes b
        Where a.car_id = b.car_id and a.time = (select max(time) from car_locations
        Where car_id = 1234) and a.car_id = 1234
```

The final query will look like:

```
Select b.traffic_info, b.timestamp

From car_locations a, sensor_location B

Where b.location = [value]
```

Chapter 13

Conclusion and Recommendations

The Modified Wireless Token Ring Protocol is a robust protocol, having inherited all the robustness from the parent protocol, the Wireless Token Ring Protocol, and the modifications that help to extend the reach of the protocol on the field.

The ad-hoc nature of the sensor network in the field makes Modified WTRP an ideal candidate, over IEEE 802.11, in which all slave nodes have to report to a master. This approach makes it vulnerable to a single node failure at the master.

The protocol maintains a cache of the connectivity tables for each node. This cache is of tremendous importance, when there is a node failure, and a fast recovery is top priority. The protocol refers to the cached connectivity table, to quickly figure out the Source and the Destination addresses, thus making it robust against a single or even multiple node failures.

Partial connectivity is effectively handled by this protocol, inherited by its parent, WTRP. Unlike in IEEE 802.11, where the network needs to be fully connected; each station in Modified WTRP only need to connect with two stations that are one hop ahead and one hop behind. With the bi-directional nature, the coverage area of the network increases considerably, without having to increase the transmission range of the node.

It also puts the transmissions in order. The stations in the ring do not interfere with the station that is transmitting, and the stations that are outside the ring, have to wait for an invitation to join the ring, causing a significant decrease in the collision probability compared to IEEE 802.11.

The Maximum Token Rotation Time of the protocol is bounded. Hence, the next time the node gets to transmit is pre-fixed. This decreases the delay, hence improving the Quality of Service (QoS). This also ensures that a new token is generated after a pre-fixed time. Hence, if the token is taking unusually long to complete one rotation, a new token gets generated, based on the idle timer on the generating node.

As mentioned in the thesis, with WTRP being developed for a platoon of vehicles traveling on the road, it will be easier to port the protocol to build an application where the sensors can also join the ring between the vehicles and transmit data directly to the vehicles. With the modified protocol, the vehicle could join as a node, receive and transmit data and leave the ring, within a fraction of seconds, without any interference or breakdown of the ring.

Thus the Modified Wireless Token Ring Protocol is an ideal choice for this problem of data collision prevention on the field.

References

(1) Rishi Sood, Principal Analyst, Gartner Dataquest, Stamford, Conn.

(2) Mark P. Gardner, Highway Traffic Monitoring, http://gulliver.trb.org/publications/millennium/00052.pdf - Accessed Oct'2003

(3) Duke Lee, Roberto Attias, Anuj Puri, Raja Sengupta, Stavros Tripakis, Pravin Varaiya, *Wireless Token Ring Protocol for Intelligent Transportation Systems,* SCI Orlando, July 2002: p1 - 3

(4) Mustafa Ergen, Duke Lee, Raja Sengupta, Pravin Varaiya, *Wireless Token Ring Protocol, Performance Comparison with IEEE 802.11*, IEEE ISCC 2003: p1 - 4

(5) Mustafa Ergen, *WTRP – Wireless Token Ring Protocol,* Master's Thesis at University of California at Berkeley, 2002

(6) *International Standard ISO IEC8802-4:1990*—ANSI/IEEE Std. 802.4-1990

(7) http://www.itsa.org

(8) Armand J. Ciccarelli, An Analysis of the Impact of Wireless Technology on Public Vs. Private Traffic Data Collection, Dissemination and Use, MIT, February 2001

(9) Cheung, Coleri, Dundar, et al, Traffic Measurement and vehicle classification with a single magnetic sensor, Traffic Measurement Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 2004

(10) http://www.businessweek.com - Accessed Oct'2004

(11) WAP for Java Developers: Develop WAP Applications with Servlets and JavaServer Pages

(12) Ken Noblitt, A Comparison of Bluetooth and IEEE 802.11,

(13) www.bluetooth.com - Accessed Dec'2003

(14) J. Bray and C.F. Sturman, "Bluetooth: Connect Without Cables", Prentice Hall

(15) 3G Americas: Unifying the Americas through Wireless Technology, http://www.3gamericas.org/ - Accessed – Dec'2003

(16) http://developers.sun.com/techtopics/mobility/getstart/articles/radio/

(17) Michael Juntao Yuan, *Enterprise J2ME: Developing Mobile Java Applications*. Prentice Hall PTR, 2004

(18) Java Technology, http://java.sun.com - Accessed January '04

(19) Jonathan Knudsen, *Wireless Java: Developing with J2ME, 2nd ed*. Apress, 2003

(20) A. Goldsmith and S. Wicker, *Design challenges for energy-constrained ad hoc wireless networks*, IEEE Wireless Communications Magazine, Aug.2002.

(21) Chalermek Intanagonwiwat, *Directed Diffusion: An Application-Specific and Data-Centric Communication Paradigm for Wireless Sensor Networks,* University of Southern California, December 2002

(22) Dragan Petrović, Rahul C. Shah, Kannan Ramchandran, Jan Rabaey, *Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks***,** 2003

(23) Y.C.Tae, *Collision-Minimizing CSMA and its Applications to Wireless Sensor Networks*, Department of Computer Science, National University of Singapore

(24) *CDMA Development Group*, http://www.cdg.org/ - Accessed October '04

(25) *Qualcomm CDMA Technologies*, http://www.cdmatech.com/ - Accessed October '04

(26) *International Engineering Consortium*, http://www.iec.org/online/tutorials/tdma/ - Accessed October '04

(27) *Nortel Networks*, http://www.nortelnetworks.com/products/wireless/tdma.html - Accessed October '04

(28) *Oracle Technology Network*, http://otn.oracle.com - Accessed October '04

(29) *Oracle Spatial User's Guide and Reference*, http://download-west.oracle.com/docs/html/A96630_01/toc.htm - Accessed October '04

(30) Liujian Qian, *Developing Spatial Application using Oracle Spatial and MapViewer: An Oracle Technical Whitepaper*, Oracle Corporation, 2004

(31) Andrew Tanenbaum, *Computer Networks*, Prentice Hall, New Jersey, 1998

(32) Berkeley Web Over Wireless Group *http://wow.eecs.berkeley.edu/* - Accessed April '05

(33) Duke Lee, Mustafa Ergen, Anuj Puri *Socket Interface for User Applications in WOW project*, 2002

(34) *Spread Spectrum Communications Handbook, Revised Edition,* M.K. Simon, J.K Omura, McGraw Hill, 1994

(35) Mustafa Ergen, Duke Lee, Raja Sengupta, Pravin Varaiya, *"WTRP-Wireless Token Ring Protocol,"* published on IEEE Transaction on Vehicular Technology, 2003: p1-13

(36) I. F. Akyildiz, J. McNair, C. L. Martorell, R. Puigjaner, Y. Yesha, *Medium access control protocols for multimedia traffic in wireless networks, IEEE Network, vol.13, (no.4),* IEEE, July-Aug. 1999

(37) http://www.itsa.org – Accessed Oct'2003

(38) Peter Rysavy, Data Capabilities: GPRS to HSDPA, September 2004

(39) Bureau of Transportation Statistics – http://www.bts.gov/ - Accessed Dec'2003

(40) Scott B. Guthery, Mary J. Cronin, *Mobile Application Development with SMS and the SIM Toolkit.* McGraw-Hill Telecom, 2003

(41) The Mobile Information Device Profile (MIDP) http://java.sun.com/products/midp/ - Accessed February'04

(42) John W. Muchow*, Core J2ME Technology and MIDP*. Prentice Hall PTR,

(43) Vivek Arora, Narin Suphasindhu, John S. Baras, Douglas Dillon, *Effective Extensions of Internet in Hybrid Satellite-Terrestrial Networks*, University of Maryland at College Park & Hughes Network Systems, Inc., 1996

(44) Dennis Roddy, *Satellite Communications*, McGraw Hill Text, 1995

(45) Tom Logsdon, *Mobile Communication Satellites*, McGraw Hill Text, February 1995

(46) Litva, J. (John), *Digital beam-forming in wireless communications*, Boston : Artech House, c1996.

(47) F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, and D. Leask, *Piconet: Embedded Mobile Networking. IEEE Personal Communications*, October 1997.

(48) *The Bluetooth Special Interest Group*, http://www.bluetooth.com - Accessed July 2004

(49) Jan M. Rabaey et al, *PicoRadio supports ad hoc ultra-low power wireless networking*, , IEEE Computer, July 2000,

(50) E. Royer, C. K. Toh, *A review of current routing protocols for ad hoc mobile wireless networks*, IEEE Personal Comm., April 1999

(51) Zohar Naor, *An Efficient Short Messages Transmission in Cellular Networks*, University of Haifa at Oranim, Israel

(52) *CDMA Tutorial*, http://www.umtsworld.com/technology/cdmabasics.htm - Accessed October '04

(53) *CDMA vs. TDMA*, http://www.arcx.com/sites/CDMAvsTDMA.htm - Accessed October '04

(54) *Performing Location-Based Analysis*, http://www.oracle.com/technology/obe/obe10gdb/content/spatial/spatial.htm - Accessed October '04

(55) *The International GPS Global Positioning System Waypoint Registry*, http://www.waypoint.org/gps1-wayp.html - Accessed October '04

(56) How Stuff Works, *http://www.howstuffworks.com* - Accessed April '05

(57) Webopedia, The Online Computer Dictionary, *http://www.webopedia.com* - Accessed April ‘05

(58) Wi-Fi Planet- The Source for Wi-Fi Business and Technology, *http://www.wi-fiplanet.com* - Accessed May ‘05

(59) TechWeb: The Business Technology Network, *www.techweb.com* - Accessed May ‘05

Appendix

The following pages contain the code for the components of the application

MessageProcessor.java

```
package com.nlogic.messenger.server;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.DriverManager;


public class MessageProcessor extends HttpServlet{
   private final static String INSERT_MSG =
                "INSERT INTO messages (username,message)  values(?,?) ";
   private final static String QUERY_MSGS =
                "SELECT message from messages WHERE username = ?";
   private static String dbUrl,User,Pass;


   public void init(ServletConfig sc) throws ServletException{
    super.init(sc);
    try{
        Class.forName("com.pointbase.jdbc.jdbcUniversalDriver").newInstance();
        dbUrl=sc.getInitParameter("jdbcURL");
        User=sc.getInitParameter("username");
        Pass=sc.getInitParameter("password");


    }catch(Exception cnfe){
        throw new RuntimeException(
                "Cannot find suitable Driver for Pointbase Database");
    }
}


public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException{
    String name = request.getParameter("name");
    String message = request.getParameter("message");
    try{
        saveMessage(name,message);
        response.setStatus(HttpServletResponse.SC_OK);
    }catch(PersistenceException pe){
        getServletContext().log("Could not save the message due to error " +
                        pe.getMessage());
```

```
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        }
        PrintWriter writer = response.getWriter();
        writer.flush();
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException{
        String name = request.getParameter("name");
        List msgs = retreiveMessages(name);
        StringBuffer sb = new StringBuffer();
        for(int i=0;i<msgs.size();i++){
            sb.append("<msg>");
            String message = ((String)msgs.get(i)).trim();
            sb.append(message);
            sb.append("</msg>");
        }
        String msgStr = sb.toString();
        getServletContext().log("Sending messages " + msgStr);
        response.setStatus(HttpServletResponse.SC_OK);
        response.setContentLength(msgStr.getBytes().length);
        PrintWriter writer = response.getWriter();
        writer.write(sb.toString());
```

```
}



private void saveMessage(String name,String message) throws
   PersistenceException {
      Connection c = null;
      PreparedStatement ps = null;
      try{
      c = getConnection();
      ps = c.prepareStatement(INSERT_MSG);
      ps.setString(1,name);
      ps.setString(2,message);
      ps.executeUpdate();
      }catch(SQLException sqe){
      sqe.printStackTrace();
      throw new PersistenceException(
            "Got SQL Error " + sqe.getMessage());
      }finally{
      try{
         if(ps != null) ps.close();
         if(c!=null)c.close();
      }catch(Exception e){}
      }
```

```
}

private List retreiveMessages(String name){
    List results = new ArrayList();
    Connection c = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    try{
    c = getConnection();
    ps = c.prepareStatement(QUERY_MSGS);
    ps.setString(1,name);
    rs = ps.executeQuery();
    while(rs.next()){
        String message = rs.getString("message");
        results.add(message);
    }
    }catch(SQLException sqe){
    sqe.printStackTrace();
    throw new RuntimeException("Got SQL Error " + sqe.getMessage());
    }finally{
    try{
        if(ps != null) ps.close();
        if(c!=null)c.close();
        }catch(Exception e){}

        }

        return results;

    }


    private Connection getConnection()throws SQLException {

        Connection c = DriverManager.getConnection(dbUrl,User,Pass);

        return c;

    }

}
```

showAllMessages.jsp

```
<%@page
import="java.sql.*,java.sql.Connection,java.sql.PreparedStatement,java.sql.SQLExceptio
n,java.sql.ResultSet,java.sql.DriverManager" %>
<html>
<link href="style.css" rel="stylesheet" type="text/css">
<body>
<h2>
List Of Messages
</h2>
<table width=700 height=234 bordercolor="#2262B7" class="border" border="1">
<th bgcolor=#2262b7 class="style3">Username</th>
<th bgcolor=#2262b7 class="style3">Message</th>

<%
String dbUrl,User,Pass;//="dbc:pointbase:server://localhost/sample";
Class.forName("com.pointbase.jdbc.jdbcUniversalDriver").newInstance();

dbUrl="jdbc:pointbase:server://localhost/sample";
User="pbpublic";
Pass="pbpublic";
Connection c = DriverManager.getConnection(dbUrl,User,Pass);
```

```
Statement stmt=c.createStatement();

ResultSet rs;

          try{

                    rs = stmt.executeQuery("SELECT * from messages");

                    while(rs.next())

                    {

                              String UserName = rs.getString(1);

                              String Message=rs.getString(2);


                              out.print("<tr><td class='style1'>"+UserName+"</td>");

                              out.print("<td class='style1'>"+Message +"</td></tr>");

                    }

                              out.print("</table>");

            }

          catch(SQLException sqe)

          {

                    sqe.printStackTrace();

                    throw new RuntimeException("Got SQL Error " + sqe.getMessage());

          }

                    stmt.close();

                    c.close();


%>
```

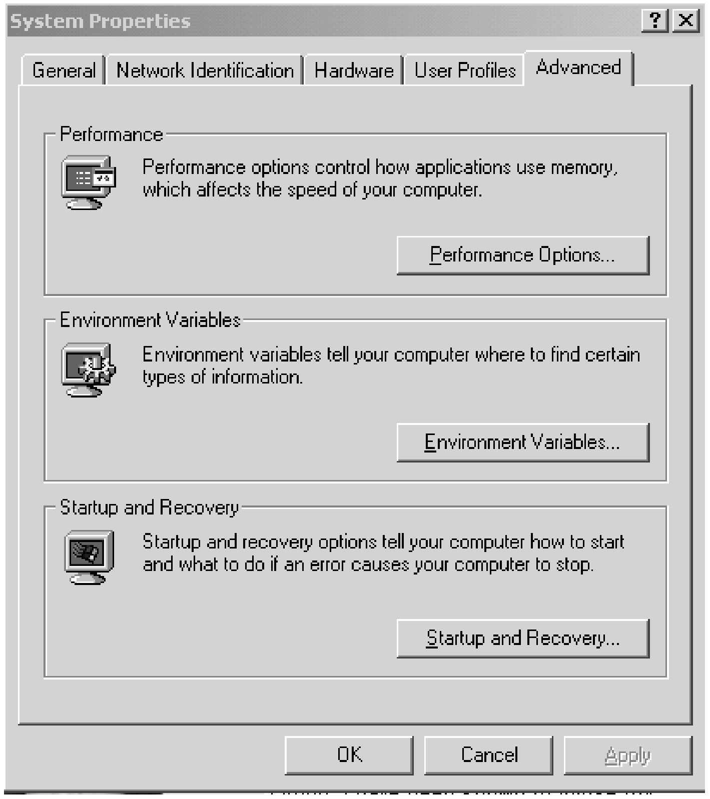
Steps to build the application:

1. ANT tool is needed to build the .war file for deployment on a J2EE container. The tool can be downloaded from this URL:

   http://mirror.candidhosting.com/apache/ant/binaries/apache-ant-1.6.2-bin.zip

2. Unzip the ant zip file and add the bin folder to your PATH variable so that the ANT bat script can be executed without specifying a full path to it.

   To add to the PATH variable, right-click on **My Computer** and click on **Properties**

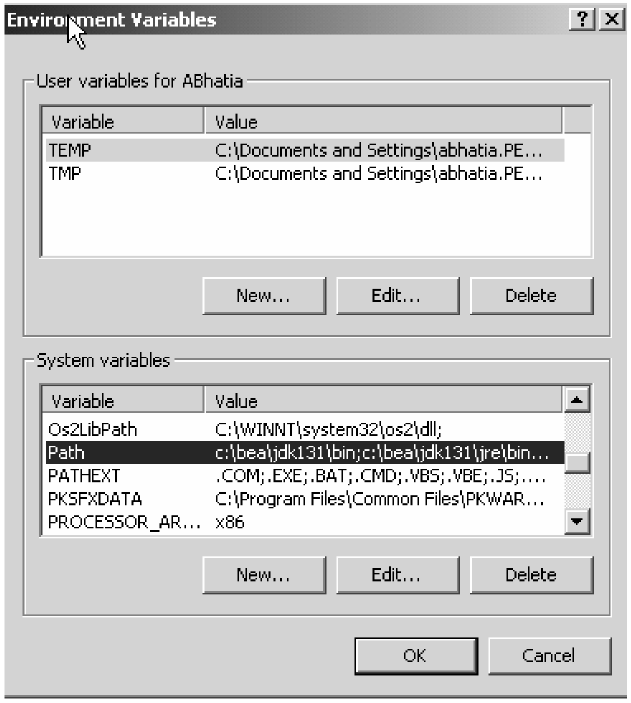


Performance options control how applications use memory, which affects the speed of your computer.

Environment variables tell your computer where to find certain types of information.

Startup and recovery options tell your computer how to start and what to do if an error causes your computer to stop.

Click on **Environment Variables**



Variable	Value
TEMP	C:\Documents and Settings\abhatia.PE...
TMP	C:\Documents and Settings\abhatia.PE...

Variable	Value
Os2LibPath	C:\WINNT\system32\os2\dll;
Path	c:\bea\jdk131\bin;c:\bea\jdk131\jre\bin...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PKSFXDATA	C:\Program Files\Common Files\PKWAR...
PROCESSOR_AR...	x86

Select the Path variable in **System Variables** and click on **Edit**. You should be able to add the bin folder for ANT tool here

3. Make sure the JAVA_HOME environment variable is set to a JDK 1.4 installation folder
4. Extract server.zip to your local hard drive. Make sure "Use Folder Names" option is checked when you extract. The following directory structure will be created on the file system:

/server
- **build.xml --- ant build file**
- **MessengerServer.jpx --- Jbuilder 9 project file.**
- **MessengerServer.War**
- */docroot*
  - **showAllMessages.jsp**

```
                /WEB-INF
                        web.xml
                        /lib
                             pbclient42RE.jar
        /src
            /com
                 /nlogic
                        /messenger
                                   /server
                                        MessageProcessor.java
                                        PersistenceException.java
        /extlib
              servlet.jar
```

5. Edit **web.xml** to change ***jdbcURL*** parameter to point it to the host where pointbase database server is running and the database with *Messages* table
6. Open a DOS shell and cd into the server folder and type ANT to invoke the build. This build should create a new updated MessengerServer.war file ready to be deployed to any Servlet container including SunOne application server

Deploying the Application

- Deploy the MessengerServer.war file to a J2EE container.
- Each web application in J2EE world runs under a **context** which is a logical name that is referred in the URLs to access that web application. By default, the J2ME client is setup to talk to the URL *http://localhost:8080/Messenger/MessageProcessor* where Messenger is the context under which the Web Application is deployed. If you plan to deploy it under a different context name, which is not recommended, you will need to

modify the Messenger.jad (J2ME application descriptor) to run the J2ME client

The J2ME Client

The J2ME client consists of JBuilder project file, Jar files, Application Descriptors, Build file and a tool to build and run the J2ME application in an emulator

The J2ME client consists of the following components:

- **/src** – holds all the .java files that form the J2ME client
- **Messenger.jpx** – JBuilder project file
- **MessengerClient.jar** – Jar file holding all the class files ready to run
- **Messenger.jad** – Application Descriptor
- **antenna-bin-0.9.12** – tool to build and run J2ME applications in an emulator

Steps to run the client

1. Extract the J2MEClient.zip file to your local file system
2. Open build.xml file and change the value of wtk.home property by modifying the following line in build.xml

   <property name="wtk.home" value="D:\JBuilder9\j2mewtk2.0"/>

   The property should point to the folder where J2ME toolkit is installed

3. Make sure the J2EE container is up and running with the web application correctly deployed. Also make sure that Pointbase database server is running as well
4. Open a DOS shell and change directory to the **Messenger** folder that gets created when you extract the zip file.
5. Type ***ant run*** to invoke the run ANT target that will launch the emulator to run the J2ME client.

*Now you are ready to send messages by entering username and the text message. You can also download the mressages that you sent by using **Retrieve Messages** option*

Viewing all the Messages in the Database through the webpage

You can view all the messages stored in the database by going to the URL
http://localhost:8080/MessengerServer/showAllMessages.jsp