PROJECT REPORT ON

IMPLEMENTATION OF WIRED CUM WIRELESS NETWORK IN NS2 SIMULATOR

Report submitted to the SASTRA Deemed to be University as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

123015117- VISVESWARAN B – IT

February 2022

SCHOOL OF COMPUTING
SASTRA DEEMED TO BE UNIVERSITY
THANJAVUR, TAMIL NADU, INDIA – 613 401

TABLE OF CONTENTS

SCHOOL OF COMPUTING
SASTRA DEEMED TO BE UNIVERSITY
THANJAVUR, TAMIL NADU, INDIA – 613 401

BONAFIDE CERTIFICATE

This is to certify that the report titled **"IMPLEMENTATION OF WIRED CUM WIRELESS NETWORK IN NS2 SIMULATOR"** submitted as a requirement for the course**, CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by VISVESWARAN B (123015117, IT) during the academic year 2021-22, in the School of Computing

Project Based Work Viva voce held on _____ .

Examiner 1 **Examiner 2**

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CBR	Constant Bit Rate
CN	Corresponding Node
DARPA	Defense Advanced Research Projects Agency
DSDV	Destination-Sequenced Distance Vector
FA	Foreign Agent
FTP	File Transfer Protocol
HA	Home Agent
ICMP	Internet Control Message Protocol
ICSI	International Computer Science Institute
IETF	Internet Engineering Task Force
IP	Internet Protocol
LBL	Lawrence Berkeley National Laboratory
MAC	Media Access Control ‘
MH	Mobile Host
MIP	Mobile Internet Protocol
MN	Mobile Node
NAM	Network Animator
NS2	Network Simulation Version 2
NS3	Network Simulation Version 3
OTCL	Object Oriented Extension of TCL
PARC	Palo Alto Research Center
RED	Random Early Detection
RFC	Request For Comments
RH	Remote Host
TCL	Tool Command Language
TCLCL	TCL with Classes

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USI	University of Southern California
VINT	Virtual Inter Network Testbed

NOTATIONS

d	Distance
G_t	Antenna gain of the transmitter
G_r	Antenna gain of the receiver
h_t	Height of the transmitter
h_r	Height of the receiver
P_t	Transmitted Signal Power
P_r	Receiving Threshold
L	System Loss
λ	Wavelength
Σ	Summation
Π	PI

ABSTRACT

Wi-Fi has become the most popular/common wireless technology that allows number of electronics devices to exchange data over a computer network. This includes high speed internet connection resulting in a very fast and convenient way to stay connected. Wi-Fi is basically known as Wireless Local Area Network (WLAN), which uses a certain internet protocol known as Institute of Electrical and Electronics Engineers (IEEE 802.11 standards). These days, Wi-Fi is used from personal computers to smartphones to even printers as well as numerous wireless devices and all these can be connected through a single wireless (Wi-Fi) Network. Packet loss is one of the biggest issues affecting throughput in wireless networks and since wireless devices can't transmit packet over larger distance there is need of wired connection to pass the packets of wireless node of one location to other location. For achieving the above one, we need a device which can communicate with wired as well as wireless network and that is referred a access point or base station node.

Also Mobile Internet Protocol, or Mobile IP (MIP), is an Internet Engineering Task Force (IETF) protocol that is used to maintain a consistent internet protocol (IP) address while moving through a number of networks. This is done by switching networks overlapping networks once the device has gone out of range of one network. The type of network that will be simulated is the common Wireless Local Access Network (WLAN).

So task of the project is to design a network that will transfer packets from a wireless device to wired connection and also to demonstrate how MIP can be used to maintain a connection when moving through several locations in the network.

CHAPTER 1

INTRODUCTION TO NS2 SIMULATOR

The effectiveness of any protocol can be appreciated only when statistical data about the performance of the protocol is available. Before evaluating the performance of protocol, we need a clear understanding of protocol; to achieve this we need a clear visualization and one of the possible ways to visualize these protocols through computer animation. In order to collect the required statistical data and data for viewing the protocol via animation, a simulation of the proposed protocol was performed and various scenarios of operation were analysed. Simulation provides for repeatable, controlled experimentation with only a modest overhead required to construct and carry out simulations. This chapter provides a quick overview of the implementation tool. In order to carry out tests on the designed system, we have to implement it on a suitable language and platform.

1.1 NETWORK SIMULATOR-2

In this project, the network is designed and simulated using Network Simulator (Version 2), widely known as NS2. It is simply an event-driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions can be done using NS2 and it also supports transport layer protocols such TCP and UDP. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours. Also NS2 supports standard MIP, provided by CMU Monarch's model.

NS2 began as a revision of NS1. From 1997 to 2000, NS development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. NS-2 is an open-source program that runs best in a Linux machine as it is developed in the UNIX environment; however, it is also possible to run on more popular operating systems such as windows and mac with help some from developmental tools available for those platforms. In order to run NS-2 on windows, a program named Cygwin is needed. In terms in running NS-2 in mac, the program X-Code is needed. Currently Network Simulator-3 (NS3) is developed and maintained

1.2 Basic Architecture of NS2

NS2 is an Object-oriented Tool Command Language (OTCL) script interpreter that has a simulation event scheduler, network component object libraries and network setup. NS2 uses an object oriented language that is used to simulate different network protocols. Its backend is written in C++ and its front end is OTcl.

With OTcl it is possible to configure different network parameters for a system. NS software promotes extensions by users. It provides a rich infrastructure for developing new protocols. Also, instead of using a single programming language that defines a monolithic simulation, NS uses the split-programming model in which the implementation of the model is distributed between two languages. The goal is to provide adequate flexibility without losing performance. In particular, tasks such as low level event processing or packet forwarding through simulated router require high performance and are not modified frequently once put into place. Hence, they can be best implemented in compiled language like C++ On the other hand; tasks such as the dynamic configuration of protocol objects and exploring a number of different scenarios undergo frequent changes as the simulation proceeds. Hence, they can be best implemented in a flexible and interactive scripting language like OTcl. Thus, C++ implements the core set of high performance primitives and the OTcl scripting language express the definition, configuration and control of the simulation

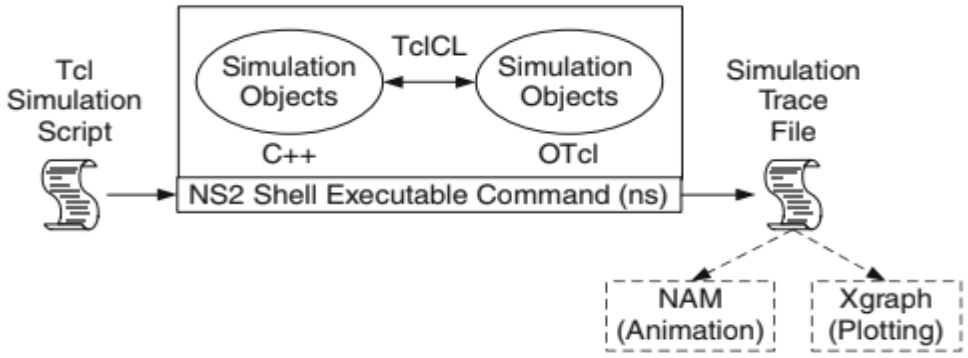

Figure 1.1 Basic architecture of NS

NS2 provides users with an executable command “ns” which takes one input argument, the name of a Tcl simulation scripting file which does the simulation. After simulation, NS2 outputs either text-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyse a particular behaviour of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

1.3. ASSISTANT TOOLS IN NS2

1.3.1 NAM FILE

Network AniMator (NAM) is a Tcl/Tk based animation tool for viewing network simulation traces and real world packet traces. It began at Lawrence Berkeley National Laboratory (LBL). It supports topology layout, packet level animation, and various data inspection tools. In Early 1990's NAM was as a tool for animating packet trace data. It has evolved vigorously over the past few years. The NAM development effort was an on-going collaboration with the Virtual Inter Network Testbed (VINT) project. The first step to use NAM is to produce the trace file. The trace file should contain topology information, e.g., nodes, links, as well as packet traces. Usually, one should specify simulator to produce NAM trace file via Tcl script. NAM Trace File is different for wired and wireless network .Hence you specify whether the network is wired or wireless to get NAM trace file accordingly. During an NS simulation, a user can produce topology configurations, layout information, and packet traces using tracing events in NS. When the trace file is generated, it is ready to be animated by NAM. Many visualization features are available in NAM. Some features available in NAM are animating coloured packet flows, dragging and dropping nodes (positioning), labelling nodes at a specified instant, shaping the nodes, colouring a specific link, and monitoring a queue.

1.3.2 TRACE FILES

Trace files are trace logs that have all information about the events that has occurred during the simulation. The information that is written to trace file must be specified in Tcl Script. The format of trace wireless is different for wired and wireless network. If you are implementing wired cum wireless network in NS2 then you will these two different formats under a single trace file. In this project, we will use older version of trace file format and not the new one. Hence older version of wired and wireless trace file format will be discussed here.

1.3.2.1 Wired Simulation Trace Format

An Example of Wired Trace format:

+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610

Trace file format of wired simulation is as shown in Figure 1.2

Figure 1.2 Trace Format of wired node

From Figure 1.2, it is obvious that wired simulation trace file format contains 12 fields

The short description of 12 fields as follows

1. " Event " field corresponds to four possible event types that a packet has experienced:
   'r' represent packet received
   '+' represent packet enqueued
   '-' represent packet dequeued
   'd' represent packet dropped
2. "Time" Field denotes the time at which such event occurred.
3. " From node " and " To node" Fields denotes the starting and the terminating nodes of the link
4. Fields 5 and 6 are "packet type" and "packet size", respectively.
5. The 7th field is a series of flags, indicating any abnormal behaviour.
6. 8th field represents the flow ID
7. Fields 9 and 10 mark the source and the destination addresses, respectively, in the form of " node.port "
8. 11th field specifies the packet sequence number.
9. 12th field denotes packet unique ID

1.3.2.2 Wireless Simulation Trace Format

An Example of Wireless Trace format:

s 2.237500000 _6_ AGT --- 129 cbr 1000 [0 0 0 0] ------- [4196353:2 12582914:2 32 0] [61] 0 0

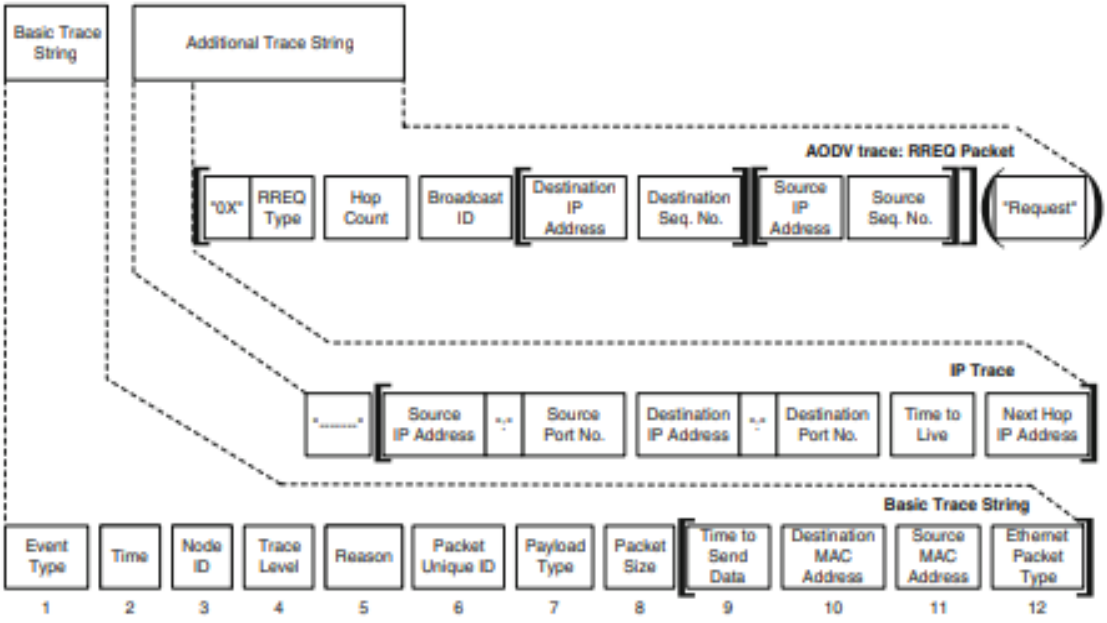

Figure 1.3 Old Trace Format of wireless node

Important Fields in a wireless trace record are shown below:

1. Event Type : field corresponds to four possible event types that a packet has experienced:
   - ‘r’ represent packet received
   - ‘s’ represent packet sent
   - ‘f’ represent packet forwarded
   - ‘d’ represent packet dropped

2. Trace level: The common levels are AGT for agent trace, RTR for routing trace, MAC for MAC trace.
3. Reason: The reason for this trace (e.g., “NRTE” for No Route Entry).
4. Time to Send Data: Expected duration required to transmit this packet over the wireless channel as indicated by the underlying MAC protocol.
5. Ethernet Packet Type: Currently, there are only two Ethernet packet types:

   – A general IP packet: The value is “ETHERTYPE_IP” defined as “0x0800.”

   – An ARP packet: The value is “ETHERTYPE_ARP” defined as “0x0860.”

6. RREQ Type: Type as indicated in the field “rq_type” of the hdr_aodv_ request struct data type. By default, the value is “AODVTYPE_RREQ” defined as “0x02.”

1.3.2.3 AWK Scripts for Analysing Trace Files

AWK is a general-purpose scripting language designed for advanced text processing. It is mostly used as a reporting and analysis tool and named after its three original author's name:

A: Alfred **A**ho

W: Peter **W**einberger

K: Brian **K**ernighan

AWK Scripts are very good in processing the data from the log (trace files) which we get from NS2. AWK program structure contains mainly three parts. They are Begin , Content , End. The Structure of AWK Scripts is as follows:

BEGIN { <initialization> }

{

<actionSet>

}

END { <final finalActionSet> }

BEGIN block in AWK Script deals with what to be executed prior to text file processing, normally which is used to initialize variable values or constants.

CONTENT block in AWK Script which processes the text file moves lines by lines and executes the <actionSet> if the current line match with the pattern. The action repeats until AWK reaches the end of the file.

END block in AWK Script This part explains what to be executed after the text file processing i.e. what to print on the terminal or to show output in terminal.

We can find out the Average Throughput, Average End to End delay and Packet Delivery Ratio by parsing the trace file using AWK Scripts

1.3.3. Xgraph and Gnuplot

Xgraph and Gnuplot are two plotter tools of NS2 used to show the results of the simulations. Xgraph is a general purpose x-y data plotter, the operation of which is using the first column as the X-axis data, and Y-axis is decided by the second column then plot the graph. Gnuplot is one kind of command-driven interactive function plotting program to generate two or three-dimensional plots of data. Nowadays, Gnuplot is widely used on UNIX, Linux and Windows platform.

But in this project python matplotlib library is used to plot the simulation throughput results

CHAPTER 2

PROBLEM STATEMENT

2.1 MOTIVATION

The mobile nodes mainly support simulation of multi-hop ad-hoc networks or wireless LANs. But what if we need to simulate a topology of multiple wireless LANs connected through wired nodes, or may need to run mobileIP on top of these wireless nodes?

2.2 PROBLEM STATEMENT

First main problem in facing the wired cum wireless scenario is the issue of routing. In NS2, routing information is generated based on the connectivity of the topology, i.e. how nodes are connected to one another through Links. Mobile nodes on the other hand have no concept of links. They route packets among themselves, within the wireless topology, using their routing protocol. So how packets would be exchanged between these two types of nodes in wired cum wireless network?

Next Problem in wired cum wireless scenario is that Mobile nodes due to its mobility property can move from one network location to another network location where each network has unique address. In order to get fit into the new network its old IP address is replaced by new IP address, as a result of it would not be able to inform the other network devices that its IP address is changed and other devices would not been able to communicate with that particular mobile node. Which protocol will help to solve the above situation?

In this project, we will try to figure out the solutions for above problem statements in forthcoming chapters.

CHAPTER 3

ROLE OF BASE STATION IN WIRED CUM WIRELESS NETWORK

In previous Chapter, the first issue stated in the problem statement section can be solved by introducing a new type of node called “base station node”. Here base station node plays the role of a gateway for the wired and wireless domains. The base station node is responsible for delivering packets into and out of the wireless domain. In order to achieve this we need Hierarchical routing. Each wireless domain along with its base-station would have a unique domain address assigned to them. All packets destined to a wireless node would reach the base-station attached to the domain of that wireless node, who would eventually hand the packet over to the destination (mobile node). And mobile nodes route packets, destined to outside their (wireless) domain, to their base-station node. The base-station knows how to forward these packets towards the (wired) destination.

One of the biggest advantages of using hierarchical routing is its scalability. It is thus suitable for very large networks. In flat routing, every node knows about every other node in the topology, thus resulting in routing table size to the order of n^2. In hierarchical routing, each node knows only about those nodes in its level. For all other destinations outside its level it forwards the packets to the border router of its level. Thus the routing table size gets downsized to the order of about log n. The routing state information can be aggregated to decrease the burden of the routing updates and storage.

In real world telecommunications, a base station is a fixed transceiver that is the main communication point for one or more wireless mobile client devices. A base station serves as a central connection point for a wireless device to communicate. It further connects the device to other networks or devices, usually through dedicated high bandwidth wire or fibre optic connections. Base stations are generally a transceiver, capable of sending and receiving wireless signals; otherwise, if they only transmitted signals out, they would be considered a transmitter or broadcast point. A base station will have one or more radio frequency (RF) antennas to transmit and receive RF signals to other devices.

CHAPTER 4

MOBILE INTERNET PROTOCOL

This chapter aims to explain a proposed solution for the second issue mention in chapter 2 under problem statement section i.e. why the mobility cannot be supported by the existing IP and presents a new protocol that can be used to overcome the weakness of IP addressing and routing scheme in supporting the mobile host

4.1 HOST MOBILITY PROBLEM WITH IP

The IP address is a 32-bit number that uniquely identifies a network interface on a machine. An IP address is typically written in decimal digits, formatted as four 8-bit fields separated by periods. Each 8-bit field represents a byte of the IP address. This form of representing the bytes of an IP address is often referred to as the dotted-decimal format. The bytes of the IP address are further classified into two parts: the network part and the host part. The original IP address was based on the assumption that a host part of network is stationary. Routers use the hierarchical structure of an IP address to route an IP datagram .The address is valid only when it is attached to the network. Suppose if the network changes, the address is no longer valid. Therefore IP addresses are designed to work with stationary hosts because part of the address defines the network to which the host is attached.

4.2 DEVOLPMENT OF MOBILE IP

Mobile IP solves a problem introduced by the fact that traditional IP addresses simultaneously represent the host's identity and encode the host's topological location on the IP network. Mobile IP provides IP level mobility to allow hosts to roam around different sub networks.

Mobile IP is an *IETF (Internet Engineering Task Force)* standard communications protocol designed to allow mobile devices (such as laptop, PDA, mobile phone, etc.) users to move from one network to another while maintaining their permanent IP (Internet Protocol) address.

Defined in RFC (Request for Comments) 2002, mobile IP is an enhancement of the internet protocol (IP) that adds mechanisms for forwarding internet traffic to mobile devices (known as mobile nodes) when they are connecting through other than their home network.

4.2.1 TERMINOLOGIES IN MOBILE IP

Most important Entities and Terminologies in Mobile IP are the following:

Mobile Node (MN) / Mobile Host (MH) **:** A mobile node is an internet connected entity that is contained within a wireless mobile station. A MN is built into a TCP/IP stack and it allows the wireless devices to connect internet or communicate with other MIP components in the network from a wide range of locations. MN devices are initially identified with the home network address, and it can roam from networks to another without changing its IP address. A MN can be any remote devices, such as laptops, cell phones or even a router.

Home Agent (HA) **:** A home agent is located in the home network of the MN, and it registered the MN with its own address. The HA captures the data packets and delivers to the MN. When the MN roam away from its home network and into a foreign network, HA identifies its care-of address and locates the MN's location. HA uses a method called tunneling to forward the packets to the foreign agent.

Foreign Agent (FA) **:** Foreign agents are residing in the foreign network. They intercepts the data packets from the HA and redistribute them to the MN. FAs send out beacon signals periodically. As the MN enters the foreign agent range and detects these beacons signals, the FA registers the MN with a care-of address. With the care-of address, the HA now forwards all data packets to the specified FA.

Correspondent Node (CN) **:** A correspondent node is another MIP component which a MN is communicating with. A CN may be another mobile or stationary device, or a host.

Remote Host : It is a host in a remote network.

Care-Of Address **:** This is an address assigned to mobile node by foreign network. It might be a foreign care-of address, which is an address of the foreign agent, or a co-located care-of address which is a temporary address assigned to the MN by the MN itself. Care-of Address allows the HA to indicate its current location and delivers the packets respectively. When the MN is registered with this temporary address, the MIP signals HA. Then the HA forwards the data packets to the foreign agent with this address.

Tunnel **:** The data packets get transported through a tunnel, where the packets get encapsulated. A tunnel is a connection between the home address and the care-of address. Tunneling is a mechanism used by the MIP to transfer any packets across the network through a tunnel.

4.2.2 WORKING OF MOBILE IP

Mobile IP communication protocol allows the mobile host to communicate with the remote host even if the mobile host is in the network other than its home network. If a mobile host wants to communicate with the remote host being itself in the foreign network, then it has to go through three phases described below:

4.2.2.1 AGENT DISCOVERY

Agent discovery is the first phase and requires the involvement of mobile host, home agent and a foreign agent. This phase also has two sub-phases as described below:

I. The mobile host has to discover the home agent's address before it moves away from the home network.
II. As the mobile host moves to a new network (foreign network), it has to discover the foreign agent's address and also the care-of address.

The agent discovery is done with the help of two types of messages: agent advertisement and agent solicitation.

AGENT ADVERTISEMENT

A simple router advertises its existence on the network with 'ICMP router advertisement' packet. If the router is playing the role of an agent, it 'appends' the agent advertisement message to the ICMP advertisement packet.

If the advertisement is done by the foreign agent, then it sends a 'list of addresses' available as care-of addresses, among which mobile host can choose the one. The announcement of the care-of address that has been chosen by the mobile host is done in the registration request.

AGENT SOLICITATION

In case, if a host in the network doesn't receive 'router ICMP advertisement' packet. The host can initiate itself by sending router ICMP solicitation packet. If a mobile host has not received the 'agent advertisement', it can use router ICMP solicitation packet to send the 'agent solicitation' message.

4.2.2.2 REGISTRATION

This is the second phase and it also requires the involvement of mobile host, home agent and a foreign agent. After discovering the home agent and foreign agent address the mobile host has to register itself to the foreign agent and the home agent.

The mobile host has to renew its registration if has been expired. While returning back to the home network the mobile host has to cancel or deregister its registration. This registration process involves two of messages: registration request and registration reply. This registration messages are first encapsulated in a UDP user datagram.

REGISTRATION REQUEST

The mobile host sends the registration request to foreign agent declaring its chosen 'care-of address' and along with this it also declares its 'home address' and 'home agent's address'. The foreign agent when receives the registration request message, it registers that mobile host and further forwards the message to the 'home agent'.

The message has been passed to the home agent in the IP packet. So, now the home agent knows the address of the foreign agent where its mobile host is present now because the IP address would have the foreign agents address in its source address field of IP packet.

REGISTRATION REPLY

The reply packet is sent from the 'home agent' to the 'mobile host' via a 'foreign agent'. This registration reply has the confirmation of whether the request is accepted or denied.

4.2.2.3 DATA TRANSFER

Now finally, after agent discovery and registration process, the mobile host being in the foreign network, can communicate with the remote host. Let us see how? Consider that the 'remote host' wants to send a data packet to the 'mobile host' unaware that the mobile host is not in its home network. The remote host would definitely send the packet with its own address in the source address field and mobile host's home address in the destination address field of the packet. As the mobile host is not it's the home network, the packet sent by the remote host is received by the home agent on behalf of the mobile host. The 'home agent' encapsulates the received IP datagram into another IP datagram (with home agent's address in the source address field and foreign agent address in the destination address field) and relay it to the foreign agent. Foreign agent receives the packet, remove the encapsulation and see the home address of the mobile host in the destination address field of the original packet sent by the remote host. The foreign agent reviews its registry table and sees that which care-of address has been registered to the corresponding home address and then forwards the packet corresponding mobile host. Next, if a mobile host being in the foreign network wants to reply or communicate with the remote host, it simply prepares a packet put its home address in the source address field of packet and remote hosts address in the destination address field of the packet. The mobile host then directly send the packet from the foreign network to the remote host.

CHAPTER 5

NETWORK LAYOUT AND OVERVIEW OF DESIGNED NETWORK

Since this project is all about implementing wired cum wireless network. Here we will be seeing three different types of node .They are wired node, Mobile node and Base station node. Base station node generally refers to access point which acts as gateway between wired and wireless nodes .In the designed network, two access points or base station nodes are directly connected to internet through wire. Another two access points are connected to router which connected to the internet through wire. Now mobile nodes or wireless nodes can communicate in the designed network by attaching it to access points. The general overall network layout designed in this project can be viewed in figure 5.1

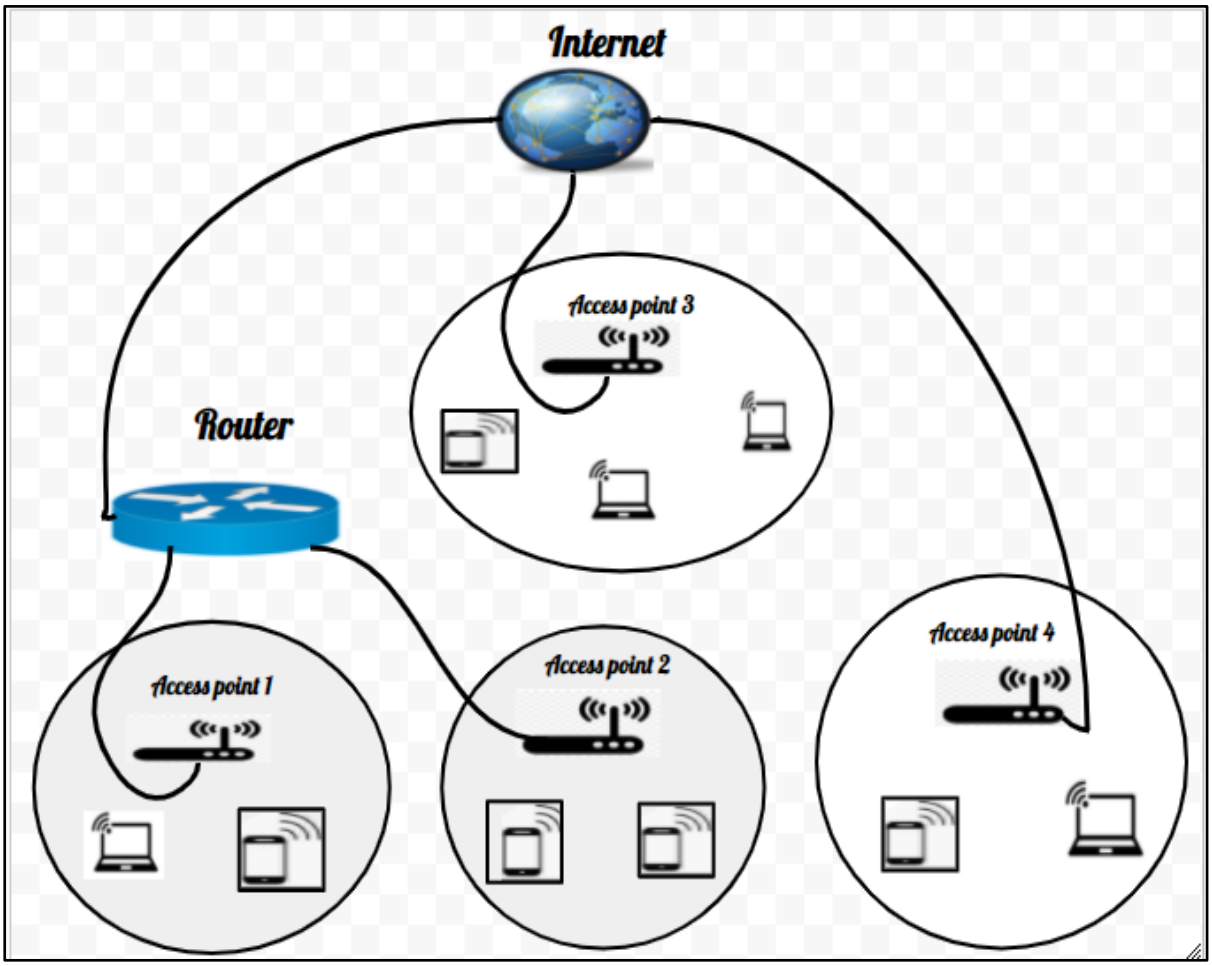

Figure 5.1 Layout of designed wired cum wireless network

Following describes the details of the designed wired cum wireless network

TYPE OF NODE	NUMBER OF NODES
Wired Nodes	2
Base Station Nodes (Access point)	4
Mobile Nodes	9

Table 5.1 Node type and its number of nodes in designed network

Details of wired connection designed using NS2 is as follows

SOURCE NODE	DESTINATION NODE	BANDWITH OF THE LINK	PROPOGATION DELAY
Access point 1	Router	3 Mb	50 ms
Access point 2	Router	3 Mb	50 ms
Access point 3	Internet	2.5 Mb	22 ms
Access point 4	Internet	2.5 Mb	22 ms
Router	Internet	7 Mb	27 ms

Table 5.2 Details of wired connection in the designed network

Here, nine mobile nodes are considered. Two mobile nodes are registered with access point1, two mobile nodes are registered with access point2, three mobile nodes are registered with access point3 and two mobile nodes are registered with access point4. They send 1.4 Mb data to home base station node in 1 microsecond.

Circle around the access points in the figure 5.1 represent its coverage area in above figure. The communication range of access points is set as 60m in NS2. The mobile nodes in the scenario will use IEEE 802.11 standards for the communication. When evaluating the designed network, throughput and average delay are critical factors that determine the performance the designed network.

CHAPTER 6

NS2 WIRED CUM WIRELESS NETWORK IMPLEMENTATION

Development of wired cum wireless network will be done in Tcl. Ns-2 uses Tcl scripts for simulations. Many parameters must be defined and set when modelling the network in NS2. Mobile node is the basic Node object with the addition of different functionalities like movement in a 3 dimensional space and the ability to transmit on a wireless channel. In short network must be designed in such a way that mobile nodes need to register with its corresponding the access point (Base Station nodes) and in turn access point need to be set up with links to wired nodes. A wired node will also be implemented, which can represent the core connection point of our network; Mobile nodes will transfer packets via wired node, where we will store and analyse transmission data. Further section in this chapter will introduce some crucial steps that are to be followed while implementing wired cum wireless network in NS2.

6.1 HIERARCHICAL ADDRESSING IN WIRED CUM WIRELESS NETWORK

Hierarchical routing requires some additional features and mechanisms for the simulation.. Therefore the user must specify hierarchical routing requirements before creating topology. Also the need of hierarchical routing is explained in chapter 3 which is required for designed network. Below is an example of how it is done.

$ns node-config -addressType hierarchical
AddrParams set domain_num_ 3
lappend cluster_num 2 1 1
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 4 4
AddrParams set nodes_num_ $eilastlevel

Here, Class AddrParams is used to store the topology hierarchy like number of levels of hierarchy, number of areas in each level like number of domains, number of clusters and number of nodes in each cluster.

6.2 CONFIGURATION AND CREATION OF WIRED NODES

The nodes for wired networks can be created by the following command:
set <node_name> [$ns node]

The command creates and returns a node instance. This command is sufficient for creating the node in wired network which is followed by the commands to create links between the nodes. The syntax for creating duplex-link between the nodes:

$ns duplex-link <node1> <node2> <bw> <delay> < qtype>

This creates a bi-directional link between node1 and node2. This procedure essentially creates a duplex-link from two simplex links, one from node1 to node2 and the other from node2 to node1.Here bw represents the bandwidth of the link , delay represents data propagation delay from source node to destination node of given bandwidth and qtype represent type of queue used i.e. DropTail or RED.

6.3 CONFIGURATION AND CREATION OF MOBILE AND BASE STATION NODES

The node configuration interface for the wireless networks consists of two parts. In the first part the node is configured and the creation of the node of specified type takes place in the second part. Mobile nodes and Base Station nodes are configured using "node-config" command. It is important to note that Base Station nodes must have the "wiredRouting" option ON whereas it be should in OFF for mobile node. Some of the common options for the Mobile nodes and Base Station nodes must be configured, as listed below:

6.3.1 CHANNEL TYPE

The channel type simply describes the channel being used and in this case it will be a wireless channel.

6.3.2 ANTENNA TYPE

For defining the antenna type an omni-directional antenna having unity gain is used by the mobile nodes to communicate

6.3.3 Radio Propagation Models

The Radio propagation models receive the packets from the network interface layer, and are used to determine the received signal power of each packet. The radio propagation models compare the received power against the receiving threshold which is specified in the physical characteristics of the wireless node. If the signal power is below the receiving threshold, MAC layer drops the packet marking it as an error packet.

There are three propagation models that are presently used in NS2. They are Free space model, Two-ray ground reflection model and shadowing model. The propagation models are specified in the node-config command of the mobile node configuration. Above all, Two ray ground propagation model gives more accurate prediction at a long distance than other two models.

6.3.3.1 FREE SPACE MODEL

The Free space model is based on the ideal propagation condition where it is assumed that there is clear line-of-sight path between the transmitter and the receiver. According to this model, the received signal power is calculated according to the equation

$$P_r(d) = \frac{P_t\, G_t\, G_r\, \lambda^2}{(4\pi)^2\, d^2\, L} \qquad (6.1)$$

Where,
P_t = transmitted signal power
P_r = receiving threshold,
G_t = antenna gain of the transmitter (1.0 for Omni antenna)
G_r = antenna gain of the receiver (1.0 for Omni antenna).
L= system loss (= 1)
λ = wavelength

Free space model represents the communication range as a circle around the transmitter. If the receiver is within the circle then it receives all packets correctly. Free space model can be specified in the simulation setup as follows:

$ns node-config -propType Propagation/FreeSpace

6.3.3.2 TWO-RAY GROUND REFLECTION MODEL

Two-ray ground reflection model considers both the direct path and the ground reflection path. This model gives prediction at a long distance than the free space model. The receiving signal power at distance d is calculated as follows

$$P_r(d) = \frac{P_t\, G_t\, G_r\, h_t^2\, h_r^2}{d^4\, L} \qquad (6.2)$$

Where,
d = distance (Communication range)
G_t = antenna gain of the transmitter (1.0 for Omni antenna)
G_r = antenna gain of the receiver (1.0 for Omni antenna).
h_t = Height of the transmitter (1.5m by default in ns2)

h_r = Height of the receiver (1.5m by default in ns2)
P_t = transmitted signal power
P_r = receiving threshold,
L= system loss (= 1)

Two-ray ground reflection model is specified in the simulation setup as follows:

$ns node-config -propType Propagation/TwoRayGround

6.3.3.3 SHADOWING MODEL

Shadowing model considers the multipath propagation effects, which is the fading effects. The model considers the received power at a distance to be a random variable due to the multipath propagation effects, rather than considering the received power to be a deterministic function of distance. This model has two parts; a path loss model which predicts the mean received power at distance d, and a log-normal variable, which reflects the variation of the received power. Shadowing model can be specified in the simulation setup as follows:

$ns node-config -propType Propagation/Shadowing

6.3.4 LINK LAYER TYPE

Link layer captures the functionality of the three lowest layers in the network stack such as Link Layer (LL), Medium Access Control (MAC) Layer and Physical (PHY) Layer. Since link layer sits above MAC layer, therefore it can have functionalities such as queuing and link-level transmission. Usually all outgoing packets are handed down to Link Layer by Routing Agent following by the packets being transferred to the interface queue. However incoming packets are transferred to Link Layer by MAC.

6.3.5 INTERFACE QUEUE TYPE

Ideally interface queue is situated between Link Layer and MAC. This is also known as interface priority queue and connected to the channel. It's defined as set val(ifq) for simulation purposes. Interface queue is implemented in order than to give priority to routing protocol packets and it also supports filter to remove packets to a specific address.

6.3.6 MAC TYPE

This is where 802.11 protocols are implemented. To send packets MAC follows a certain medium access protocol to send packets on the channel and on the other hand MAC is responsible for delivering packets to link layer when it comes to receiving.

6.3.7 NETWORK INTERFACE TYPE

Network interfaces are used in multicast routing. Network Interfaces serve as hardware interface, which is used by mobile node to access the channel. It is implemented as Phy/WirelessPhy class and subjected to collisions and radio propagation.

6.3.8 TOPOGRAPHY

The topography that is the area in which the wireless nodes move must be specified. The topography of a flat grid with length and width in x times y meters is defined as:

set topo [new Topography]

$topo load_flatgrid $opt(x) $opt(y)

The instance of the topography can then be passed to the mobile node in the node config command with the "-topolnstance" option.

Options	Available Values	Values used in the designed implementation
addressType	flat, hierarchical	Hierarchical
wiredRouting	ON , OFF	ON (for Base station nodes) OFF (for Mobile nodes)
llType	LL , LL/Sat	LL
macType	Mac/802_11,Mac/Csma/ca, Mac/Sat, Mac/Sat , Mac/Sat/UnslottedAloha, Mac/Tdma	Mac/802_11
ifqType	Queue/DropTail, Queue/DropTail/PriQueue	Queue/DropTail,
adhocRouting	DSDV, AODV	DSDV
antType	Antenna/OmniAntenna	Antenna/OmniAntenna
propType	Propagation/FreeSpace, Propagation/TwoRayGround, Propagation/Shadowing	Propagation/TwoRayGround
Channel	Channel/WirelessChannel, Channel/Sat	Channel/WirelessChannel
mobileIP	ON , OFF	ON
agentTrace	ON , OFF	ON
routerTrace	ON , OFF	ON
macTrace	ON , OFF	ON

Table 6.1 Wireless Node Configuration Options

The most commonly used node configuration options are listed in the above table.

An example of node configuration with “node-config” command is as follows:

```
$ns node-config  -adhocRouting  DSDV \
                 -llType  LL \
                 -macType MAC/802_11 \
                 -ifqType Queue/DropTail \
                 -ifqLen 50 \
                 -mobileIP ON\
                 -antType Channel/WirelessChannel  \
                 -propInstance Propgation/TwoRayGround \
                 -phyType Phy/WirelessPhy \
                 -topoInstance $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF
```

6.3.9 ROLE OF GOD IN WIRELESS NODE

GOD (General Operations Director) object must be specified. This object stores the information about the total number of nodes and the table of the shortest number of hops that is required to reach from one node to another . It traces the information about the number of hops from source to destination in the trace file. The GOD object is specified in the simulation setup in the following way:

create-god [expr $opt(nn) + $opt(bs_nodes)]

Here,
$opt(nn) refers to Number of Mobile Nodes in the Network
$opt(bs_nodes) refers to Number of Base Station Nodes in the Network

6.3.10 CREATION OF WIRELESS NODE

After the configuration the mobile / BaseStation node is created with the following command:

set <node> [$ns node $h_addr]

Here $h_addr represent the hierarchical address of mobile or base station node

Make sure you turn on wiredRouting configuration for base station nodes and turn it off for mobile nodes.

The initial node position is set by the following command:

$mobilenode set X <x>
$mobilenode set Y <y>

These commands will set the start position of the mobile nodes with (x, y, 0)

The mobile node in ns-2 move in a flat topology which means that only x and y values are specified. The third coordinate z, even though present in the node position setup is set to zero.

6.4 Communication Range Specification

Communication range of the wireless nodes can be specified in the simulation setup after defining the network interface characteristics of the wireless node. Receiving threshold can be calculated from the network interface characteristics and the selected radio propagation model can be used for the communication range adjustment. The command to adjust the communication range is as follows:

Phy/WirelessPhy set RXThresh_ <value>

6.5 REGISTER MOBILE NODE WITH BASE STATION

Mobile node has an address called the care-of-address (COA). Based on the registration/beacons exchanged between the Mobile node and the base-station node (domain of the Mobile node is currently in), the base-station's address is assigned as the Mobile node's COA. Thus in this simulation, address of the home agent is assigned initially as the COA of mobile node . As MH moves in to the domain of FA, its COA changes .Following commands are used to register mobile node with base station node (Home Agent)

set home_addr [AddrParams addr2id [$BS0 node-addr]]
[$MH0 set regagent_] set home_agent_ $home_addr

6.6 Transport Agents

Agents are the end-points where the network layer packets are generated and are used for implementing protocols at various layers. The routing agents and traffic sources, sinks are

the frequently used agents in the simulations. The various agents that are available in NS2 belong to the class agent. The class agent is implemented by OTcl and C++.

Agents are attached to the nodes by issuing a command called attach-agent to the simulator instance. The syntax of attaching the agent to the node is as follows:

$ns attach-agent <node> <agent>

The agents that are attached to two nodes can communicate with each other by issuing the command that connect to the simulator instance. The syntax of the command is as follows:

$ns connect <src_agent> <dest_agent>

6.7 ATTACHING APPLICATION TO AGENT

Applications can be attached to the agents by using an API which provides the interaction between the agents and applications. The API sends notifications of received data to the application. The API is used for simple application such as FTP, CBR , telnet and traffic generator applications. The command used to create and attach an ftp application to the tcp agent is:

set ftp [new Application/FTP]
$ftp attach-agent $tcp

6.8 DEFINING MOBILITY TO MOBILE NODES

The node can be changed to a different position using the following command:

Sns at $time $mobilenode setdest <x> <y> <speed>

The above command is used to move the node at time $time sec to new position specified by <x>, <y> coordinates. The node moves with a velocity specified by <speed> in m/sec.

These are some important steps to follow after creating the simulation object and before scheduling the events in the simulation in Tcl Script.

CHAPTER 7

PARAMETERS USED IN PERFORMANCE EVALUATION

In order to assess the performance of our wired cum wireless network, a number of parameters and metrics were recorded and monitored. After collecting the data recorded in the trace files in ns2, we use AWK scripts to calculate the performance of transport agent over designed network. Quality of Service (QoS) is a measurement of how good the routes in the network are. The routes should guarantee a set of pre-specified service attributes such as delivery, bandwidth, and delay variance. For a protocol to provide good QoS it must determine new routes rapidly and with minimal bandwidth consumption. There are several metrics that directly affect the QoS of every protocol of which three main performance metrics will be calculated in this project.

7.1 THROUGHPUT OBTAINED

Throughput is the number of successfully received packets in a unit time and it is represented in Mbps. Throughput is calculated using AWK script which processes the trace file and produces the result. Formula for calculating throughput is as follows:

$$Throughput = \frac{Bytes\ Recieved \times 8}{Delay \times 1000000} \qquad (7.1)$$

Bytes Received of above formula is the packet size received by the receiver and the value is obtained from the 8^{th} field of wireless trace format (since mobile node is considered as receiver in this project). Delay represent time interval between the sending time and receiving time of the packet. Based on the event sending (s) or receiving (r), it corresponding time is noted and mathematical subtraction is performed on the receiving time and sending time to obtain the delay. 2^{th} field and 6^{th} field of wireless trace format represent time and packet number. Calculated bytes received per delay is multiple by 8 for converting bytes to bits and divided by 1000000 for converting bits to Mbits. Thus if throughput is high, then reliability of data transfer in communication is also high

7.2 PACKET DELIVERY RATIO

Packet delivery ratio is the quotient resulting from the number of unique data packets arrived at the destination divided by the unique data packets sent from a source. Packet delivery ratio measures the protocol performance in the network and this performance may depend on factors such as packet size, network load, as well as the effects of frequent topological changes.

Now, the packet delivery ratio of the protocol in network can be calculated as follows:

$$Packet\ Delivery\ Ratio = \frac{\sum Total\ Packets\ received\ by\ the\ sink\ node}{\sum Total\ Packets\ send\ by\ the\ source\ node} \quad (7.2)$$

The sending and receiving event can be identified from the 1st field of the trace file.
The Node number can identified from 3rd field. Based on the first and third field, the number of sent and received packets over the transport agent is calculated via AWK Script from which packet delivery ratio of a protocol is obtained.

7.3 AVERAGE END-TO-END DELAY

End-to-end delay is the time it takes for a packet to travel through the network from source to destination. The average end-to-end delay is the summation of all end-toend delays divided by total data packets arrived at destination. End-to-end delay is an important routing performance metric since voice and video applications are especially dependent on low latency to perform well. In networks with a low PDR, samples with short paths are favoured and thus the delay will be low. Calculation of the average end-to-end delay includes all possible delays caused by buffering during route discovery latency, queuing at the interface queue, retransmission delays at the MAC, and propagation and transfer times.

Finally, the average end-to-end delay is calculated by taking the time of the last packet received subtracting the initial time of transmitting that packet, and then dividing by the total number of packets sent that sampling interval. By doing this, we can achieve the average end-to-end delay of the transport agent connection.

$$Average\ end - to - end\ delay = \frac{Time(packet\ received) - Time(packet\ sent)}{Total\ packets\ recieved} \quad (7.3)$$

From 1st and 6th field of trace files necessary values for calculation of average end-to-end delay is obtained and its result is calculated.

CHAPTER 8

EXPERIMENTAL RESULTS

To visualize and analyse the performance of transport agent protocol and Mobile IP over the designed wired cum wireless network different situations were considered in the simulations. Network AniMator (NAM) is useful in visualizing the scenarios. AWK scripts and python matplotlib are used in analysing and plotting the necessary data available from statistical data.

8.1 UDP CONNECTIONS OVER THE DESIGNED WIRED CUM WIRELESS NETWORK

Here three different scenarios are considered for the UDP connection. Following table summarises the simulation of UDP connection.

Time Interval	0.5 to 45.5 seconds
Transport Agent	UDP (User Datagram Protocol)
Application Agent	CBR (Constant bit rate)
Node Number which sends data	6
Node Number which receives data	14
Number of Scenario	3

Table 8.1 Details of UDP Connection in simulation

8.1.1 ESTABLISHING UDP CONNECTION BETWEEN SENDER AND RECEIVER NODE AND CBR TRAFFIC FLOWING OVER IT

Between 0.5 and 16.0 seconds both sender and receiver nodes are located within communication range of its corresponding base station node and this scenario is visualized in NAM animator as shown in the below figure. Also note that no other traffic flow is introduced at this point of time.

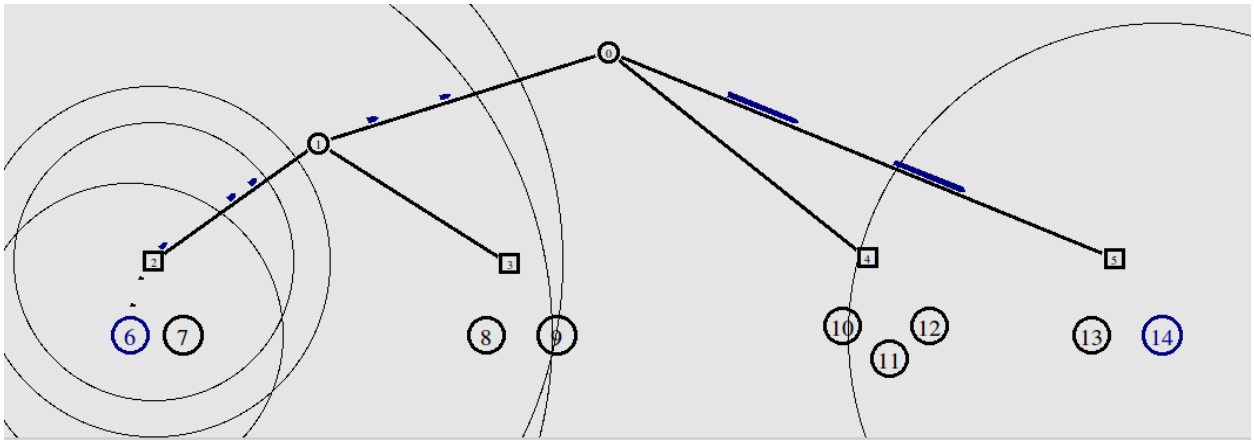

Figure 8.1 CBR Traffic flow between node 6 and node 14

8.1.2 WHEN RECEIVER NODE IS MOVING AWAY FROM ITS BASE STATION NODE DURING THE COMMUNICATION

After 16.0 seconds the receiver node 14 starts moving away from its corresponding base station node and at certain point of time it has reached far away from its base station node so that packets cannot be reached to receiver node and thus results in packet loss and this is scenario is visualized in NAM as shown below.

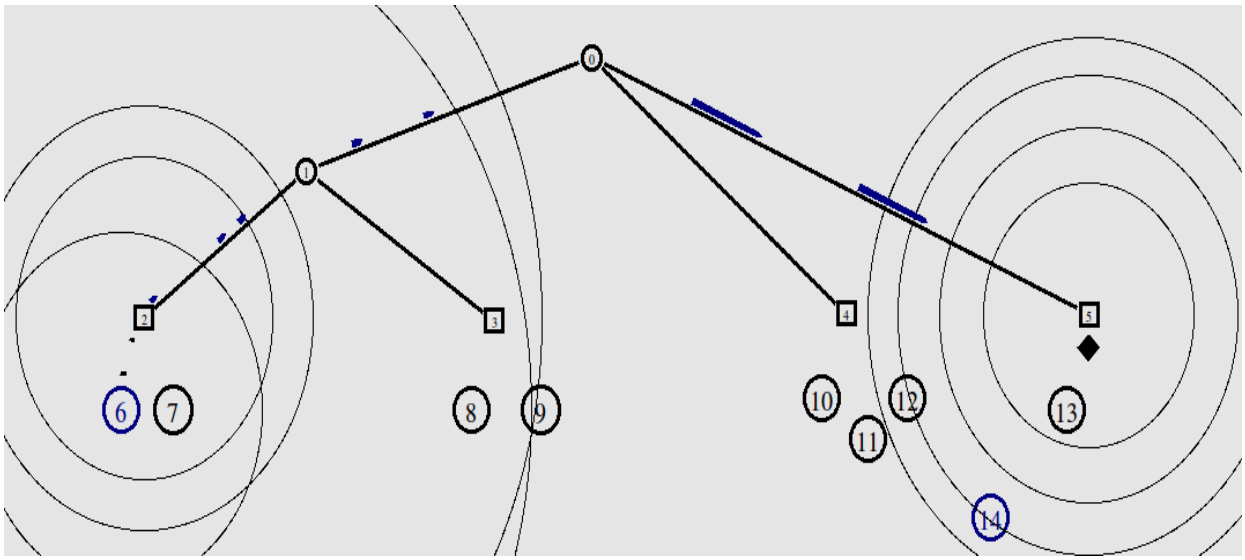

Figure 8.2 Receiver node 14 is in out of coverage area

Diamond in the above figure 8.2 near base station node number 5 represent that packet has been dropped.

8.1.3 RECEIVER NODE IS AGAIN IN COMMUNICATION RANGE WITH ITS BASE STATION NODE AND TRAFFIC LOAD IS MADE HEAVY WITH ITS BASE STATION NODE

After 30.8 seconds, the receiver node 14 is made to enter within communication radius of its corresponding base station node (node number 5) and before entry of node 14 other traffic flows is created in the network. This is to visualize how UDP transport agent works along with other transport agents. Details about other traffic are given in the section 8.2 of the same chapter.

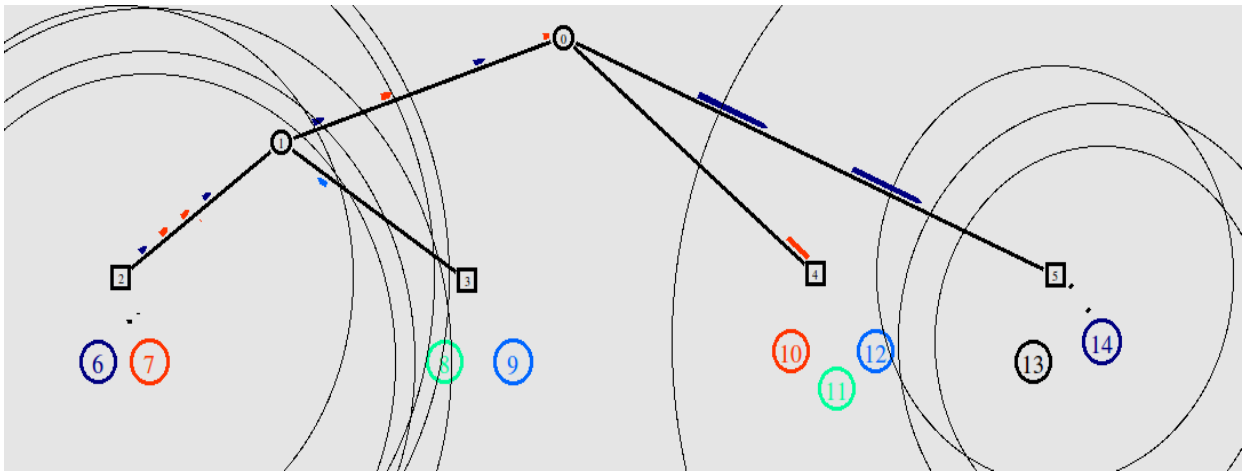

Figure 8.3 Other Traffic flows introduced along with UDP Connection

8.1.4 PERFORMANCE METRICS OF UDP CONNECTION

Performance metrics like throughput, packet delivery ratio (pdr) and average end to end delay of the UDP connections are calculated using AWK scripts by parsing the statistical data found in the trace file. In below figure 8.4, the file mentioned 'udp.awk' contains logic for parsing the trace file to find the performance metrics of UDP Connection and 'Trace_out.tr' is the trace file that contains the simulation data.

```
viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$ awk -f udp.awk Trace_out.tr
Throughput Results :

Throughput Obtained = 0.338240 Mbps

Packet Delivery Ratio Results :

No of packets Sent       :   2880
No of packets Recieved   :   1864
No of packets dropped    :   1016
Packet delivery Ratio    :   0.647222

Calculating end to end delay:

Total End-to-End Delay = 315.312372
Average End-to-End Delay = 0.169159
```

Fig 8.4 Performance metrics evaluation of UDP Connection

8.1.5 GRAPHICAL REPRESENTATION OF INSTANTANEOUS THROUGHPUT IN UDP CONNECTION

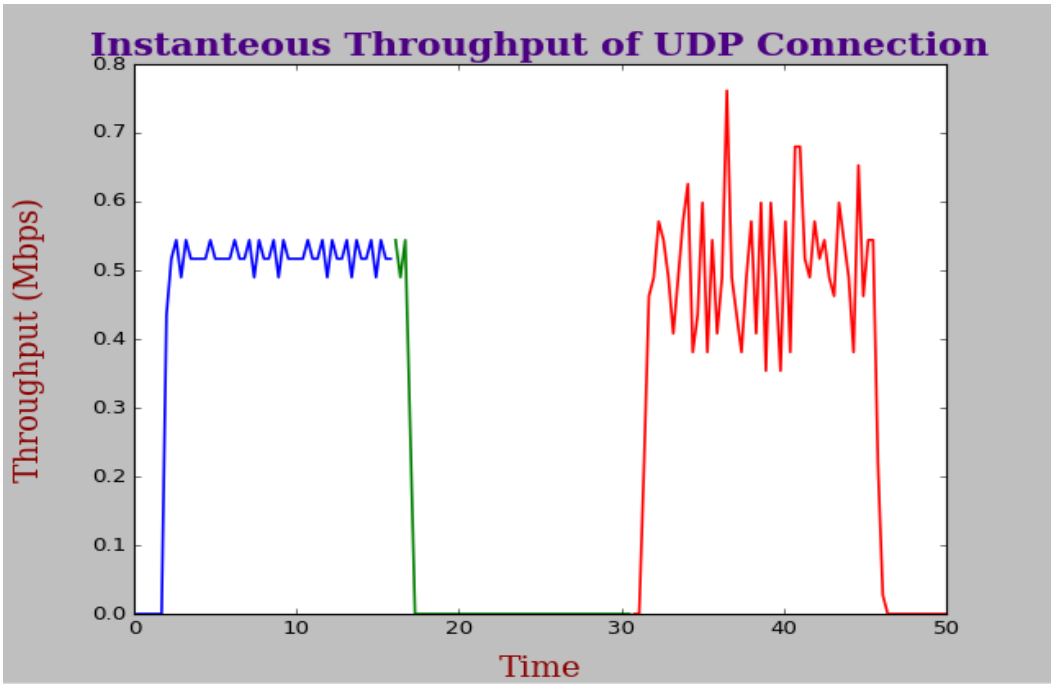

Fig 8.5 Throughput plot of UDP Connection

The graphical representation of throughput is given in above figure 8.5 where x axis represents the time and y axis represent throughput obtained. The blue line plot in graph represents the first scenario of UDP connection, where it achieves constant throughput as there is no other traffic flow is introduced. Next the green line plot in graph represents the second scenario where packet loss occurred due to which we can see that throughput achieved is mostly zero. Finally red plot of graph represent the third scenario, where due to more traffic introduced in the network, we can see that throughput is oscillating manner within this time interval. So we can conclude that a constant throughput cannot be achieved by UDP Connection when the network is busy.

8.2 TCP CONNECTIONS OVER THE DESIGNED WIRED CUM WIRELESS NETWORK

In the time interval 22.0 to 66.0 seconds, three different TCP connections are established in designed network. Tahoe, Reno and Vegas are the three different congestion control algorithms that are used in these three TCP connections. Details of these three TCP connections are as follows:

Time Interval	23.7 to 63.0 seconds
Transport Agent	TCP Tahoe
Application Agent	FTP (File Transfer Protocol)
Node Number which sends data	7
Node Number which receives data	10
Colour of Packet flow in NAM	Red

Table 8.2 Details of TCP Tahoe Connection in simulation

Time Interval	23.1 to 66.0 seconds
Transport Agent	TCP Reno
Application Agent	FTP (File Transfer Protocol)
Node Number which sends data	8
Node Number which receives data	11
Colour of Packet flow in NAM	Green

Table 8.3 Details of TCP Reno Connection in simulation

Time Interval	23.7 to 65.0 seconds
Transport Agent	TCP Vegas
Application Agent	FTP (File Transfer Protocol)
Node Number which sends data	9
Node Number which receives data	12
Colour of Packet flow in NAM	Blue

Table 8.4 Details of TCP Vegas Connection in simulation

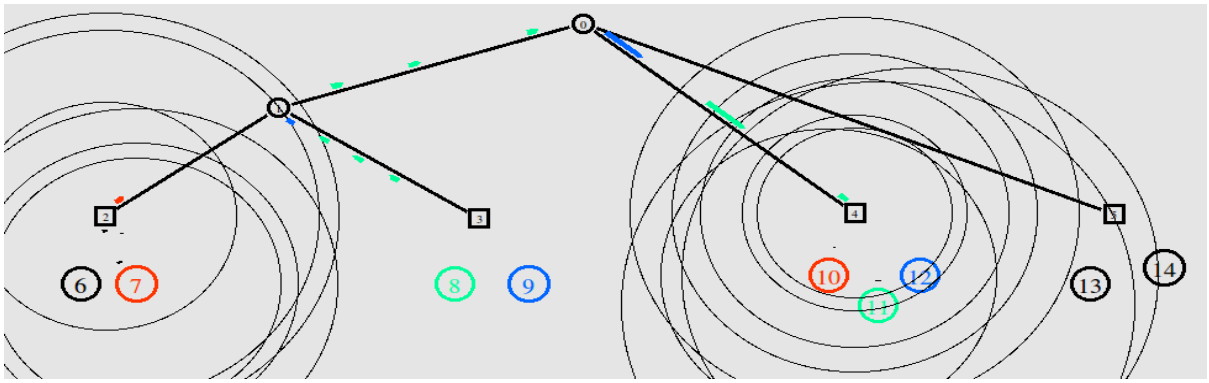

Figure 8.6 Three different TCP Connections in the network

The main motive for creating this situation is to check how TCP Vegas works or reacts when Vegas and Reno congestion control mechanisms connections share a bottleneck link in the designed wired cum wireless network. Here TCP Tahoe creates the bottleneck for previous UDP Connection situation

8.2.1 PERFORMANCE METRICS OF TCP CONNECTIONS

Performance metrics like throughput, packet delivery ratio (pdr) and average end to end delay of the Three different connections are calculated using AWK scripts by parsing the statistical data found in the trace file. Below figure shows the performance metrics evaluation results of three different TCP congestion algorithms.

```
viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$ awk -f tcp_CA.awk Trace_out.tr
        Transport Layer -----> TCP Tahoe

Throughput Results :
Throughput Obtained = 0.346188 Mbps

Packet Delivery Ratio Results :
No of packets Sent       :   1627
No of packets Recieved   :   1605
No of packets dropped    :   22
Packet delivery Ratio    :   0.986478

Calculating end to end delay :
Total End-to-End Delay = 278983.751535 ms
Average End-to-End Delay = 173.821652 ms

        Transport Layer -----> TCP Reno

Throughput Results :
Throughput Obtained = 0.118275 Mbps

Packet Delivery Ratio Results :
No of packets Sent       :   600
No of packets Recieved   :   584
No of packets dropped    :   16
Packet delivery Ratio    :   0.973333

Calculating end to end delay:
Total End-to-End Delay = 107555.530214 ms
Average End-to-End Delay = 184.170428 ms
```

```
        Transport Layer -----> TCP Vegas

Throughput Results :
Throughput Obtained = 0.031707 Mbps

Packet Delivery Ratio Results :
No of packets Sent          :    164
No of packets Recieved      :    160
No of packets dropped       :    4
Packet delivery Ratio       :    0.975610

Calculating end to end delay:
Total End-to-End Delay = 50512.767561 ms
Average End-to-End Delay = 315.704797 ms

viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$
```

Fig 8.7 Performance metrics evaluation of TCP Connections

8.2.2 RESULTS INTERPRETED FROM CONGESTION WINDOW PLOT OF TCP CONNECTIONS

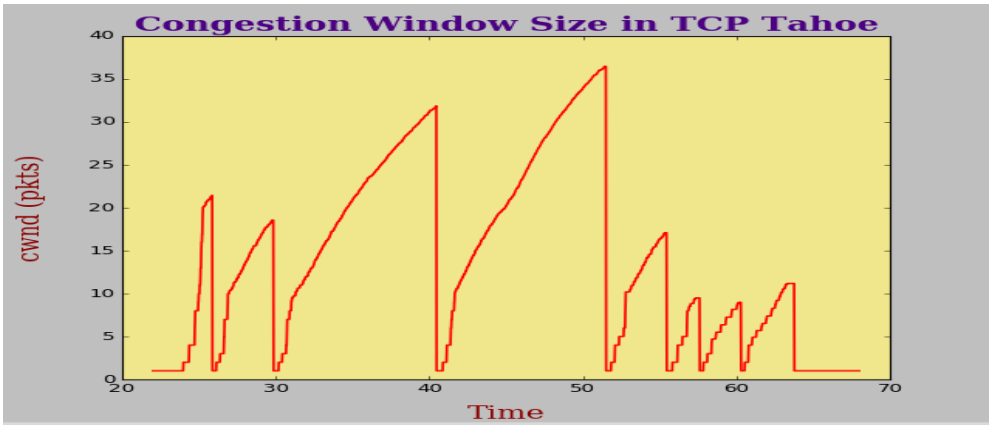

Fig 8.8 Congestion window graph of TCP Tahoe in interval 22-70 seconds

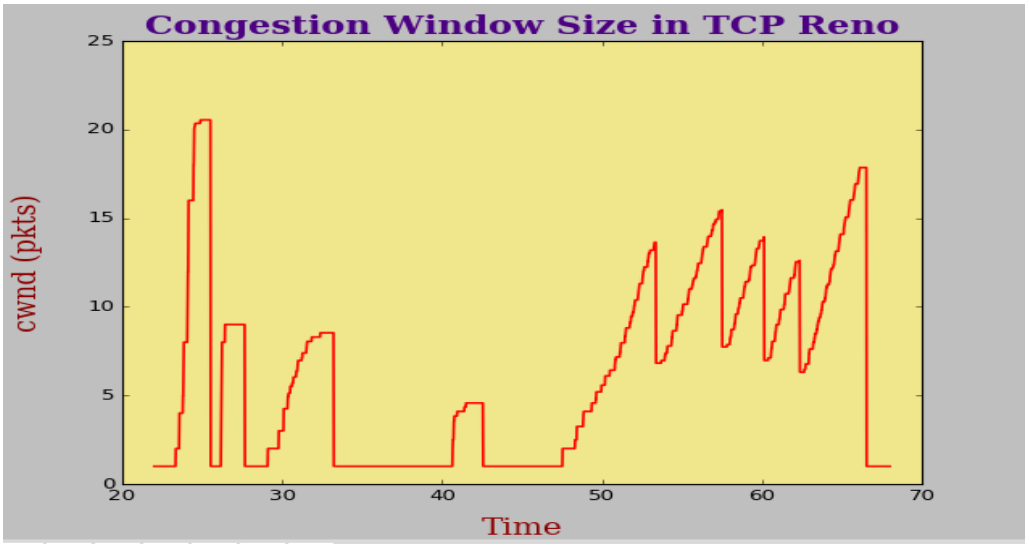

Fig 8.9 Congestion window graph of TCP Reno in interval 22-70 seconds

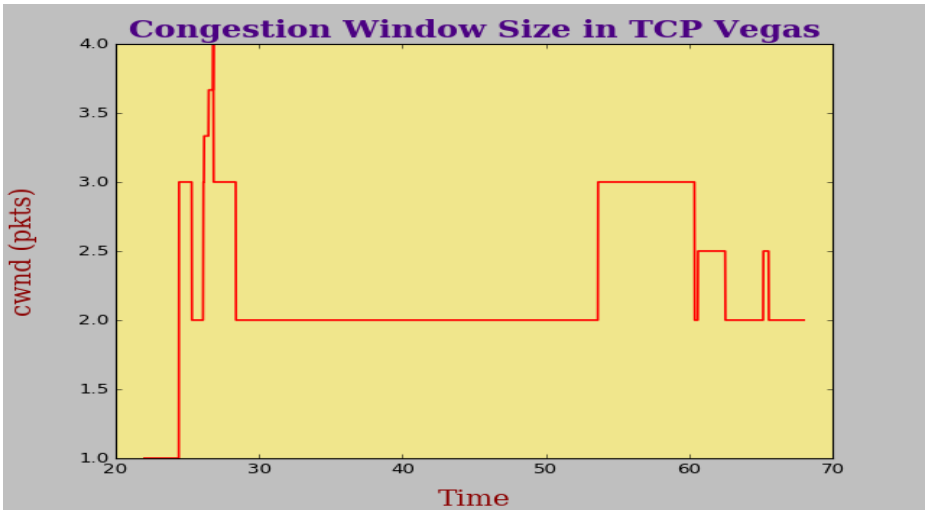

Fig 8.10 Congestion window graph of TCP Vegas in interval 22-70 seconds

If congestion window size increases, then throughput received by the receiver will high. TCP Reno and TCP Tahoe models can determine the congestion in the network only when there some packet loss occurred in the system. In these models when packet loss occurs the window size is decreased and the system enters the congestion avoidance phase. While TCP Vegas senses the congestion in the network before any packet loss occurs and instantly it decreases the window size. So, TCP Vegas handles the congestion without any packet loss occurs. This is a quick overview the three different TCP congestion control mechanisms.

It is proved in some papers after experiments and simulation that TCP Vegas performs 40 -70 percent better than other TCP models like Tahoe and Reno. In this second situation we consider, TCP Vegas share bottleneck link with TCP Reno. From above congestion plots figures it is found that then Vegas doesn't achieve the good throughput because of its lesser congestion window size compared to TCP Reno. Below paragraph explain why the congestion window of TCP Vegas is low when compared to Reno.

When TCP Vegas and Reno connections share a bottleneck link in the network, TCP Reno fully utilizes the bandwidth, continues to increase the window size until a packet loss is detected. Thus Reno uses at most of the link and router buffer space. Whereas Vegas interpreting this as a sign of congestion, decreases its congestion window, which leads to an unfair sharing of available bandwidth in favour of Reno. This unfairness worsens when router buffer sizes are increased.

From Performance metric evaluation, it is found that TCP Vegas achieved 73.1921% throughput less than TCP Reno. Thus we can conclude that TCP Vegas doesn't perform well with other TCP congestion control mechanism in the designed network based on throughput it achieved, but it ensures that there is only a minimal packet drops in the established connection than other congestion control algorithms.

8.2.3 THROUGHPUT PLOT OF TCP CONNECTIONS

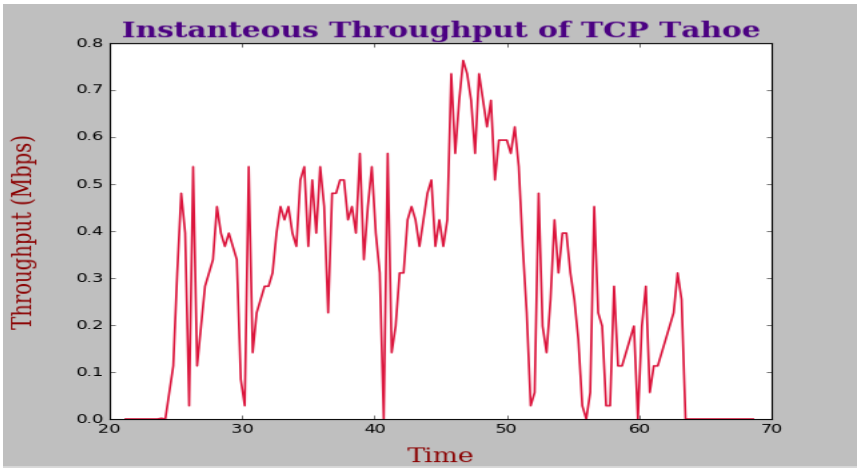

Fig 8.11 Instantaneous throughput of TCP Tahoe in interval 22-70 seconds

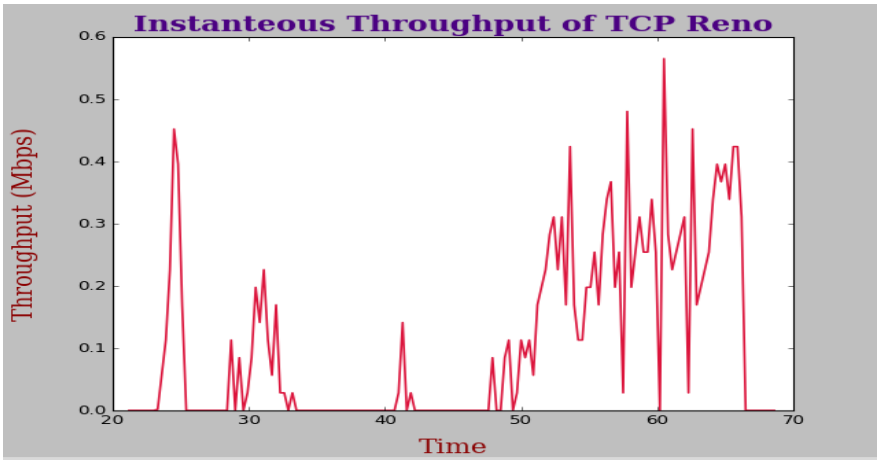

Fig 8.12 Instantaneous throughput of TCP Reno in interval 22-70 seconds

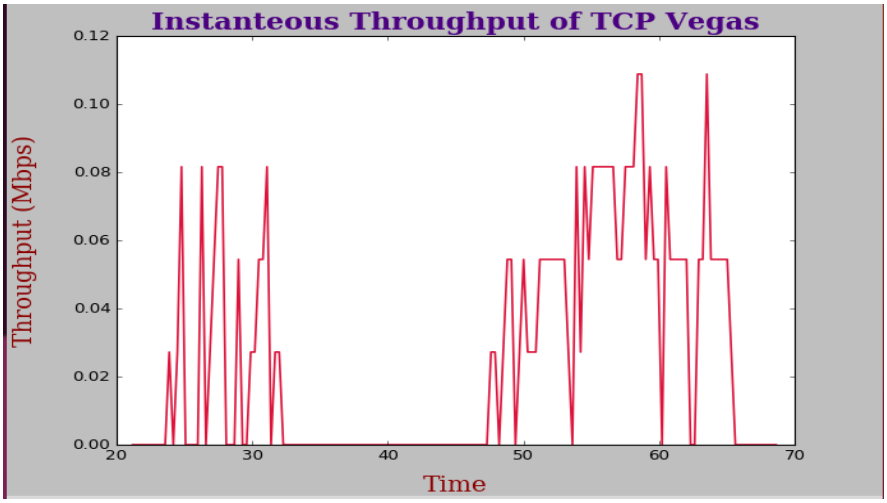

Fig 8.13 Instantaneous throughput of TCP Vegas in interval 22-70 seconds

8.3 MOBILE HOST IS MOVING IN THE NETWORK

TCP Connection is established between node 7 and node 13 in the network. Here node 13 is referred as Mobile Host (MH) which is receiving Data and node 7 is referred as remote host (RH) which sends data. In this situation only the receiver node 13 is made to move in the network and also making the sender node 7 to be remained in the same position. We visualise how Mobile IP works in this situation in NAM animator. Below table describe further details about this situation

Time at which traffic flow start	68.0^{th} second
Transport Agent	TCP Tahoe
Application Agent	FTP (File Transfer Protocol)
Node Number which sends data	7
Node Number which receives data	13
Number of bytes to be sent to receiver	3.5 Mb
Time at which the traffic flow ended in NAM animator (approximate time)	138.0^{th} second
Approximate Time taken to send data	70 seconds
Number of Scenario	2

Table 8.5 Details about situation 1 for Mobile IP in simulation

Here we are not so sure about the end time of this situation because we have schedule this situation in simulation based on the number of bytes to be sent to receiver and not based on the finish time of the Traffic flow.

8.3.1 COMMUNICATION BETWEEN MH AND RH IN CORRESSPONDING HOME AGENT

Between 68.0 and 98.5 seconds both sender and receiver nodes are located within communication range of its corresponding base station node. Here base station node of sender and receiver is node 2 and node 5. This scenario is visualized in NAM animator as shown in the below figure.

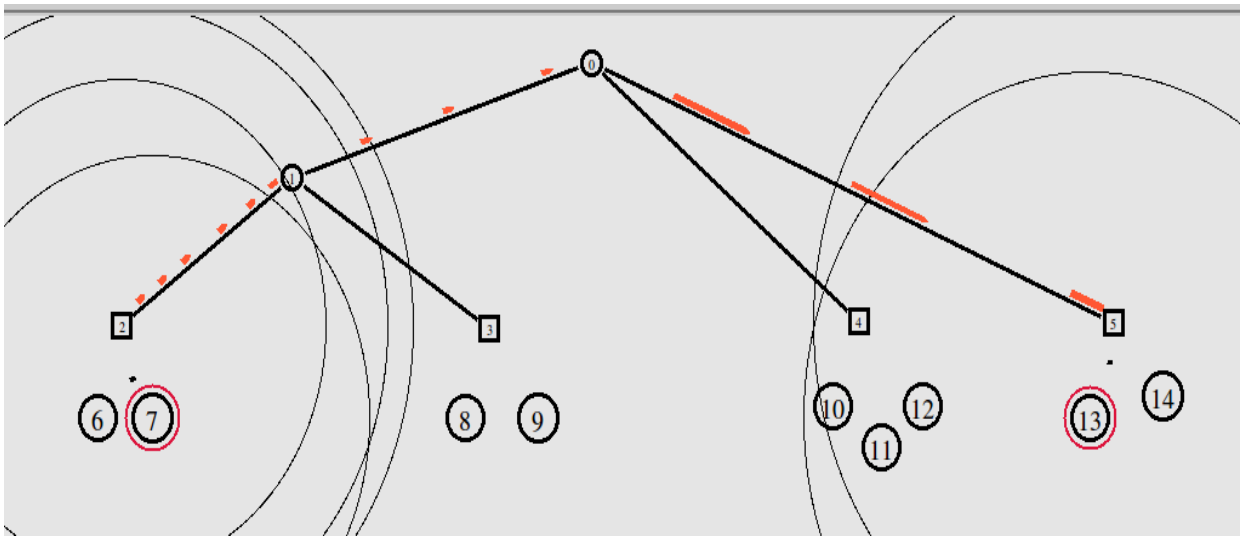

Figure 8.14 Communication between node 7 and 13

8.3.2 COMMUNICATION BETWEEN RH IN HOME AGENT AND MH IN FOREIGN AGENT

After 98.5 seconds the receiver node 13 starts moving away from its corresponding base station node number 5 and after certain point of time the receiver node reaches the foreign agent (enters the region of base station node number 3) via base station node number 4. The packet flow of receiver node after reaching the foreign agent can be visualized in NAM animator as shown in below figure 8.6

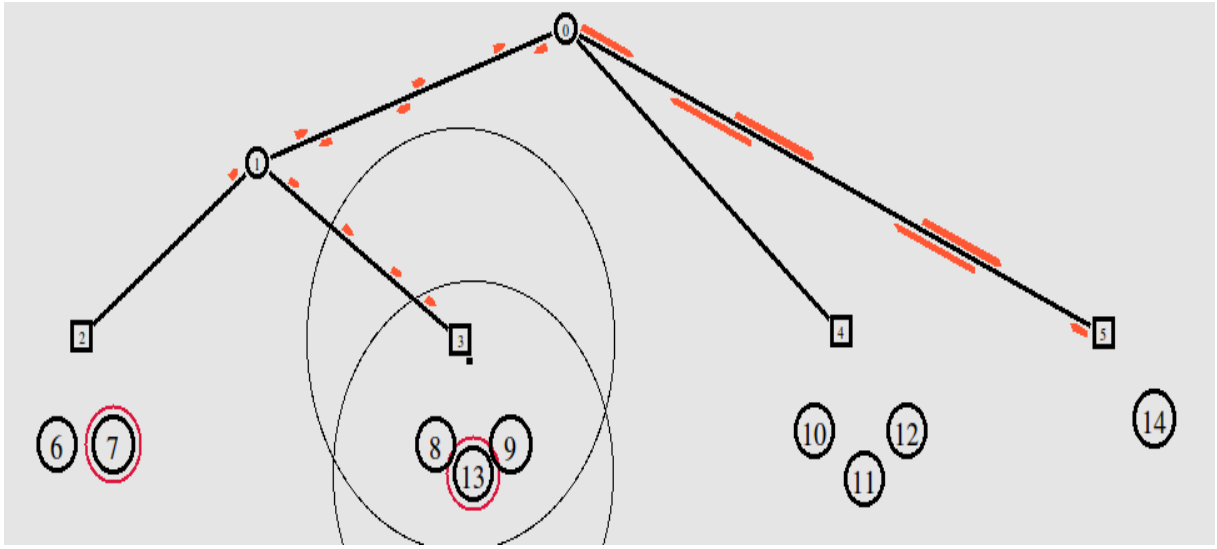

Figure 8.15 Node 13 reached foreign agent during the communication

8.3.3 PERFORMANCE METRICS EVALUATION

Performance metrics like throughput, packet delivery ratio (pdr) and average end to end delay are calculated using AWK scripts by parsing the statistical data found in the trace file. Below figure shows the performance metrics evaluation results of this situation.

```
viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$ awk -f mip1.awk Trace_out.tr
Throughput Results :

Throughput Obtained = 0.452477 Mbps

Packet Delivery Ratio Results :

No of packets Sent       :   3707
No of packets Recieved   :   3673
No of packets dropped    :   34
Packet delivery Ratio    :   0.990828

Calculating end to end delay:

Total End-to-End Delay = 567807.642031 ms
Average End-to-End Delay = 154.589611 ms

viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$
```

Fig 8.16 Performance metrics evaluation of situation 1 for Mobile IP

8.1.5 GRAPHICAL REPRESENTATION OF INSTANTANEOUS THROUGHPUT OF THIS SITUATION

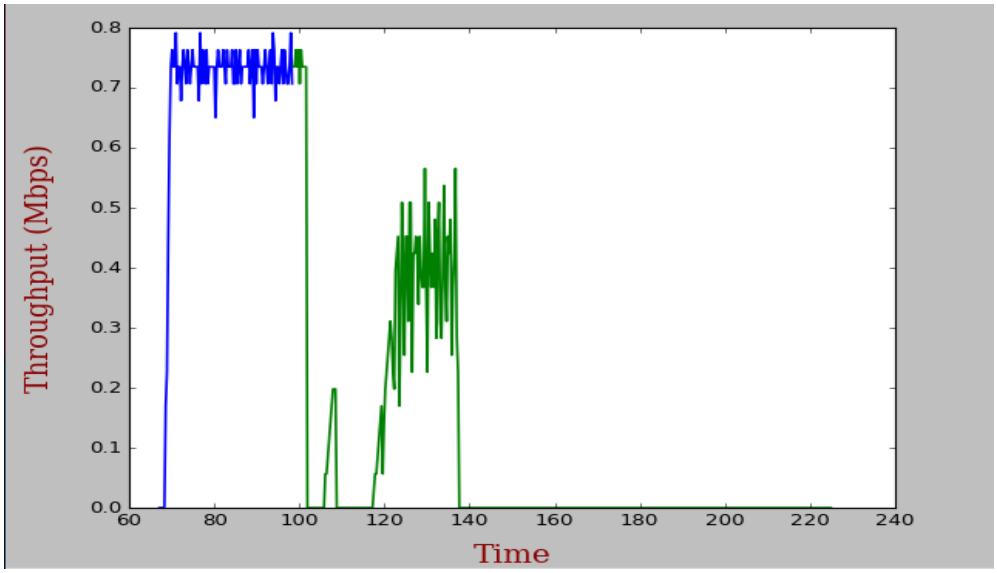

Fig 8.17 Instantaneous Throughput obtained by situation 1 considering Mobile IP

The graphical representation of throughput is given in above figure 8.5 where x axis represents the time and y axis represent throughput obtained. The blue line plot of the graph is for scenario 1 where MH and RH are present in its home agent. The green line plot represents throughput achieved after started moving towards the foreign agent (scenario2).

In time interval between 100th second and 120th second a little throughput is obtained because in between path of home agent (HA) and foreign agent (FA) there is base station node number 4 which notifies the HA of receiver node when this node is present in its coverage area due to which the HA send the packets to base station number 4 and the packets are received by the node. After 120th second, we can see the throughput achieved by receiver node when reached the foreign network (base station node 3) .Throughput achieved by receiver node in FA will be low than the throughput achieved by receiver node in FA because of triangular routing is involved in sending packets to the FA

8.4 REMOTE HOST IS MOVING IN THE NETWORK

TCP Connection is established between node 6 and node 10 in the network. Here node 6 is referred as Mobile Host (MH) which is receiving Data and node 10 is referred as remote host (RH) which sends data. In this situation the sender node 13 is made to move in the network and also making the receiver node 6 to be remained in the same position. We visualise how Mobile IP works in this situation in NAM animator. Below table describe further details about this situation

Time at which traffic flow start	139.0th second
Transport Agent	TCP Vegas
Application Agent	FTP (File Transfer Protocol)
Node Number which sends data	10
Node Number which receives data	6
Number of bytes to be sent to receiver	1.8 Mb
Time at which the traffic flow ended in NAM animator (approximate time)	198.0th second
Approximate Time taken to send data	68 seconds
Number of Scenario	3

Table 8.6 Details about situation 2 for Mobile IP in simulation

8.4.1 COMMUNICATION BETWEEN MH AND RH IN CORRESSPONDING HOME AGENT

Initially, both sender node and receiver node are located in the home locations .NAM output for this scenario can visualized as shown in the below figure

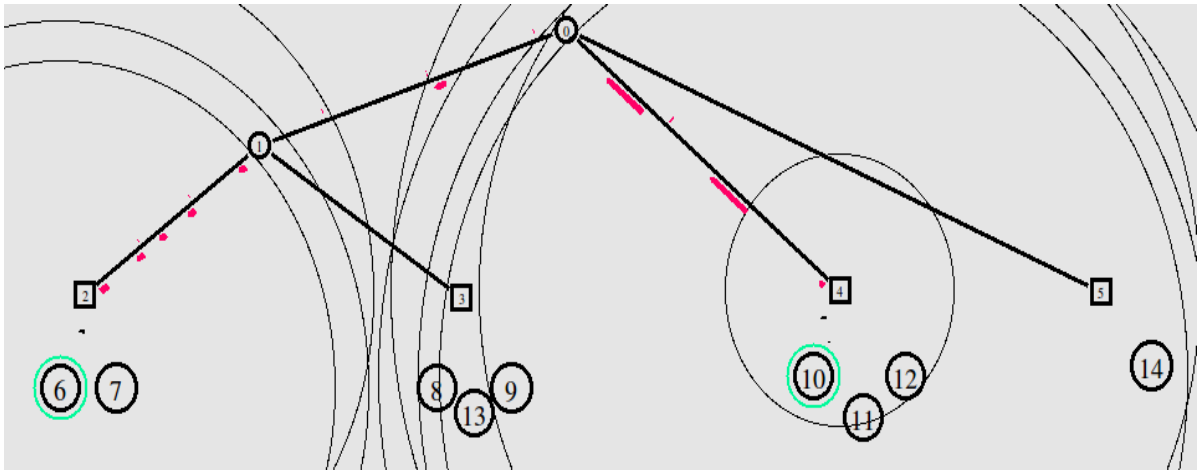

Figure 8.18 Node 10 reached the Foreign Agent (FA)

8.4.2 COMMUNICATION BETWEEN RH IN FOREIGN AGENT AND MH IN HOME AGENT

After 143.0th second the sender node 10 starts moving away from its corresponding base station node number and after certain point of time the receiver node reaches the foreign agent (enters the region of base station node number 3) . The packet flow can be visualized in NAM animator as shown in below figure 8.6

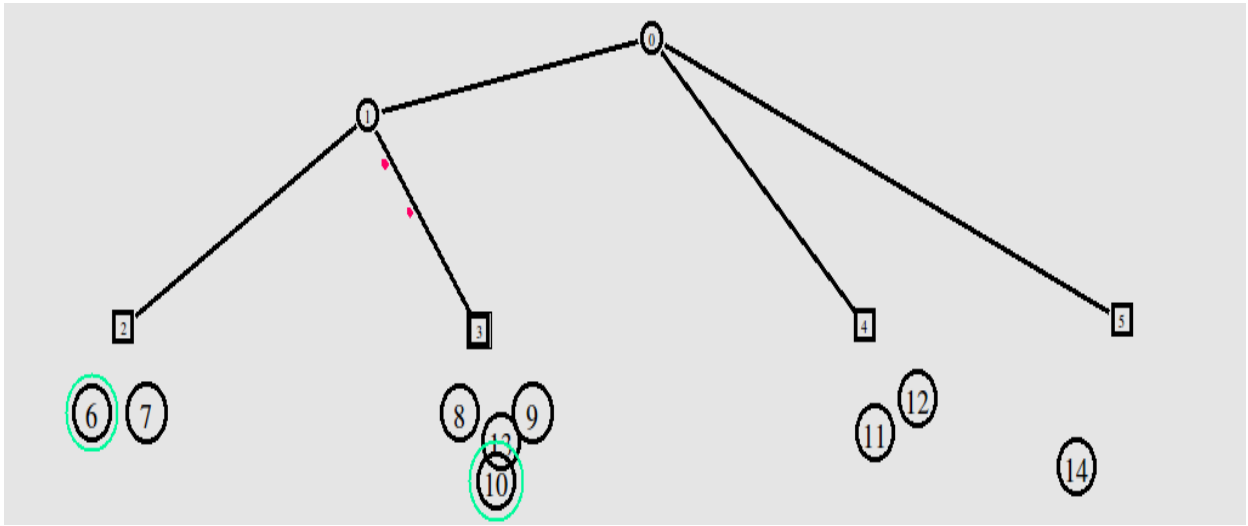

Figure 8.19 Node 10 reached foreign agent during the communication

In this scenario, TCP acknowledgment packet experiences triangle routing issue.

8.4.3 COMMUNICATION BETWEEN MH AND RH IN MH'S HOME AGENT

When Mobile Host (node 6) and Remote Host (node 10) are present in Mobile host location then the communication takes place directly. This is visualized in NAM animator as shown in the given figure

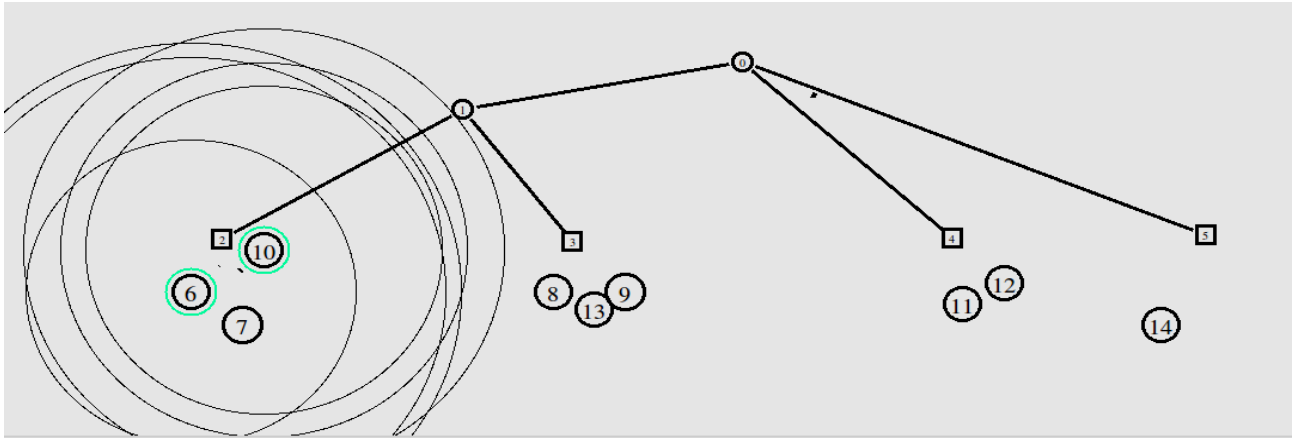

Figure 8.20 Node 10 reached home agent of node 6 during the communication

8.4.4 PERFORMANCE METRICS EVALUATION

Performance metrics like throughput, packet delivery ratio (pdr) and average end to end delay are calculated using AWK scripts by parsing the statistical data found in the trace file. Below figure shows the performance metrics evaluation results of this situation.

```
viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$ awk -f mip2.awk Trace_out.tr
Throughput Results :

Throughput Obtained = 0.329930 Mbps

Packet Delivery Ratio Results :

No of packets Sent       :   1532
No of packets Recieved   :   1526
No of packets dropped    :   6
Packet delivery Ratio    :   0.996084

Calculating end to end delay:

Total End-to-End Delay = 27449.218075 ms
Average End-to-End Delay = 17.987692 ms

viswa@viswa-VirtualBox:~/Documents/ns2program/Wired-cum-wireless$
```

Fig 8.21 Performance metrics evaluation of situation 2 for Mobile IP

8.1.5 GRAPHICAL REPRESENTATION OF INSTANTANEOUS THROUGHPUT OF THIS SITUATION

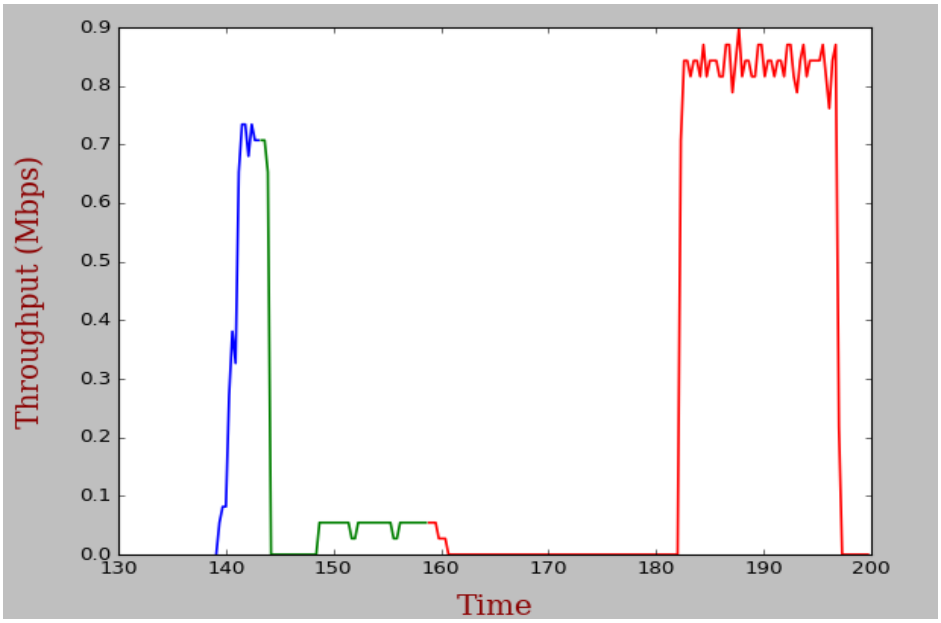

Fig 8.22 Instantaneous Throughput obtained by situation 2 Considering Mobile IP

The graphical representation of throughput is given in above figure 8.5 where x axis represents the time and y axis represent throughput obtained. The blue line plot of the graph is for scenario 1 where MH and RH are present in its home agent. The green line plot of the graph represents the throughput obtained after RH is moving towards the Foreign Agent (FA). The Red line plot in the graph represents the throughput obtained when MH and RH is located in the MH's Home agent area. From above it obvious that during communication if MH and RH are present in the MH's Home location then higher throughput can be achieved.

CHAPTER 9

SOURCE CODES

9.1 TCL SCRIPT FOR SIMULATING WIRED CUM WIRELESS NETWORK

```
set ns  [new Simulator]


# open the Trace and nam files

set tf  [open Trace_out.tr w]
$ns trace-all $tf


set nf [open nam_out.nam w]
$ns namtrace-all $nf
$ns namtrace-all-wireless $nf 610 610

#Trace file for plotting congestion window

set f1 [open tahoeCW.tr w]
set f2 [open renoCW.tr w]
set f3 [open vegasCW.tr w]

#Trace file for plotting Throughput

set f0 [open res0.tr w]

set f4 [open tahoeTput.tr w]
set f5 [open renoTput.tr w]
set f6 [open vegasTput.tr w]

set f7 [open scene1.tr w]
set f8 [open scene2.tr w]



#set up for hierarchical routing



$ns node-config -addressType hierarchical
AddrParams set domain_num_ 4  ;
lappend cluster_num  1 3 1 1
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 3 2 3 3
AddrParams set nodes_num_ $eilastlevel
```

```
set num_wired_nodes  2
set node_adds { 0.0.0 1.0.0 }

for {set i 0} {$i < $num_wired_nodes} {incr i} {

      set W($i) [$ns node [ lindex $node_adds $i ]]

}

$ns duplex-link $W(0) $W(1) 7Mb 27ms DropTail
$ns duplex-link-op $W(0) $W(1) orient left
$ns queue-limit $W(0) $W(1) 16



global opt

set opt(chan)        Channel/WirelessChannel
set opt(prop)        Propagation/TwoRayGround
set opt(netif)       Phy/WirelessPhy
set opt(mac)         Mac/802_11
set opt(ifq)         Queue/DropTail/PriQueue
set opt(ll)          LL
set opt(ant)         Antenna/OmniAntenna
set opt(x)              610
set opt(y)              610
set opt(ifqlen)          10
set opt(nn)              9
set opt(adhocRouting)    DSDV
set opt(bs_nn)           4


Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 0.9
#Antenna/OmniAntenna set Gt_ 1.0
#Antenna/OmniAntenna set Gr_ 1.0

Phy/WirelessPhy set Pt_       0.2818    ;
Phy/WirelessPhy set RXThresh_ 8.9062e-9 ;

Mac/802_11 set dataRate_ 1.4Mb
LL set delay_ 1us



# configuration and creation of Base Station (Access point)

set topo   [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# god needs to know the number of all wireless interfaces

create-god [ expr $opt(nn) + $opt(bs_nn) ]
```

```
$ns node-config  -adhocRouting $opt(adhocRouting) \
                 -mobileIP ON \
                  -llType $opt(ll) \
                  -macType $opt(mac) \
                  -ifqType $opt(ifq) \
                  -ifqLen $opt(ifqlen) \
                  -antType $opt(ant) \
                  -channel [new $opt(chan)] \
                  -propInstance [new $opt(prop)] \
                  -phyType $opt(netif) \
                  -topoInstance $topo
                  -wiredRouting ON \
                  -agentTrace ON \
                  -routerTrace OFF \
                  -movementTrace OFF \
                  -macTrace ON

set bs_adds { 1.1.0 1.2.0 2.0.0 3.0.0 }

for {set i 0} { $i < $opt(bs_nn) } {incr i} {

      set BS($i) [$ns node [ lindex $bs_adds $i ] ]
      $BS($i) shape box
      $BS($i) random-motion 0 ; #disable the random motion

}

#Locating the base station nodes in topography

$BS(0) set X_ 10.0
$BS(0) set Y_ 400.0
$BS(0) set Z_ 0.0

$BS(1) set X_ 210.0
$BS(1) set Y_ 400.0
$BS(1) set Z_ 0.0


$BS(2) set X_ 411.0
$BS(2) set Y_ 400.0
$BS(2) set Z_ 0.0


$BS(3) set X_ 550.0
$BS(3) set Y_ 400.0
$BS(3) set Z_ 0.0


$ns duplex-link $W(1) $BS(0) 3Mb 50ms DropTail
$ns duplex-link $W(1) $BS(1) 3Mb 50ms DropTail
$ns duplex-link $W(0) $BS(2) 2.5Mb 22ms DropTail
$ns duplex-link $W(0) $BS(3) 2.5Mb 22ms DropTail

$ns queue-limit $W(1) $BS(0) 5
$ns queue-limit $W(1) $BS(1) 5
$ns queue-limit $W(0) $BS(2) 8
$ns queue-limit $W(0) $BS(3) 8
```

```
# configuration and creation of Mobile Nodes

$ns node-config -wiredRouting OFF

set temp { 1.1.1 1.1.2 1.2.1 1.2.2 2.0.1 2.0.2 2.0.3 3.0.1 3.0.2 }

set y 0
set j 0
set xVal { 10 40 210 250 410 430 459 550 590 }
set yVal { 375 375 375 375 380 370 380 375 375 }

while {$j < 9} {

        set MH($j) [ $ns node [lindex $temp $j ] ]
        set home_addr [AddrParams addr2id [$BS($y) node-addr]]
        [$MH($j) set regagent_] set home_agent_ $home_addr

        #Locating the wireless nodes in topography

        $MH($j) set X_ [lindex $xVal $j ]
        $MH($j) set Y_ [lindex $yVal $j ]
        $MH($j) set Z_ 0.0

        $ns initial_node_pos $MH($j) 20

        if { $j == 1 || $j == 3 || $j == 6 } {
                set y [ expr $y + 1 ]
        }

        $MH($j) color black
        set j [ expr $j + 1 ]

}




$ns color 1 "#000080"; #for udp connection
$ns color 2 "#ff3300" ; #for tcp tahoe connection
$ns color 3 "#00ff99" ; #for tcp Reno connection
$ns color 4 "#0066ff" ; #for tcp Vegas connection
$ns color 5 "#FF5733 " ; #for mip scenario1
$ns color 6 "#ff0066" ; #for mip scenario2



#Create Necessary Transport Agent and attach it to nodes


#UDP Transport layer

set udp [new Agent/UDP]
set null0 [new Agent/LossMonitor]

$ns attach-agent $MH(0) $udp
```

```
$ns attach-agent $MH(8) $null0

$ns connect $udp $null0
$udp set fid_ 1


#TCP Transport layer

set tcp1 [new Agent/TCP]
set sink1 [new Agent/TCPSink]

$ns attach-agent $MH(1) $tcp1
$ns attach-agent $MH(4) $sink1

$ns connect $tcp1 $sink1
$tcp1 set fid_ 2

set tcp2 [new Agent/TCP/Reno]
set sink2 [new Agent/TCPSink]

$ns attach-agent $MH(2)  $tcp2
$ns attach-agent $MH(5)  $sink2

$ns connect $tcp2 $sink2
$tcp2 set fid_ 3

set tcp3 [new Agent/TCP/Vegas]
set sink3 [new Agent/TCPSink]

$ns attach-agent $MH(3) $tcp3
$ns attach-agent $MH(6) $sink3

$ns connect $tcp3 $sink3
$tcp3 set fid_ 4

set tcp4 [new Agent/TCP]
set sink4 [new Agent/TCPSink]

$ns attach-agent $MH(1) $tcp4
$ns attach-agent $MH(7) $sink4

$ns connect $tcp4 $sink4
$tcp4 set fid_ 5

set tcp5 [new Agent/TCP/Vegas]
set sink5 [new Agent/TCPSink]

$ns attach-agent $MH(0) $sink5
$ns attach-agent $MH(4) $tcp5

$ns connect $tcp5 $sink5
$tcp5 set fid_ 6



# Creating Traffic Agent and Scheduling Events
```

```
proc udp_traffic { } {

        global ns udp MH

        set cbr [new Application/Traffic/CBR]
        $cbr set packetSize_ 2000 ;
        $cbr set rate_ 512Kb
        $cbr attach-agent $udp

        $ns at 0.5 "$MH(8) color #000080"
        $ns at 0.5 "$MH(0) color #000080"

        puts "UDP Connection in wired-Cum-Wireless Network"

        $ns at 0.5 "$cbr start"
        $ns at 45.5 "$cbr stop"

        $ns at 16.1 "$MH(8) setdest 480 310 100"
        $ns at 30.8 "$MH(8) setdest 590 385 100"

        $ns at 45.6 "$MH(0) color black"
        $ns at 45.6 "$MH(8) color black"
}

$ns at 0.3 "udp_traffic"


proc tcp_traffic { } {

        global ns
        global tcp1 tcp2 tcp3
        global MH

        $ns at 23.6 "$MH(4) color #ff3300"
        $ns at 23.1 "$MH(5) color #00ff99"
        $ns at 23.6 "$MH(6) color #0066ff"

        $ns at 23.6 "$MH(1) color #ff3300"
        $ns at 23.1 "$MH(2) color #00ff99"
        $ns at 23.6 "$MH(3) color #0066ff"

        set ftp1 [new Application/FTP]
        $ftp1 attach-agent $tcp1


        set ftp2 [new Application/FTP]
        $ftp2 attach-agent $tcp2

        set ftp3 [new Application/FTP]
        $ftp3 attach-agent $tcp3

        puts "TCP Connection in wired-Cum-Wireless Network"

        $ns at 23.7 "$ftp1 start"
         $ns at 63.0 "$ftp1 stop"

         $ns at 23.1 "$ftp2 start"
```

```
        $ns at 66.0 "$ftp2 stop"

        $ns at 23.7 "$ftp3 start"
        $ns at 65.0 "$ftp3 stop"

        $ns at 63.3 "$MH(4) color black"
        $ns at 65.3 "$MH(5) color black"
        $ns at 66.3 "$MH(6) color black"

        $ns at 63.3 "$MH(2) color black"
        $ns at 65.3 "$MH(3) color black"
        $ns at 66.3 "$MH(1) color black"

}

$ns at 22.0 "tcp_traffic"

proc mip1 { } {

        global ns tcp4 MH

        set ftp4 [new Application/FTP]
        $ftp4 attach-agent $tcp4

        $ns at 67.6 "$MH(1) add-mark m1 #DC143C circle"
        $ns at 67.6 "$MH(7) add-mark m2 #DC143C circle"

        #downloading file of size 3.5MB

        puts "Mobile node moving while receicving data ..."

        set filesize [ expr 3.5*1024*1024 ]
        $ns at 68.0 "$ftp4 send $filesize"

        $ns at 98.5 "$MH(7) setdest 230 364 20"

        $ns at 138.0 "$MH(1) delete-mark m1"
        $ns at 138.0 "$MH(7) delete-mark m2"

}

$ns at 67.2 "mip1"


proc mip2 { } {

        global ns tcp5 MH

        set ftp5 [new Application/FTP]
        $ftp5 attach-agent $tcp5

        $ns at 138.9 "$MH(0) add-mark m1 #00FA9A circle"
        $ns at 138.9 "$MH(4) add-mark m2 #00FA9A circle"

        #uploading file of size 1.8MB

        puts "Mobile node moving while sending data ..."
```

```
        set filesize [ expr 1.8*1024*1024 ]
        $ns at 139.0 "$ftp5 send $filesize"


        $ns at 143.0 "$MH(4) setdest 230 350 80"
        $ns at 158.8 "$MH(4) setdest 50 400 70"

        $ns at 198.0 "$MH(0) delete-mark m1"
        $ns at 198.0 "$MH(4) delete-mark m2"

}

$ns at 138.6 "mip2"



#Collecting data for plotting congestion window TCP protocol

proc congestionWindow {} {
        global ns
        global tcp1 tcp2 tcp3
        global f1 f2 f3
        set now [$ns now]
        set time 0.01
        set cwnd1 [$tcp1 set cwnd_]
        puts $f1 "$now $cwnd1"
        set cwnd2 [$tcp2 set cwnd_]
        puts $f2 "$now $cwnd2"
        set cwnd3 [$tcp3 set cwnd_]
        puts $f3 "$now $cwnd3"
        $ns at [expr $now+$time] "congestionWindow"
}

$ns at 22.0 "congestionWindow"


#Collecting data for ploting graph between Throughput vs Time


proc throughput {} {

        global null0 sink1 sink2 sink3 sink4 sink5
        global f0 f4 f5 f6 f7 f8
        global ns

        set time 0.3
        set now [$ns now]

        set bw0 [$sink1 set bytes_]
        set bw1 [$sink2 set bytes_]
        set bw2 [$sink3 set bytes_]
        set bw3 [$null0 set bytes_]
        set bw4 [$sink4 set bytes_]
        set bw5 [$sink5 set bytes_]

        puts $f4 "$now [expr $bw0/$time*8/1000000]"
        puts $f5 "$now [expr $bw1/$time*8/1000000]"
```

```
        puts $f6 "$now [expr $bw2/$time*8/1000000]"
        puts $f0 "$now [expr $bw3/$time*8/1000000]"
        puts $f7 "$now [expr $bw4/$time*8/1000000]"
        puts $f8 "$now [expr $bw5/$time*8/1000000]"

        $sink1 set bytes_ 0
        $sink2 set bytes_ 0
        $sink3 set bytes_ 0
        $null0 set bytes_ 0
        $sink4 set bytes_ 0
        $sink5 set bytes_ 0


        #Re-schedule the procedure
        $ns at [expr $now+$time] "throughput"
}
$ns at 0.2 "throughput"



# To end the simulation and close the files

proc finish {} {

  global ns tf nf f1 f2 f3 f4 f5 f6 f0 f7 f8
  $ns flush-trace
  close $tf
  close $nf
  close $f0
  close $f1
  close $f2
  close $f3
  close $f4
  close $f5
  close $f6
  close $f7
  close $f8
  exit 0

}


# Mentioning Start event and stop event in the simulation

for {set i 0 } {$i < $opt(nn) } {incr i} {
      $ns at 199.5 "$MH($i) reset";
}

for {set i 0 } {$i < $opt(bs_nn) } {incr i} {
      $ns at 199.5 "$BS($i) reset";
}

$ns at 200.0 "finish"
puts "Starting the Simulation ...."
$ns run
```

9.2 AWK SCRIPT FOR FINDING PERFORMANCE METRICS OF UDP CONNECTION

```
BEGIN {

        # variables for pdr calculation
        sendpkts = 0;
        recvpkts = 0;

        # variables for finding tput achieved
        srt_time = 0;
        end_time = 0;
        flag = 0;
        dataRec = 0;

        # variables for finding average e2e delay
        num_samples = 0
        total_delay = 0
}
{
        if (($1 == "s") && ($3 == "_6_") && ($4 == "AGT") && ($7 ==
"cbr"))
        {
                #for pdr calculation
                sendpkts++;

                #for throughput calculation
                if (flag == 0)
                {
                  srt_time = $2;
                  flag = 1;
                }
                end_time = $2;

                #for avg e2e delay calculation
                stime_arr[$6] = $2;
        }

        if ( ($1 == "r") && ($3 == "_14_") && ($4 == "AGT") && ($7 ==
"cbr") )
        {
                #for pdr calculation
                recvpkts++;

                #for throughput calculation
                dataRec += $8;

                #for avg e2e delay calculation
                if (stime_arr[$6] >= 0.0) {
                        ind_delay = 0
                        etime_arr[$6] = $2
                        ind_delay = etime_arr[$6] - stime_arr[$6]
                        total_delay += ind_delay
                        num_samples += 1
```

```
                }

        }

}
END {
      printf("Throughput Results : \n\n");
      delay = end_time-srt_time;
      tput  = (dataRec*8)/(delay*1000000);
      printf("Throughput Obtained = %f Mbps \n\n",tput);


      printf("Packet Delivery Ratio Results : \n\n");
      printf("No of packets Sent       :   %d\n",sendpkts);
      printf("No of packets Recieved   :   %d\n",recvpkts);
      printf("No of packets dropped    :   %d\n",sendpkts-recvpkts);
      printf("Packet delivery Ratio    :   %f\n",recvpkts/sendpkts);


      avg_delay = total_delay/num_samples
      printf("\nCalculating end to end delay:\n")
      printf("\nTotal End-to-End Delay = %f " ,total_delay)
      printf("\nAverage End-to-End Delay = %f \n\n" ,avg_delay)

}
```

9.3 AWK SCRIPT FOR FINDING PERFORMANCE METRICS OF TCP CONNECTION

```
BEGIN {

        # variables for pdr calculation

        tahoe_sp = 0;

        tahoe_rp = 0;


        reno_sp = 0;

        reno_rp = 0;


        vegas_sp = 0;

        vegas_rp = 0;
```

```
# variables for finding tput achieved
th_srt_time = 0;
th_end_time = 0;
th_flag = 0;
th_dataRec = 0;


rn_srt_time = 0;
rn_end_time = 0;
rn_flag = 0;
rn_dataRec = 0;

vg_srt_time = 0;
vg_end_time = 0;
vg_flag = 0;
vg_dataRec = 0;


# variables for finding average e2e delay
th_num_samples = 0;
th_total_delay = 0;

rn_num_samples = 0;
rn_total_delay = 0;

vg_num_samples = 0;
vg_total_delay = 0 ;
```

```
}


{
        if (($1 == "s") && ($2 <= 63.0) && ($3 == "_7_") && ($4 == "AGT")
&& ($7 == "tcp"))
        {
                tahoe_sp++;

                if (th_flag == 0)
                {
                  th_srt_time = $2;
                  th_flag = 1;
                }

                th_stime_arr[$6] = $2;
                th_end_time = $2;
        }

        if (($1 == "s") && ($2 <= 65.0) && ($3 == "_8_") && ($4 == "AGT")
&& ($7 == "tcp"))
        {
                reno_sp++;

                if (rn_flag == 0)
                {
                  rn_srt_time = $2;
                  rn_flag = 1;
                }
```

```
                rn_end_time = $2;


                rn_stime_arr[$6] = $2
        }


        if (($1 == "s") && ($2 <= 66.0) && ($3 == "_9_") && ($4 == "AGT")
&& ($7 == "tcp"))
        {
                vegas_sp++;


                if (vg_flag == 0)
                {
                  vg_srt_time = $2;
                  vg_flag = 1;
                }
                vg_end_time = $2;


                vg_stime_arr[$6] = $2;
        }



        if (($1 == "r") && ($2 <= 63.0) && ($3 == "_10_") && ($4 == "AGT")
&& ($7 == "tcp"))
        {
                   tahoe_rp++;


                   th_dataRec += $8;


                   if (th_stime_arr[$6] >= 0.0) {
```

```
                                th_etime_arr[$6] = $2

                                th_ind_delay    =    th_etime_arr[$6]    -
th_stime_arr[$6]

                                th_total_delay += th_ind_delay

                                th_num_samples += 1

                        }

        }


        if (($1 == "r") && ($2 <= 65.0) && ($3 == "_11_") && ($4 == "AGT")
&& ($7 == "tcp"))

        {

                        reno_rp++;


                        rn_dataRec += $8;


                        if (rn_stime_arr[$6] >= 0.0) {

                                rn_etime_arr[$6] = $2

                                rn_ind_delay    =    rn_etime_arr[$6]    -
rn_stime_arr[$6]

                                rn_total_delay += rn_ind_delay

                                rn_num_samples += 1

                        }

        }


        if (($1 == "r") && ($2 <= 66.0) && ($3 == "_12_") && ($4 == "AGT")
&& ($7 == "tcp"))

        {

                        vegas_rp++;


                        vg_dataRec += $8;
```

```
                    if (vg_stime_arr[$6] >= 0.0) {
                         vg_etime_arr[$6] = $2
                         vg_ind_delay = vg_etime_arr[$6] - vg_stime_arr[$6]
                         vg_total_delay += vg_ind_delay
                         vg_num_samples += 1
                    }
          }
}

END {

          printf("\tTransport Layer -----> TCP Tahoe\n\n");

          printf("Throughput Results : \n");
          delay = th_end_time-th_srt_time;
          tput  = (th_dataRec*8)/(delay*1000000);
          printf("Throughput Obtained = %f Mbps \n\n",tput);

          printf("Packet Delivery Ratio Results : \n");
          printf("No of packets Sent      :   %d\n",tahoe_sp);
          printf("No of packets Recieved   :   %d\n",tahoe_rp);
          printf("No of packets dropped   :   %d\n",tahoe_sp-tahoe_rp);
          printf("Packet delivery Ratio   :   %f\n",tahoe_rp/tahoe_sp);

          avg_delay = th_total_delay/th_num_samples
          printf("\nCalculating end to end delay :\n")
          printf("Total End-to-End Delay = %f ms " ,th_total_delay*1000)
          printf("\nAverage End-to-End Delay = %f ms \n\n" ,avg_delay*1000)
```

```
printf("\tTransport Layer -----> TCP Reno\n\n")


printf("Throughput Results : \n");
delay = rn_end_time-rn_srt_time;
tput  = (rn_dataRec*8)/(delay*1000000);
printf("Throughput Obtained = %f Mbps \n\n",tput);


printf("Packet Delivery Ratio Results : \n");
printf("No of packets Sent      :   %d\n",reno_sp);
printf("No of packets Recieved   :   %d\n",reno_rp);
printf("No of packets dropped   :   %d\n",reno_sp-reno_rp);
printf("Packet delivery Ratio   :   %f\n",reno_rp/reno_sp);


avg_delay = rn_total_delay/rn_num_samples
printf("\nCalculating end to end delay:\n")
printf("Total End-to-End Delay = %f ms " ,rn_total_delay*1000)
printf("\nAverage End-to-End Delay = %f ms \n\n" ,avg_delay*1000)




printf("\tTransport Layer -----> TCP Vegas\n\n")


printf("Throughput Results : \n");
delay = vg_end_time-vg_srt_time;
tput  = (vg_dataRec*8)/(delay*1000000);
```

```
        printf("Throughput Obtained = %f Mbps \n\n",tput);


        printf("Packet Delivery Ratio Results : \n");
        printf("No of packets Sent      :   %d\n",vegas_sp);
        printf("No of packets Recieved   :   %d\n",vegas_rp);
        printf("No of packets dropped   :   %d\n",vegas_sp-vegas_rp);
        printf("Packet delivery Ratio   :   %f\n",vegas_rp/vegas_sp);


        avg_delay = vg_total_delay/vg_num_samples
        printf("\nCalculating end to end delay:\n")
        printf("Total End-to-End Delay = %f ms " ,vg_total_delay*1000)
        printf("\nAverage End-to-End Delay = %f ms \n\n" ,avg_delay*1000)



}
```

9.4 AWK SCRIPT FOR FINDING PERFORMANCE METRICS OF MOBILE IP SITUATION 1

```
BEGIN {
        # variables for pdr calculation
        sendpkts = 0;
        recvpkts = 0;

        # variables for finding tput achieved
        srt_time = 0;
```

```
        end_time = 0;
        flag = 0;
        dataRec = 0;

        # variables for finding average e2e delay
        num_samples = 0
        total_delay = 0

}

{
        if ( ($1 == "s") && ($2 >= 68.0) && ($3 == "_7_")   && ($4 ==
"AGT") && ($7 == "tcp") )
        {
                sendpkts++;

                if (flag == 0)
                {
                  srt_time = $2;
                  flag = 1;
                }
                end_time = $2;
                stime_arr[$6] = $2;

        }
        if ( ($1 == "r") && ($2 >= 68.0) && ($3 == "_13_") && ($4 ==
"AGT") && ($7 == "tcp") )
        {
                recvpkts++;
```

```
                dataRec += $8;


                if (stime_arr[$6] >= 0.0) {
                        etime_arr[$6] = $2
                        ind_delay = etime_arr[$6] - stime_arr[$6]
                        total_delay += ind_delay
                        num_samples += 1
                }
        }

}

END {

      printf("Throughput Results : \n\n");
      delay = end_time-srt_time;
      tput  = (dataRec*8)/(delay*1000000);
      printf("Throughput Obtained = %f Mbps \n\n",tput);

      printf("Packet Delivery Ratio Results : \n\n");
      printf("No of packets Sent       :   %d\n",sendpkts);
      printf("No of packets Recieved   :   %d\n",recvpkts);
      printf("No of packets dropped    :   %d\n",sendpkts-recvpkts);
      printf("Packet delivery Ratio    :   %f\n",recvpkts/sendpkts);

      avg_delay = total_delay/num_samples
      printf("\nCalculating end to end delay:\n")
      printf("\nTotal End-to-End Delay = %f ms " ,total_delay*1000)
      printf("\nAverage End-to-End Delay = %f ms \n\n" ,avg_delay*1000)
```

```
}
```

9.5 AWK SCRIPT FOR FINDING PERFORMANCE METRICS OF MOBILE IP SITUATION 2

```
BEGIN {
        # variables for pdr calculation
        sendpkts = 0;
        recvpkts = 0;

        # variables for finding tput achieved
        srt_time = 0;
        end_time = 0;
        flag = 0;
        dataRec = 0;

        # variables for finding average e2e delay
        num_samples = 0
        total_delay = 0

}

{
        if ( ($1 == "s") && ($2 >= 159.0) && ($3 == "_10_")   && ($4 ==
"AGT") && ($7 == "tcp") )
        {
                sendpkts++;
                if (flag == 0)
                {
                  srt_time = $2;
                  flag = 1;
                }
                end_time = $2;

                stime_arr[$6] = $2;

        }

        if ( ($1 == "r") && ($2 >= 159.0) && ($3 == "_6_") && ($4 ==
"AGT") && ($7 == "tcp") )
        {
                recvpkts++;
                dataRec += $8;
                if (stime_arr[$6] >= 0.0) {
                        etime_arr[$6] = $2
                        ind_delay = etime_arr[$6] - stime_arr[$6]
                        total_delay += ind_delay
                        num_samples += 1
                }
```

```
        }

}
END {
      printf("Throughput Results : \n\n");
      delay = end_time-srt_time;
      tput  = (dataRec*8)/(delay*1000000);
      printf("Throughput Obtained = %f Mbps \n\n",tput);

      printf("Packet Delivery Ratio Results : \n\n");
      printf("No of packets Sent       :   %d\n",sendpkts);
      printf("No of packets Recieved   :   %d\n",recvpkts);
      printf("No of packets dropped    :   %d\n",sendpkts-recvpkts);
      printf("Packet delivery Ratio    :   %f\n",recvpkts/sendpkts);

      avg_delay = total_delay/num_samples
      printf("\nCalculating end to end delay:\n")
      printf("\nTotal End-to-End Delay = %f ms " ,total_delay*1000)
      printf("\nAverage End-to-End Delay = %f ms \n\n" ,avg_delay*1000)

}
```

9.6 PYTHON CODE FOR PLOTTING GRAPH

```
import matplotlib.pyplot as plt

class DataPlotter:
    """ This Class is created for plotting the graph between Network Parameters """

    font1 = {'family':'serif','color':'indigo','size':20}
    font2 = {'family':'serif','color':'darkred','size':18}

    def cwndPlot(self,file,algo_name):
        """ For PLotting the congestion window for 3 different Congestion Algorithms """

        xpoints  = []
        ypoints  = []

        with open(file,'r') as fp:
            for i in fp:
                z = i.split()
                if(float(z[0]) < 68.0):
                    xpoints.append(float(z[0]))
                    ypoints.append(float(z[1]))

        plt.plot(xpoints, ypoints,color = "red",linewidth = 1.5,linestyle = '-')
        ax = plt.axes()
        ax.patch.set_facecolor('khaki')
        heading = "Congestion Window Size in TCP " + algo_name
        plt.title(heading,fontdict=self.font1,fontweight='bold')
```

```
    plt.xlabel("Time",labelpad=10,fontdict=self.font2)
    plt.ylabel("cwnd (pkts)",labelpad=25,fontdict=self.font2)

    plt.show()

def tcp_tput(self,file,algo_name):

    """ For PLotting the Instantaneous Throughput for 3 different Congestion Algorithms """

    xpoints = []
    ypoints = []
    with open(file,'r') as fp:
        for i in fp:
            z = i.split()
            if (float(z[0]) >= 21.0 and float(z[0]) < 68.8):
                xpoints.append(float(z[0]))
                ypoints.append(float(z[1]))

    plt.plot(xpoints,ypoints,color="Crimson",linewidth=1.5,linestyle = '-')
    heading = "Instanteous Throughput of TCP " + algo_name
    plt.title(heading,fontdict=self.font1,fontweight='bold')
    plt.xlabel("Time",labelpad=10,fontdict=self.font2)
    plt.ylabel("Throughput (Mbps)",labelpad=25,fontdict=self.font2)
    plt.show()


def udp_tput(self,file):

    """ For PLotting the Instantaneous Throughput for UDP Connection """

    xp1  = []
    xp2  = []
    xp3  = []

    yp1  = []
    yp2  = []
    yp3  = []

    #storing x and y points
    with open(file,'r') as fp:
        for i in fp:
            z = i.split()

            if (float(z[0]) < 16.1):
                xp1.append(float(z[0]))
                yp1.append(float(z[1]))

            elif (float(z[0]) >= 16.1 and float(z[0]) < 30.8):
                xp2.append(float(z[0]))
                yp2.append(float(z[1]))

            elif (float(z[0]) >= 30.8 and float(z[0]) <= 50.0):
                xp3.append(float(z[0]))
                yp3.append(float(z[1]))
```

```
        plotdata = []
        plotdata.append((xp1,yp1,'MH in Home Agent'))
        plotdata.append((xp2,yp2,'MH is in out of coverage area'))
        plotdata.append((xp3,yp3,'MH  again in Home Agent'))


        for k in plotdata:
            plt.plot(k[0],k[1],label=k[2],linewidth = 1.5,linestyle = '-')


        plt.title("Instanteous Throughput of UDP Connection",fontdict=self.font1,fontweight='bold')
        plt.xlabel("Time",labelpad=10,fontdict=self.font2)
        plt.ylabel("Throughput (Mbps)",labelpad=25,fontdict=self.font2)
        plt.show()

    def mipScenario1(self,file):

        """ For PLotting the Instantaneous Throughput for mip Scenario1  """

        xp1  = []
        xp2  = []
        yp1  = []
        yp2  = []

        with open(file,'r') as fp:
            for i in fp:
                z = i.split()
                if(float(z[0]) >= 67.0 and float(z[0]) < 98.5):
                    xp1.append(float(z[0]))
                    yp1.append(float(z[1]))

                elif(float(z[0]) >= 98.5):
                    xp2.append(float(z[0]))
                    yp2.append(float(z[1]))


        plotdata = []
        plotdata.append((xp1,yp1,'MH in Home Agent'))
        plotdata.append((xp2,yp2,'MH in Foreign Agent'))

        for k in plotdata:
            plt.plot(k[0],k[1],label=k[2],linewidth = 1.5,linestyle = '-')


        plt.xlabel("Time",labelpad=10,fontdict=self.font2)
        plt.ylabel("Throughput (Mbps)",labelpad=20,fontdict=self.font2)
        plt.show()


    def mipScenario2(self,file):

        """ For PLotting the Instantaneous Throughput for mip Scenario2  """

        xp1  = []
        xp2  = []
```

```
        xp3  = []
        yp1  = []
        yp2  = []
        yp3  = []

        with open(file,'r') as fp:
           for i in fp:
              z = i.split()
              if(float(z[0]) >= 158.8):
                 xp3.append(float(z[0]))
                 yp3.append(float(z[1]))

              elif(float(z[0]) >= 143.0 ):
                 xp2.append(float(z[0]))
                 yp2.append(float(z[1]))

              elif(float(z[0]) >= 139.0):
                 xp1.append(float(z[0]))
                 yp1.append(float(z[1]))

        plotdata = []
        plotdata.append((xp1,yp1,'CN in Home Agent'))
        plotdata.append((xp2,yp2,'CN in Foreign Agent'))
        plotdata.append((xp3,yp3,'CN in CN\'s Home Agent'))

        for k in plotdata:
           plt.plot(k[0],k[1],label=k[2],linewidth = 1.5,linestyle = '-')


        plt.xlabel("Time",labelpad=10,fontdict=self.font2)
        plt.ylabel("Throughput (Mbps)",labelpad=20,fontdict=self.font2)
        plt.show()



dp1 = DataPlotter()

dp1.udp_tput('res0.tr')

dp1.cwndPlot('tahoeCW.tr','Tahoe')
dp1.cwndPlot('renoCW.tr','Reno')
dp1.cwndPlot('vegasCW.tr','Vegas')

dp1.tcp_tput('tahoeTput.tr','Tahoe')
dp1.tcp_tput('renoTput.tr','Reno')
dp1.tcp_tput('vegasTput.tr','Vegas')

dp1.mipScenario1('scene1.tr')

dp1.mipScenario2('scene2.tr')
```

CHAPTER 10

CONCLUSION AND FUTURE PLANS

10.1 CONCLUSION

This project made me to realize the necessity of base station or access point in routing the packets from wireless device to a wired device or vice-versa in a network. It also gave an insight of how Mobile IP solved the shortcomings faced by the IP. I learned that we use Mobile IP constantly in everyday life and it will continue to be a very important protocol as mobility becomes of increasing importance in the network. It introduced us to new programming and scripting languages like Tcl, AWK scripting and python matplotlib (for plotting the graphs) which is useful in the study of numerous application. This project also introduces us to use Linux operating system which considered to be one of most powerful operating system till now.

10.2 FUTURE WORKS

The base Mobile IP allows any mobile node to move about changing its point of attachment to the Internet, while continuing to be identified by its home IP address. The Correspondent nodes send IP datagrams to a mobile node at its home address in the same way as with any other destination. This scheme allows transparent inter operation between mobile node and its correspondent node, but forces all datagrams for a mobile node to be routed through its home agent. This indirect routing delays the delivery of the datagrams to mobile nodes, and places an unnecessary burden on the networks and routers along their paths through the Internet." There is a concept called Route Optimisation in Mobile IP which solves the above issue. So this project can further improved by doing a Comparison how Mobile IP works with and without route optimisation technique on the same designed network.

CHAPTER 11

REFERENCES

[1] Marc Greis, Tutorial for the Network Simulator "ns":

https://www.isi.edu/nsnam/ns/tutorial/

[2] Kevin Fall, Kannan Varadhan. (2011, November 5). The ns Manual:

https://www.isi.edu/nsnam/ns/doc/index.html

[3] Subangkar. (2019, January 14). Wired-cum-Wireless-Network-Simulation-ns2:

https://github.com/Subangkar/Wired-cum-Wireless-Network-Simulation-ns2

[4] Jain, N., & Srivastava, N. (2015). Comparison of TCP I-Vegas with TCP Vegas in Wired-cum-Wireless Network:

https://www.ijser.org/researchpaper/Comparison-of-TCP-I-Vegas-with-TCP-Vegas-in-Wired-cum-Wireless-Network.pdf

[5] Kaniganti, M. C. (2005). *Feasibility of Ns-2 Models in Simulating the Custody Transfer Mechanism* (Doctoral dissertation, Ohio University).

[6] Swanlund, E., Loodu, P., & Chowdhury, S. (2013). Analysis and performance evaluation of a Wi-Fi network using ns-2. *School of Engineering Science, Simon Fraser University*.

[7] Xiang, C. (2008). WIRELESS NETWORK STUDY AND ANALYSIS USING NS2 SIMULATOR.

[8] Romdhani, Imed. (2015). Can we change the node icon in ns2?

https://www.researchgate.net/post/Can_we_change_the_node_icon_in_ns2