A NEW FRAMEWORK FOR MINIMISING HANDOVER IN MULTICAST MOBILITY

Khanista NAMEE

Ph.D. Thesis **2015**

A NEW FRAMEWORK FOR MINIMISING HANDOVER IN MULTICAST MOBILITY

Khanista NAMEE

School of Computing, Science and Engineering

College of Science and Technology

University of Salford, Salford, UK

Submitted in Partial Fulfilment of the Requirements of the Degree of Doctor of Philosophy, 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to deliver my greatest appreciation to my supervisor, Prof. Nigel Linge, for all his support and guidance throughout the research period; without his support this research could not have been completed successfully.

I would also like to thank the Thai government for providing me with the scholarships and living allowances in order to support my quest for a PhD degree. Appreciation would also be given to my office, King Mongkut's University of Technology North Bangkok for granting me the study leave and the opportunity to further my studies in this PhD scholarship.

I would also like to thank all the staff and members at CNTR (Computer Networking and Telecommunications Research) centre for all their support and suggestions. I have really enjoyed my time there with everyone.

Last but not least, I would also like to express my gratitude to my family, parents and friends for their continuous patience and support throughout this period of study.

LIST OF ABBREVIATIONS

3G	Third Generation Wireless (Communications)
3GPP	3^{rd} Generation Partnership Project
3GPP2	3^{rd} Generation Partnership Project 2
3GPP-PSS	3^{rd} Generation Partnership Project – Packet Switched Streaming
AP	Access Point
AR	Access Router
ASM	Any Source Multicast
BCMCS	Broadcast and Multicast Service
BGMP	Border Gateway Multicast Protocol
BSC	Base Station Controller
BTS	Base Station Transceiver System
BU	Binding Update
CBS	Cell Broadcast Service
CBT	Core-Based Trees
CDMA	Code-Division Multiple Access
CN	Core Network
CNTR	Centre of Networking and Telecommunication Research
CoA	Care of Address
CPU	Central Processor Units
DMM	Distributed Mobility Management
DMSP	Designated Multicast Service Provider
DVMRP	Distance-Vector Multicast Routing Protocol
EAP-AKA	Extensible Authentication Protocol method for UMTS Authentication and Key Agreement
EoS	Economies of Scale
FA	Foreign Agent
FMIP6	Fast Mobile IPv6
fps	Frames per Second
FSMs	Finite State Machines
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GPS	Global Positioning System

GSM	Global System for Mobile Communications
GTP	Generic Tunnelling Protocol
HD	High Definition
HDTV	High-definition TV
HMIPv6	Hierarchical Mobile IPv6
HoA	Home Address
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol version 6
IETF	Internet Engineering Task Force
IG	Integration Gateway
IGMP	Internet Group Management Protocol
IGMPv2	Internet Group Management Protocol version 2
IMS	Internetworking Management System
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISHO	Inter-System Handover
ITU-T	International Telecommunications Union - Telecommunication Standardization Sector
LMA	Local Mobility Anchor
MAC	Media Access Control
MAG	Mobile Access Gateway
MBMS	Multimedia Broadcast/Multicast Service
MBONE	Multicast Backbone
Mbps	Megabit per second
MDP	Markov Decision Process
MIH	Media Independent Handover
MIPv6	Mobile Internet Protocol version 6
MLD	Multicast Listener Discovery
MOM	Mobile Multicast Protocol
MOSPF	Multicast Open Shortest Path First
MPEG-2	Motion Picture Experts Group 2
MPEG-4	Motion Picture Experts Group 4
MR	Multicast Router

MSP	Multicast Service Provider
ND	Neighbour Discovery
NIA	Network Interoperating Agent
NS-2	Network Simulator version 2
OMNET++	Objective Modular Network Testbed in C++
PIM	Protocol Independent Multicast
PIM-DM	Protocol-Independent Multicast Dense Mode
PIM-SM	Protocol-Independent Multicast Sparse Mode
PMIPv6	Proxy Mobile IPv6
PTM	Point to Multipoint
PTP	Point to Point
QoE	Quality of Experience
QoS	Quality of Service
RFC	Request for Comments
RNC	Radio Network Controller
RP	Rendezvous Point
RSSI	Received Signal Strength Indicator
SAP	Session Announcement Protocol
SDTV	Standard-definition TV
SGSN	Serving GPRS Support Node
SIP	Session Initial Protocol
SNMP	Simple Network Management Protocol
SPT	Shortest Path Tree
SSM	Source-Specific Multicast
STP	Spanning Tree Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TV	Television
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Terrestrial Radio Access Network
VoD	Video on Demand
WiFi	Wireless Fidelity

ABSTRACT

Nowadays, mobile devices support a variety of multimedia applications such as live video, radio or online gaming. People spend their time on mobile devices for entertainment more and more via the internet. Due to the requirements of multimedia applications over wireless communication those applications require a huge bandwidth on the network to support them, which creates problems for the network provider. However, one pattern that is appropriate for the efficient delivery of multimedia messages is multicast delivery.

Multicast services do, however, introduce challenges within the network when the recipients of the service are moving. Powerful multicast routing protocols are designed for static client IP addresses. Hence, when the mobile node changes the location, it introduces the problem of access network handover. Therefore, this is the aim of the research where a new framework will be developed for multicast mobility within WiFi network to reduce and provide smooth mobility when handover occurs. This research is focused on techniques to reduce handover latency, minimize packet loss and provide connection when a user moves between network zones.

To achieve these aims, this designed framework lets mobile nodes send the message to register to foreign agents in advance for addressing IP address of the new zone and to establish the multicast tree earlier. Moreover, there are processes that keep the connection of the path alive.

The framework is being simulated on OPNET Modeler for evaluating the performance in terms of handover latency time, the number of packet loss and so on. There are many scenarios that have been tested. According to the results, it shows that the new framework has reduced handover latency time around 60% on average and minimized packet delay approximately 0.7 - 150 ms on mobile node depending on network topology. This framework can provide IP address reconfiguration, binding update, joining multicast group and distribution path of multicast tree in advance. However, there are some overheads and cost that this framework has to pay for such as IP address database, increasing broadcast within networks and keeping connection path alive.

Chapter 1 Introduction

1.1 Background

Today's mobile device supports a variety of multimedia applications such as live video, radio or online gaming. People spend their time on their devices for personal entertainment more and more. There are statistics which show that on average, people are spending 2.7 hours per day on the internet via smart phone [1].

The smart phones are characterized by small screens, limited CPU power and memory. Due to the requirements of multimedia applications over wireless communication those applications require a huge amount of bandwidth from the network to support them, which creates problems for the network provider and also affects to QoE (Quality of Experience) of end user for multimedia services.

However, one pattern that is appropriate for the efficient delivery of multimedia messages is multicast. For instance, transmitting live video data from a media server using multicast allows for a single data stream to be simultaneously sent to several users thereby offering a considerable bandwidth saving over sending multiple separate data streams.

Multicast services do, however, introduce challenges within the network when the recipients of the service are moving. Current multicast routing protocol standards are designed for static client IP addresses. Not only does mobile node movement introduce the problem of access network handover, but also, when considering devices could result in them switching between access network hotspots. Support for this requires an efficient solution to be found for managing multicast services where mobile devices have the capability of operating over several different networks.

Therefore, this research is investigating multicast mobility handover within WiFi networks. The aim of this PhD research is to propose a new method and framework that tries to provide smooth mobility in homogeneous networks such as within WiFi zones.

1.2 Research Aim

The research has the aim to propose a new technique that provides efficient mobility for multicast services in WiFi networks. This research is focused on techniques to reduce handover latency, multicast address management and provide a connection when a user moves between networks.

This research is focused on techniques to reduce handover latency, which comprises multicast handover delay, end-to-end delivery delay and to minimize packet loss. For multicast routing, the research will focus on how to maintain the multicast session for mobile nodes and manage multicast group memberships thereby also minimising packet delay.

1.3 Research Objectives

To achieve this research aim, four objectives have been acknowledged. These research objectives are:

1. To understand, analyse and identify major issues in multicast mobility and focus on issues, which are related to the handover process.
2. To propose the technique that can support multicast mobility when a mobile node changes the zone within WiFi network environments.
3. Implement and evaluate, within a simulation environment, that part of the framework that handles WiFi zone handover.
4. To publish the results of the research and write up a PhD thesis.

1.4 Research Questions

During the research process, the following key research questions will be addressed:

1. How can we reduce latency time in a multicast stream when handover occurs?
2. What performance advantages can be gained over existing schemes for mobile multicast delivery?

1.5 Contribution to Knowledge

Designing a new framework for minimizing multicast mobility within WiFi networks by focusing on techniques to reduce handover latency, packet delay and maintain multicast services for mobile nodes. The key protocols that had been modified in the framework are Mobile IP, IGMP, PIM and ICMP protocols. The key novel features of this new framework are that firstly, mobile nodes register with each foreign network that is within range, secondly a Care of Addresses is obtained for each foreign network, thirdly that the associated multicast trees are established but finally, that the chosen foreign network and multicast tree is only activated once handover is confirmed. In this way, the new framework is able to minimise handover delay and loss of connectivity when handover is taking place.

1.6 Overview of Research Methodology

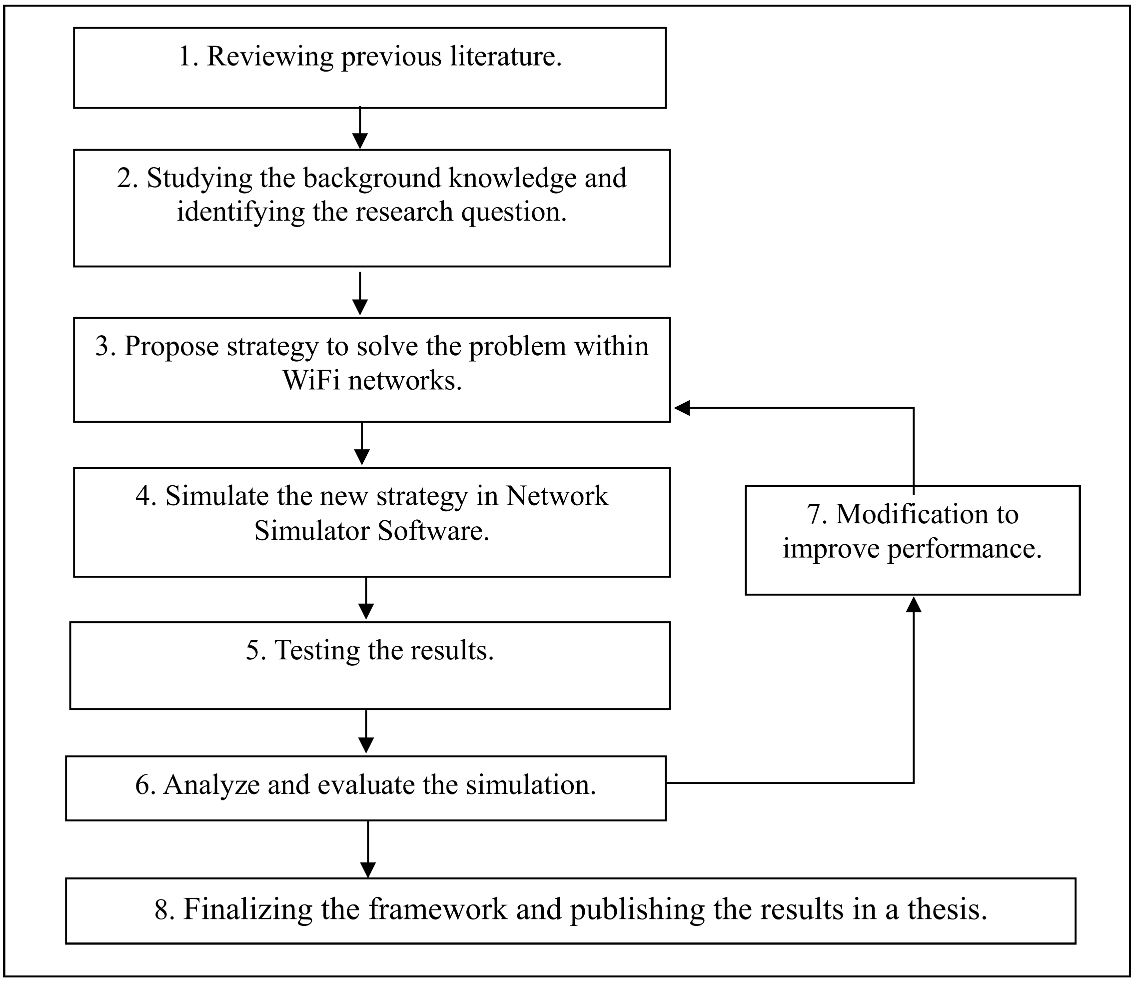

Figure 1-1 Research Methodology

In order to complete this research, the following research methodology has been adopted as shown in Figure 1-1.

The details of each step are described as follows:

1. *Reviewing previous literature* – More than 150 research articles have been reviewed. At this stage, the researcher has gained more knowledge about the problems in multicast mobility, and the mechanism of handover.
2. *Studying the background knowledge and identifying the research question* – The literature review confirms the importance and relevance of the research field and shows where gaps exist within the knowledge. These gaps then help to formulate the research question for this research.
3. *Propose strategy to solve the problem within WiFi networks* – From the background knowledge, literature review, and research question it is then possible to start to propose and design a new framework for improving the performance and QoS guarantees of multicast mobility.
4. *Simulate the new strategy in network simulator software* – The first stage of the framework will be evaluated for use within WiFi networks only. This allows the basic concepts of the framework to be evaluated and as necessary, modified.
5. *Testing the results* – The detailed simulation results will allow the performance of the framework to be quantified and areas for improvement identified across a broad range of networking scenarios.
6. *Analyze and evaluate the simulation* – The outputs from the simulation will be analysed in detail to assess performance and to compare to standard framework.
7. *Modification to improve performance* – Depending upon the simulation results, further refinement and modification of the scheme will follow as necessary.
8. *Finalizing the framework and publishing the results in a thesis* – The results of all of the simulation studies will allow for a final design to be confirmed and this, together with the contribution to knowledge of the research, will be presented within a PhD thesis.

1.7 Structure of the Thesis

The rest of the thesis is organized in the following chapters.

- **Chapter 2** presents the literature review concentrating on the concept of the multicast, multicast mobility, multicast handover issues within wireless networks.
- **Chapter 3** presents the concept and theory idea design of the research, which is to combine network architecture, protocol overview, connection management and modified protocol message.
- **Chapter 4** consists of framework simulation details which have been implemented within OPNET Modeler.
- **Chapter 5** will present the network scenario, simulation result from OPNET Modeler software and performance evaluation.
- **Chapter 6** consists of conclusion and recommendations for further research.

Chapter 2 Literature Survey

2.1 Introduction

This chapter provides an introduction and a summary of the key literature that has been consulted for this research. Creating a framework to provide multicast services for mobile nodes in a wireless communication network is a challenging issue.

The relevant published research has been surveyed in the fields of multicast delivery, multicast mobility and multicast handover within WiFi networks. This framework can support both IPv4 and IPv6 WiFi environments, so the key protocol information and literature review in this chapter will be discussed and focuses on IPv4 and IPv6 environments.

This chapter begins with an overview of multicast delivery concept and protocols. It will then go on to a review of multicast mobility protocols and problems within WiFi networks. After that, the processes and problems of multicast handover are examined. Following this, details about multicast handover issues and approaches will be investigated. Finally, a brief summary is given for the chapter.

2.2 Multicast Delivery

In the IP network system there are three types of communication in a network:

- ***Unicast Delivery*** – one source is sending a single transmission of data directly to one destination.

- ***Broadcast Delivery*** – one source is sending the data to all destinations in the network.

- ***Multicast Delivery*** – one source is sending the data to a select group of destinations.

In traditional computer networks, data is typically sent from a source node to a destination node known as unicast delivery, which is suitable for most applications in the network; alternatively, one source node can transmit a copy data to all end point destination nodes and then the destination nodes will decide if they want to use those data or not known as broadcast delivery. One benefit of broadcasting is that it reduces loads at source node from duplicating data that are sent for multiple destination nodes. The source node sends only one copy of the data to the broadcast address and then the network devices on the network will duplicate the data and transmit to cover the network. However, there are some kinds of application that need to send the same data to multiple destination nodes such as multipoint videoconferencing, online gaming and live TV. It will waste bandwidth on the network if it uses the process of unicast delivery to support those applications. Hence, the aim of multicast delivery is to deliver data packets between one source to multiple destinations more efficiently.

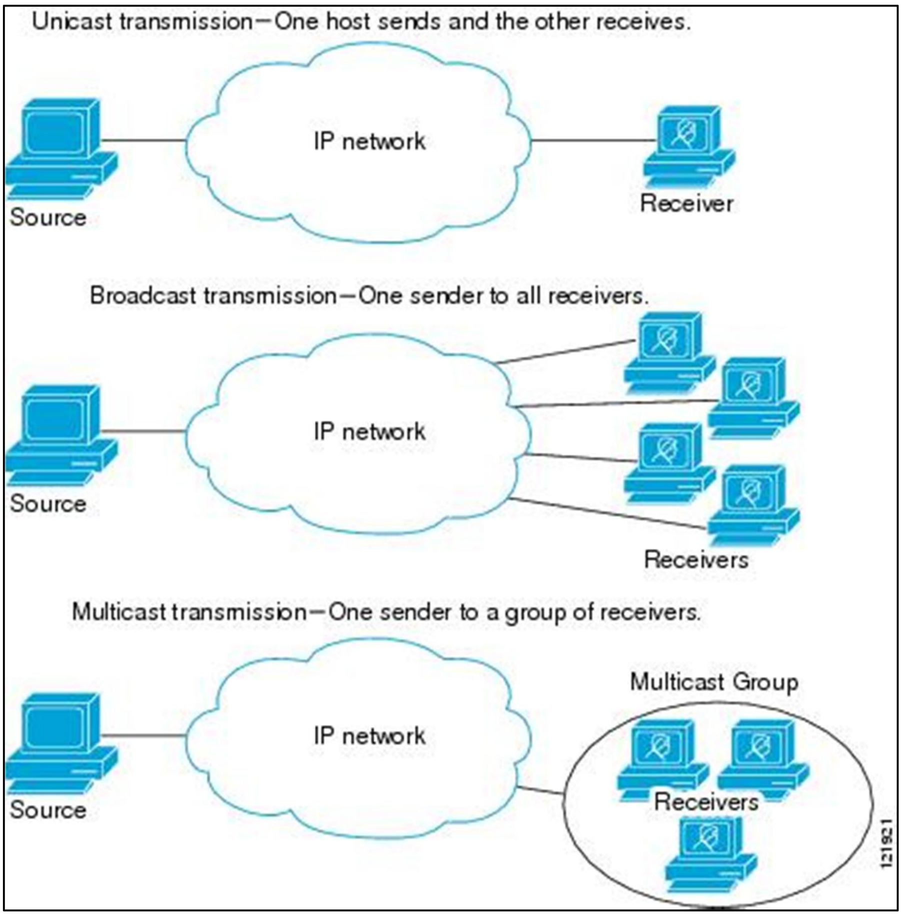

Figure 2-1 Comparison between Unicast, Broadcast and Multicast transmission [2]

The concept of multicast delivery was introduced in the late 1980s by Stephen Deering [3, 4]. The idea is to transmit a single copy of IP packet to a group of destinations, which is identified by a same multicast IP address. A main factor in multicasting is bandwidth efficiency in the network. The multicast functions and protocols have evolved over time as refined in RFC 3376 [5], RFC 4604 [6] and so on.

Multicast is the delivery of data packet to a group of user devices using a common IP multicast destination address. When a multicast tree is set up, the source starts sending IP datagram to the host group address. Then, the network devices take on the responsibility for sending the IP datagram to all destinations within multicast group. Multicast routers along the path are responsible for ensuring that datagrams are transmitted over the appropriate links to ensure they reach all hosts of the multicast group.

The process of coping datagrams occurs only when paths diverge at a multicast router, thus reducing the bandwidth consumption on the network. Figure 2-2 is shown an example of multicast delivery in a TCP/IP (Transmission Control Protocol / Internet Protocol) network. The arrows represent the direction of multicast delivery packets that are sent to host B, host D and host E in the network.

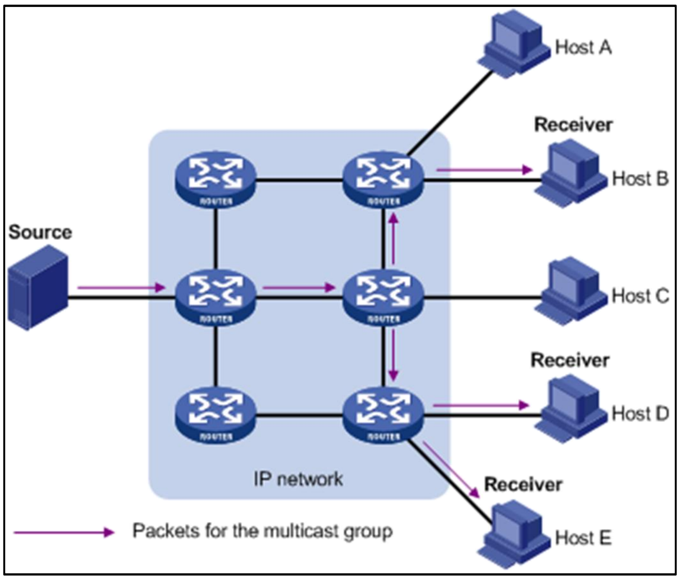

Figure 2-2 Multicast delivery [7]

In networking, the term multicast is synonymous with IP multicast. Multicast delivery is a technique for delivery datagrams from one source to multiple destinations over the network. The optimal distribution paths are created after the member in the multicast group joins in the multicast tree in real time. Multicast delivery technique and group management can support a large number of destination nodes without needing to know how many destination nodes there are. Multicast group requires the source node to send a request to join message only once, even if there are a large number of receive nodes to be sent. IP multicast has an efficient process to maintain the members within the multicast group.

The model of the multicast group is known as an open and dynamic group. This means that, the source node can send multicast packets at any time when it is ready, with no need to announce, register or schedule transmission. The only one thing that the source node needs to know is a multicast address. Also, it is not necessary for the source node to know about group membership in advance. The host members in the multicast group can join or leave a multicast group at any time. However, source and destination multicast nodes can communicate to each other via IP multicast group address. Sources use the multicast IP address as the destination address in their sending data stream. Destination hosts use multicast IP address to notify the network device that they want to receive those packets that are sent to this multicast tree. However, the destination host has to send "*join message*" to be a member of the multicast group first.

In order to deliver multicast data stream, the network creates *a "multicast tree"*. The multicast tree construction is begun with network devices close to the destination nodes and is thus receiver-driven. The multicast tree is built for that group once there are members in a particular IP multicast group and maintained by "*multicast routing protocol"*. There are several kinds of multicast routing protocol depending on the network. Also, each one of them has its own unique method and technique. More information about multicast routing protocols will be given in a later section.

2.2.1 Benefits of Multicast Delivery

The main benefit of multicast delivery is a reduced bandwidth requirement on the network, because this technique sends a single copy of data stream along any link that forms the route.

Live multimedia services such as video, radio, television etc., are delivered as multimedia streaming services. These services require large bandwidth allocations and if they are delivered as separate streams for each user, this will place huge demands on network resources. Hence, multicast delivery can help to control network traffic and reduce this problem [8]. This advantage is known as optimized performance, because multicast delivery eliminates traffic redundancy on the network. Figure 2-3 illustrates a multicast tree, which is represented in Figure 2-2.

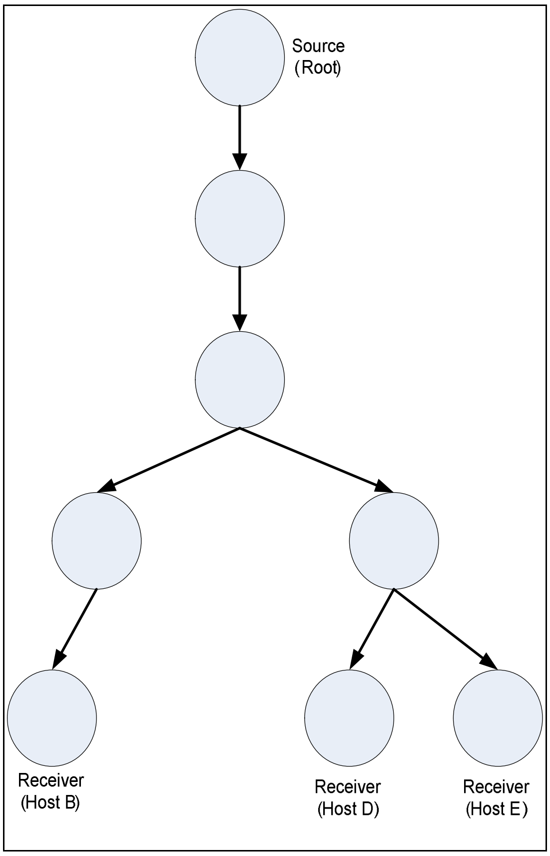

Figure 2-3 Multicast Tree

Bandwidth requirements for multicast delivery are widely dependent on applications such as news, which requires around 10 kbps as a lightweight data. However, it will be up to 384 kbps for delivery of video content and video streaming. Similarly, audio streaming will require a bandwidth of 48 to 64 kbps [9, 10], while low-quality video with audio will require a bandwidth up to 128 kbps [9].

Moreover, in 1994, a host in Japan was found to require a bandwidth over the entire MBONE (Multicast Backbone) of 650 kbps for video streaming. However, if compared with the bandwidth requirement for digital TV, there is a big difference. SDTV (Standard-definition TV), with MPEG-2 compression, will require 2-5 Mbps and 1.5-2 Mbps with MPEG-4 compression. On the other hand, the bandwidth requirement for HDTV (High-definition TV) is increased about five times, 15-20 Mbps with MPEG-2 compression and 5-10 Mbps with MPEG-4 compression [11].

In addition, another advantage of multicast is enhanced efficiency by reducing loads on central media servers in terms of CPU (Central Processor Units) power, memory usage and protocol management. Network routers also only need to manage and maintain one data stream per multicast service [12]. However, routers do have the added burden of needing to maintain knowledge of which devices belong to which multicast tree and to manage routing along these trees.

By using multicast delivery, video streaming application can offer a higher quality service because the load for single multicast delivery is less than multiple unicast delivery [13].

The network bandwidth saving is the main point expected of multicast's effect on the network. Hence, network providers normally must consider the cost of managing, maintaining and implementing multicast. However, the main factor is bandwidth. R. Chalmers and K. Almeroth [14] defined a metric to compare the performance between unicast and multicast delivery. The metric compares the total number of links traversed by unicast and multicast datagrams on a given network infrastructure. The metric refers to each links as a hop in the route path of a single multicast or unicast datagram.

$$\delta = 1 - \frac{multicast\ hops}{unicast\ hops} \tag{2.1}$$

Where

Multicast hops are the total number of multicast links in the distribution link in the network.

Unicast hops are the total number of unicast hops in the network.

δ is the multicast metric which is a fraction in the range $0 \leq \delta \leq 1$.

δ marks the percentage increase in the bandwidth utilization achieved by using multicast more than unicast delivery. For this metric if the value of δ is zero, this means the number of hops is the same. If the δ has a value of nearly one, it means the performance of using multicast delivery is higher compared to unicast.

Due to the number of multicast members being dynamic, new nodes and destinations can join and leave the multicast group at will. This means the multicast group size always changes over time, especially in a real time application. J. Chuang and M. Sirbu [15] presented a cost function of path, which is related to equation 2.1 and defines a direct relationship between the hop counts and the size of multicast group.

$$\frac{\boldsymbol{L_m}}{\boldsymbol{\bar{L}_u}} = \boldsymbol{N^k} \tag{2.2}$$

Where

$\boldsymbol{L_m}$ is the total multicast distribution tree length
$\boldsymbol{\bar{L}_u}$ is the average length of unicast routing path
N is the size of multicast group
k is the economies of scale (EoS) factor.

The range of value k is between 0 and 1, which is the value of the slope of the graph relationship between normalized tree cost and multicast group size. For most of the topologies investigated in their experiment, which is in real and generated networks, it is shown that $\boldsymbol{k \approx 0.8}$.

Now assuming $\boldsymbol{\bar{L}_u} = \frac{\boldsymbol{L_m}}{\boldsymbol{N}}$ will get:

$$\boldsymbol{\delta = 1 - N^{\varepsilon}} \tag{2.3}$$

Where

$$\boldsymbol{\varepsilon = k - 1 \approx 0.8 - 1 \approx -0.2} \tag{2.4}$$

So

$$\boldsymbol{\delta \approx 1 - N^{-0.2}} \tag{2.5}$$

Equation 2.5 gives us an estimate for the multicast performance, which is based on the number of destination nodes in the multicast group.

Another advantage of multicast is distributed application, as multicast makes multipoint applications possible. The example applications of multicast services are provided in the following.

- Multimedia Conferencing
- Online video/audio streaming
- Interactive distance learning
- Online TV
- Group online gaming
- Video on Demand (VoD)
- Commercial stock exchanges

However, a major problem with this kind of application is the lack of reliable delivery of data because multicast delivery is UDP protocol based, not TCP protocol. There are some multicast disadvantages such as [13]:

- *Best Effort Delivery*: dropping packets are to be expected. Therefore, multicast applications should be designed accordingly and should not expect high reliability of packet transmission. Packet losing should be accepted on those applications.

- *Duplicate Packets*: some multicast protocol techniques such as registers, asserts and STP transitions may result in the occasional production of duplicate datagrams.

- *Same Video Stream*: only one copy of multicast stream is sent to all users in the same multicast group, so every user receives the same data stream at the same time. Hence, individual users cannot choose the content they want, and also they cannot pause the video stream, rewind it or skip some parts.

- *Specific content*: in the multicast network, it is more complicated to control the users' access to specific streaming content because all users in the same group access the data at the same time.

- *Out of Order Delivery*: as multicast is based on UDP protocol, some routing protocol techniques may also result in packets being out of order. Hence, multicast applications should be designed to solve the issue of out of order packets arriving.

2.2.2 Functions of Multicast Delivery

This section provides a brief overview of the important functions of multicast delivery.

- ***Multicast Address Management:*** This function is about the assignment and the scope of multicast address.

- ***Multicast Service Announcement and Discovery:*** These services allow destination hosts to discover the availability of multicast source. In TCP/IP network, the SAP (Session Announcement Protocol) protocol handles this function [16].

- ***Multicast Group Management:*** Handles the collection and maintenance members of the multicast group. In IPv4 network, IGMPv2 (Internet Group Management Protocol version 2) protocol deals with this function. The counterpart in IPv6 network is MLD (Multicast Listener Discovery) protocol.

- ***Multicast Routing:*** Multicast Routing protocols are responsible for the multicast tree for example to build and maintenance the multicast trees. Also they are connect multicast group members and for the forwarding of data stream on the multicast tree. A number of multicast routing protocols have been standardized such as MOSPF (Multicast Open Shortest

Path First), PIM-DM (Protocol-Independent Multicast Dense Mode), PIM-SM (Protocol-Independent Multicast Sparse Mode), DVMRP (Distance-Vector Multicast Routing Protocol) and Core-Based Trees (CBT) [17].

- ***Reliable Multicast Transport:*** This function is to ensure the reliable delivery of multicast stream to a potentially large destination group.

- ***Multicast Mobility:*** Multicast delivery mechanism designs for static receiver. However, the mobile source and destination moves create issues for delivering the multicast data such as how to keep connection, managing IP address in a new location and so on. Solving these problems is a big challenge. This research has investigated this function in some depth and proposed a new framework to achieve the required functionality of this function.

According to multicast delivery, it is not only the way to send one message source to many destinations but also it can send from many sources to many destinations. Moreover, in a mobile environment this means the source could possibly move. In terms of a mobile source, these are divided into Any Source Multicast (ASM) and Source-Specific Multicast (SSM) [18].

ASM is where a mobile node submits data to any source in a multicast tree via an ASM group and either creates the root of a source-specific shortest path tree (SPT) forwarding datagram to a rendezvous point (RP) or destinations, or it distributes packets directly down a shared tree.

Source-Specific Multicast or SSM has been designed for multicast sources with static source addresses. Normally, the source addresses in a mobile node subscribed to an SSM group are directly used for route identification. However, the address of the SSM source possibly changes under mobility. So, mobile node implementations of SSM source filtering must be MIPv6 (Mobile Internet Protocol version 6) aware in the sense that a logical source identifier, which is HoA (Home Address), is correctly mapped to its current location represented by CoA (Care of Address) address.

2.3 Multicast Protocols in IP Networks

The aim of this section is to review the standard protocols, which are related to the multicast services within the network. In a TCP/IP network, the protocol responsible for multicast group management is MLD protocol for IPv6 network and IGMP protocol for IPv4 network. Hence, in the next section MLD protocol will be the focus.

Multicast routing protocols use algorithms to build multicast distribution trees. Each routing protocol is different in regards to how they share content, information and create paths [19]. Figure 2-4 is shown the classification of the intra-domain multicast protocols.

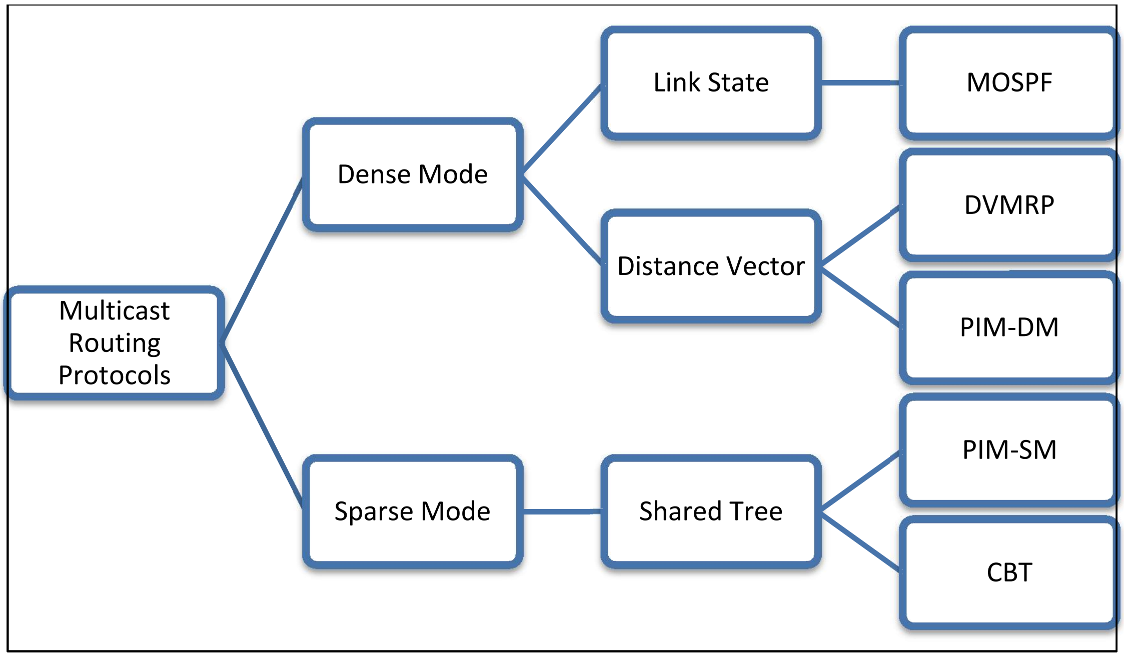

Figure 2-4 Intra-domain routing protocols [20]

A brief summary of these multicast routing protocols are as follows:

- ***MOSPF:*** This protocol is an extension of the OSPF routing protocol in unicast delivery. The MOSPF protocol uses the link-state algorithm the same as OSPF to create the paths with minimum protocol traffic overhead [20]. Moreover, if the number of multicast groups and group member size increase the computational complexity of the link-state algorithm will increase also [21].

- ***DVMRP:*** This protocol defined in 1988 by Waitzman, Partridge and Deering, was the first multicast routing protocol, which was deployed over the MBONE network [22]. This protocol is an extension of the RIP routing protocol in unicast delivery and extends to multicast network [21].

- ***PIM:*** It is a popular multicast routing protocol standard in TCP/IP network. This research has modified the protocol in the framework. In the following section PIM protocol will be discussed in detail.

- ***CBT:*** This protocol was defined in 1997 [23]; it is a shared tree protocol and cannot create a source-based tree.

2.3.1 MLD Protocol

Multicast data delivery comprises both local and global techniques in which local techniques are responsible for multicast group management. Global techniques are responsible for multicast routing. For LAN multicasting each node can choose whether it wants to receive multicast data or not. Multicast destination nodes will inform the network device that they want to receive data packets that are sent to the multicast group.

This process is called a “join” and is controlled by the IGMP protocol in an IPv4 network and controlled by the MLD protocol in an IPv6 network. In this research, global mechanisms are managed by the PIM protocol (Protocol Independent Multicast), which is multicast routing protocol in intra-domain network. For local mechanisms we chose to work with MLD protocol [24].

The MLD protocol was defined in 2004 by [25]. There are two versions of the MLD protocol, which are MLD version 1 (MLDv1) and version 2 (MLDv2). In this research we concentrate on the MLDv2 protocol, which is used by IPv6 routers to discover the receivers who wanted to join in a multicast tree [12].

Receiver nodes on a route keep an interface state for every IPv6 multicast address from which they want to receive data stream per interface [26]. The interface state on a receiver node contains a record including a filter mode and also source list for each multicast IP address [20].

2.3.2 PIM Protocol

To support multicast communication, a multicast routing protocol is required. For this research we consider the PIM protocol because PIM is one of the most popular shared tree multicast routing protocols. There are two types of PIM protocol, namely PIM-DM, which was defined in 2005 [27] and used in environments where multicast trees are populated densely within the network. Another type is PIM-SM defined in 1998, which is better suited to sparsely populated networks.

In this research we are focusing on the PIM-SM protocol because it is designed to support large region networks such as the internet. Sparse mode is activated when multicast groups are thinly populated across a large network region. This mode is designed for that situation. However, PIM-DM builds a separate source-based tree for every source, while a shared tree has been used for all sources within a multicast group.

All multicast sources in the tree transmit all multicast stream traffic to the root, and then the root forwards the multicast traffic to all destination nodes in the network. Normally, multicast sources encapsulate their multicast data in unicast data packets addressed to the RP router within the multicast tree. When the RP receives those packets, the RP will decapsulate these packets and then forward them over the multicast delivery tree to all members in the multicast group [20].

Normally, destination hosts join a multicast tree by sending a join message towards the RP. The routers on the route towards the RP will store status information for the multicast group while passing the join request to the RP router, thereby building the multicast delivery tree in the direction of the new destination.

There are many control messages, which are used within the PIM-SM protocol. The PIM-SM message can show the following parts:

- Bootstrap message
- PIM-SM control-message encapsulation
- Hello message
- PIM-SM packet header
- Encoded Unicast Address field
- Join/Prune message
- Encoded Group Address field
- Candidate RP advertisement
- Encoded Source Address field
- Register message
- Assert message
- Register-Stop message

2.4 Multicast Mobility in WiFi Network

In this section, a review of mobility within the wireless network will be presented, including multicast mobile and related protocols.

2.4.1 Overview of WiFi Network

Wireless networks support different data rates and coverage area sizes. For example, IEEE802.11g supports a data rate of 54 Mbps but GPRS on 3G in practice only manages a data rate of around 9.6-384 kbps, such as specifying a minimum data rate of 144 kbps high mobility application like a car, while slow mobility like a pedestrian application requires 384 kbps and up to 2 Mbps stationary applications. Typical network data rates are summarised in Table 2-1 [28].

Table 2-1 A data rate and coverage area of wireless technologies [28]

Network type	Frequency	Data Rate	Coverage
Bluetooth	2.4 GHz ISM band	Max 721 kbps	0.1-10m
IEEE 802.11a	5 GHz	20 Mbps	50-300 m
IEEE 802.11b	2.4 GHz	11 Mbps	Up to 100 m
IEEE 802.11g	2.4 GHz	54 Mbps	30-150 m
IEEE 802.16 (WiMAX)	10-66 GHz	Max 70 Mbps	Over 50 km
IMT2000, UMTS	2 GHz	Max 2 Mbps	30m-20km
GPRS (GSM), EDGE (HSCSD)	900, 1800, 1900 MHz	9.6-384 kbps	Up to 35 km

2.4.2 Mobility in WiFi Networks

In IPv6 wireless network, the protocol that provides mobility support is Mobile IPv6 protocol, and this protocol will maintain the connection when the mobile node moves. Also, it includes the responsibility for the reachability of the mobile node and network, keeping records and IP address. So, in the next section the protocols that are related with mobility will be presented.

2.4.2.1 Mobile IPv6 Protocol

The mobile IPv6 protocol has been proposed by the IETF (Internet Engineering Task Force) to provide mobile support for hosts within IPv6 networks. In Mobile IP, a mobile node uses two IP addresses that are its home address and a care-of-address. The home address is a stable IP address assigned to a device and based on their point of network connection [29].

In contrast, the care-of-address is a temporary address provided for a 'foreign' network and this address will change as the device moves between different IP subnetworks. The care-of-address in IPv6 network can be formed based on stateless or stateful mechanisms [30]. When the mobile node moves, it first forms a care-of-address based on the prefix of the foreign link. Then, the mobile node sends a Binding Update (BU) message to the home agent, which is its temporary care-of-address (CoA).

After that, although the home agent wants to grant or deny the request from the mobile node, the home agent will send a registration message reply [31]. After the registration process is successful, any messages destined for the mobile node are intercepted by the home agent, which encapsulates the packets and tunnels them to the foreign agent. Then, the data streams are forwarded to the mobile node [32].

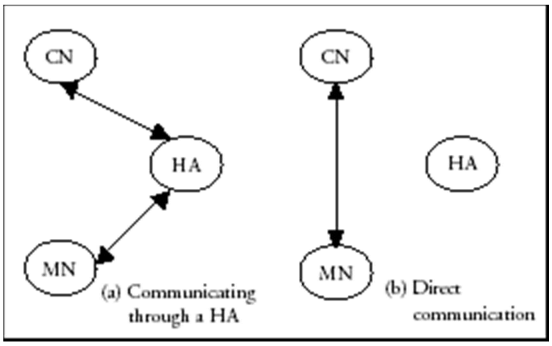

Figure 2-5 Mobile IPv6 Protocol

Mobile IPv6 offers improvements in this process compared to Mobile IPv4. For example, Mobile IPv6 can eliminate the triangular routing issue and produces route optimization. Route optimization is the process that enables the correspondent node to reroute data stream on a direct path to mobile destination [33].

Moreover, Mobile IPv6 includes embedded binding updates for the home agent and care-of-address configuration for location updates. Additional security has also been incorporated such as authentication header processing to provide validation for mobile nodes [19].

2.4.2.2 ICMPv6 Protocol

The ICMPv6 (Internet Control Message Protocol version 6) protocol [34] is used to carry IP control messages for various purposes such as destination unreachable, time exceeded and parameter problem. In addition, ICMPv6 is defined to carry information between hosts, between routers, or between hosts and routers.

There are two types of ICMPv6 message [35]:

- **ICMP error message**

  The functions of ICMPv6 error messages are to report forwarding or delivery errors by either a router or the destination node. The ICMPv6 error messages consist of destination unreachable, packet too big, and parameter problem and so on.

- **ICMP informational message**

  Informational messages provide simple diagnostic functions such as echo request, echo reply, and additional host functionality for example MLD and ND (Neighbour Discovery), which is a set of processes and messages that determine relationships between neighbouring devices.

2.4.3 Multicast Mobility in IPv6 WiFi Network

When a mobile node moves from one network to another network, it is a challenging problem to maintain reachability and transparency of a mobile node. In case of multicast wireless network, the scenario of handover is particularly challenging and serval issues emerge with most solutions due to the handover impacts.

The main problem when the receivers move is multicast latency problem. ***Multicast latency problem****,* whenever an MN moves to a foreign network, the delay experienced by executing handover process.

Moreover, if a Home Agent (HA) does not support multicast router functionalities, an MN have to discover other a multicast router (MR) and send notification by using MLD

(Multicast listener discovery) query/report messages including a request to join a multicast tree. In this case, the maximum query period of MLD is up to 125 seconds [36]. For some applications, this increased latency time is undesirable such as video conferencing.

Run-liu, W and Yun-hui [37] proposed a multicast routing algorithm trying to reduce the cost of created multicast tree combines with Mobile IP. Also this algorithm tries to reduce the bandwidth of multicast data stream. For applying to large scale of wireless network and reducing join delay time in handover process.

Holbrook, Cain and Haberman [38] proposed a new approach called the Mobile Multicast Protocol (MOM) in 2003. This approach introduces a new entity called the designated multicast service provider (DMSP) and uses a foreign agent entity. The main issue of this research is to reduce the duplicated multicast packets on home agent. However, it created the problem between the HA and FA networks.

Figueiredo, Jeon and Aguiar [39] proposed the solution to reduce vertical handover by adapting a cross layer approach and IEEE 802.21 Media-Independent Handover. There is the process of selected FA links in the network. Also the process uses the single tunnel for completing the delivery process.

2.4.4 Multicast Mobility in UMTS Network

For 3G (UMTS) mobile networks, multicast can be delivered through IP multicast, MBMS (Multimedia Broadcast/Multicast Service) and CBS (Cell Broadcast Service). CBS is a standard that allows the delivery of messages to multiple users in both GSM and UMTS networks. Similarly, MBMS is a standard that is designed to support efficient multimedia broadcast and multicast delivery in GPRS and UMTS networks.

With MBMS, there is provision for streaming services for the delivery of continuous multimedia data traffic. For CDMA2000 networks, there is the BCMCS standard for managing the capability to deliver broadcast and multicast services [20].

2.5 Multicast Mobility Problems in WiFi Networks

2.5.1 Multicast Mobility Problems

Although the research area of multicast mobility has been a concern for about 10 years, there are numerous proposals but not yet a generally accepted standard solution. One reason for this is that the standard multicast protocol was designed for stationary nodes and not for mobility. The problem and challenge of multicast mobility can be divided into 3 categories [40]:

➢ ***Multicast routing problems***: due to the movement of mobile nodes and the source there are routing problems, such as:

- *Network inactivity*: means when the foreign network the mobile node visited does not support multicast delivery. Hence, the mobile node has to stop receiving the multicast message.

- *Core placement*: when the mobile node moves to a foreign network, the new route will be established. If the mobile node changes zone more often, the frequent handovers can lead to a situation that those multicast routers are off centre. That results in a possible non-optimality configured path.

- *Multicast Encapsulation/Decapsulation:* a variety of methods are used for tunnels to keep connection between the mobile node and home network when the mobile node moves to foreign network.

➢ ***Mobile receiver problems***: In multicast routing when the MNs are moving, they need both efficient IP mobility management and multicast mechanisms to provide a seamless service. Hence, multicast mobility has many constraints that should be considered. It can be classified into:

- *Packet loss:* a mobile node may miss some multicast data because when it moves normal multicast packets will continue to be delivered to the home network for a short period of time.

- *Packet duplication*: happens when the mobile node receives the packet from a different multicast router but from the same multicast tree.

- *Packet out of order:* when handover occurs.

- *Tunnel convergence problem:* the tunnel convergence problem is concerned with the delivery of multicast packets to mobile member nodes that are located in several foreign networks over bi-directional tunnels using Mobile IP. An MN receives multicast packets from a home agent, duplicated multicast packets are transmitted to over several tunnels and these become a problem [36].

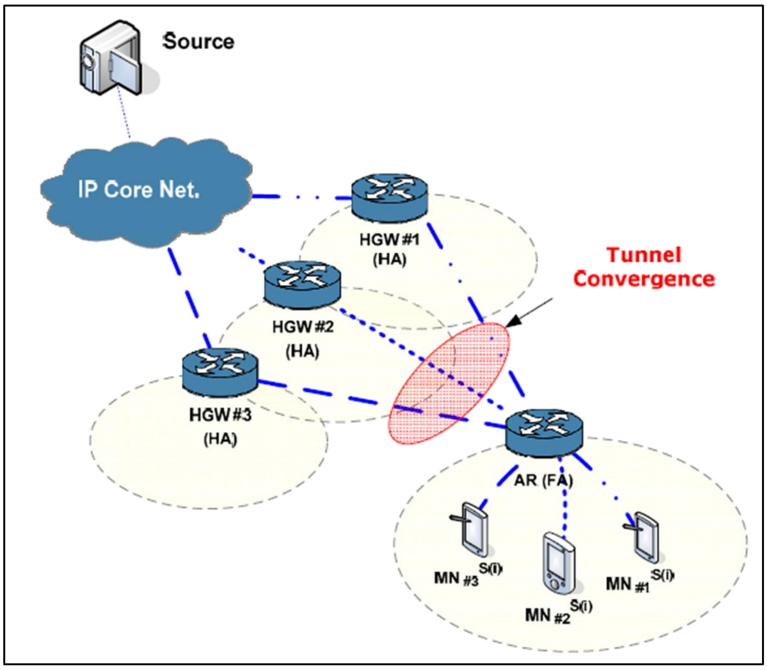

Figure 2-6 Tunnel convergence problem [36]

➢ ***Mobile source problems***: In multicast mobility, it is not only the mobile node that can move. Multicast sources can also move. Consequently, the following aspects must be considered:

- *Transparency:* is a major problem when a multicast source moves. There is a problem with the CoA of the multicast source, because when a mobile source uses a new CoA as a multicast source address, it cannot send multicast messages immediately. It has to wait until the mobile node explicitly notifies that CoA[41].

- *Reverse path forwarding (RPF):* because it is specific to source location.

- *Packet loss*.

- *Source active problem*: because the multicast tree has to be reconstructed when the mobile source moves to the new network.

2.5.2 Overview of Handover Problem

Currently, there are several wireless technologies to provide ubiquitous information access to users when they are moving. A mobile device such as a smart phone offers multiple wireless network interfaces and can access these as it moves between different network environments.

Typically, when a mobile device is within the building they receive signals from WiFi and 3G at the same time but choose WiFi for data services. However, when they move out of the building, they will be receiving only the 3G signal. The smart phone will, of course, automatically switch between these networks when WiFi connectivity is lost.

For wireless networks, there are two types of handover that can occur in the network.

Horizontal handover

Horizontal handover is the process by which mobile devices switch from one cell to another cell within the same network technology; this will be called "intra-system handover".

Vertical handover

For vertical handover or "inter-system handover" the mobile device is switching between different network technologies; for example between 3G and WiFi. That means that vertical handover is different in several aspects such as data rate, bandwidth and frequency of operation.

Moreover, vertical handover can be divided into two sub types, which are upward-vertical handover and downward vertical handover [20]. For example, the device moves from WiFi to 3G. Downward vertical handover is a handover that disconnects from a cell providing broader coverage to a wireless overlay with a smaller cell size, and generally higher bandwidth per unit area such as from 3G to WiFi.

2.5.2.1 The Handover Process

One of the major problems of multicast mobility is the handover process which occurs between cells or between different technologies. The handover process can be divided into three phases [8],[28].

Network Discovery or System Discovery: This phase is where the mobile node (MN) searches for reachable wireless networks, usually based upon values of received signal strength.

Handover decision: These are the rules that determine when a mobile node should perform handover. A decision for vertical handover may depend on various parameters such as bandwidth, delay and transmitted power.

Handover Implementation or Handover Execution: This is the process by which a mobile node's connection is rerouted from their existing network to a new network in a seamless manner. Mostly, it requires the network to transfer routing information about the mobile terminal.

Recently, research has developed various techniques and approaches to deal with the multicast mobility process and handover at different layers of the network. Most of these strategies are based on a modification and implementation at the network layer for example modifying the Mobile IP protocol [28,19,42]. Other approaches operate at the transport layer and at the application layer such as a modification to SIP (Session Initial Protocol) [43].

For the Network Discovery Phase, the MN must search for available reachable wireless networks. In this state most MNs should always keep all network interfaces on. However, keeping network interfaces on all of the time becomes a weakness because it consumes battery power without any benefit of delivering real data.

In [8], the context information about a network is stored in the context awareness database. Many parameters are collected by system discovery and are dependent on the adopted network interface card within the MN. For instance, these could include signal RSSI (Received Signal Strength Indicator), bit rates, MAC (Media Access Control), etc.

In addition, network monitoring systems adopt SNMP (Simple Network Management Protocol) to extract more relevant network information from all APs (Access Points) and ARs (Access Routers) in each subnet such as data transmission/receiving rate, network loading and multicast connections.

2.5.2.2 Handover Decision Phase

There are many research papers proposing strategies for making decisions about handover. We categorize these into three types: network-controlled handover, mobile-controlled handover and mobile-assisted handover.

Network-controlled handover is when the network makes the handover decision for a mobile node. Shantidev Mohanty [45] proposed a novel architecture using the Network Inter-operating Agent called NIA and Integration Gateway (IG) to integrate the 3G systems and WiFi networks of various providers. The IG functions as a traffic monitoring unit and seamless roaming module.

In [42], mechanisms are presented for PMIPv6 (Proxy Mobile IPv6) by two multicast mobility listeners called LMA-MLM and MAG-MLM. Both listeners will be responsible for subscribing to the multicast group and receiving multicast packets on behalf of MNs in its

domain. The MAG-MLM uses a MAG (Mobile Access Gateway) to detect the detachment of the MN.
The LMA-MLM uses a LMA (Local Mobility Anchor), which is responsible for maintaining the reachability of MNs by updating the binding cache and maintaining the tunnel to the MAG for packet delivery. Their process can achieve the whole multicast handover process without the involvement of the MN. However, the LMA-MLM still has to deal with the problem of encapsulation.

Kim and Han [46] have proposed PMIP protocol with IP multimedia system. The protocol that they proposed naming PMIP-M protocol. The main idea of this protocol is the user can continue received multicast data stream even when they migrates in the new network which without IP multicast capability.

Mobile-controlled handover is where the mobile node must take its own signal strength measurements and make the handover decision on its own. In [47], an algorithm is proposed based on the Markov decision process (MDP) formulation, which tries to maximise the expected total reward of a connection.

Mobile-assisted handover is where the decision to handover is made by the mobile node and network in cooperation. In [43] a mobile QoS (Quality of Service) framework is proposed for heterogeneous IMS (Internetworking Management System) interworking by modifying SIP multicast. However, this method consumes network bandwidth and MNs need to reserve bandwidth. Park and Won [48] analyse about mobility management architecture such as MIP and PMIP protocol for avoiding any tunnels for multicast delivery in heterogeneous network.

C. Wen et al. [8] proposed an integrated framework for MNs and core wireless networks by using a context-aware handover scheme. They periodically collect the parameters of various available network status reports and information about host application services. For example, the best one from a list of available access points is selected based on network conditions and user defined policies. The multicast connection management is then performed efficiently by a multicast agent in WiFi and 3G networks. Hence, the MNs can make handover decisions to activate the proper network interface switching to avoid discontinuities in the delivery of multimedia application services.

One thing that is important when the handover occurs is considering the handover metrics used to make the decision. There are many handover metrics that are used to indicate when handover should be performed:

Connection cost

For users, connection cost is a key consideration, especially when different network operators may create different billing schemes. Hence, it might affect the user's choice of handover.

Network-related parameters

There are many network parameters used for making the decision such as bandwidth, load, network latency, traffic congestion, location information and so on. Moreover, that information is useful for load balancing across different networks and QoS [20].

Application types

For example, some multimedia applications require reliability in networks. Hence, different types of applications may require different levels of QoS determined by the percentage of lost packets or the delivered data rate.

Battery power

Battery power may be a significant factor for handover in some cases. Moreover, an MN with multiple interfaces must keep an interface active all the time but this consumes battery power even without receiving any data.

2.5.2.3 Handover Implementation

Several multicasting schemes have been proposed for mobile networks. In [44] they used the ISHO (inter-system handover) protocol and include the concept of a dynamic boundary area to support seamless roaming between different networks. However, this scheme requires the NIA and IG to be added into the network architecture.

In addition, when an MN moves across to the new network, it is important to consider the associated security mechanisms. The security mechanisms of 3G-WiFi do not address the security of multicasting. However, for supporting a secure link for multicasting a framework of multicast key agreement by a modified EAP-AKA protocol has been presented [49]. Here

EAP-AKA is the Extensible Authentication Protocol method for UMTS Authentication and Key Agreement used for authentication in wireless networks. They are divided into two phases: Initial phase and Key refresh phase. The benefit of this technique is that it saves communication overhead, computation overhead and does not need a huge change for existing protocols [50].

2.5.3 Handover within WiFi Networks

For the WiFi-3G handoff process, [51] a method is proposed to reduce latency by using the ISHO (inter-system handover) protocol which includes the concept of a dynamic boundary area to support seamless roaming between different networks. However, most of this research tries to solve the problem in the network layer with others seeking to solve it by modifying the transport layer. For instance, in [52] a mobile QoS framework is proposed for heterogeneous IMS interworking by modifying SIP multicast. However, this method consumes network bandwidth and MNs need to reserve bandwidth.

The handover procedure of Mobile IPv6 protocol can be expressed as 2 parts: L2 (link layer) handover latency and L3 (network layer) handover latency. The L2 handover consists of channel scanning process, authentication and association process. Generally, L2 handover latency is about 100 – 300 ms however it depends on the structure of network topology. For L3 handover latency in MIPv6 consists of two main parts: CoA configuration and Binding update. The process of CoA configuration is starting from Router discovery process until the MN obtained a new CoA. The Binding update procedure is about the MN inform HA and CA nodes about their new location which is new CoA address. Normally, handover latency of Mobile IPv6 is about 2000-3000 ms [53] this is why it is possible that MN can lose connection completely during handover process.

Tien-Thinh Nguyen [54] had applied DMM (Distributed Mobility Management) concept with multicast mobility in IPv6 network by enable IP Multicast with MLD proxy function. However, the result show that when the mobile receiver moves, the network have to build the tunnel between source and destination. This is a case of tunnel convergence problem. Also it has a problem about service disruption and delay which cannot acceptable in some delay-sensitive service. Moreover Nguyen and Bonnet [55] had been studying about load balancing mechanism among LMAs (Local Mobility Anchor) to solve a bottleneck and single point of failure issues.

2.5.4 Multicast Handover in Wireless Networks

For streaming multimedia content in 3G network has been standardized under the 3GPP-PSS (3^{rd} Generation Partnership Project – Packet Switched Streaming Standard) which is released in April 2001. The 3GPP-PSS are described presentation of information, the audio and video formats of that stream within complete protocol stack in IP layer [56]. In UMTS, the IMS was extended to include MBMS. The 3GPP MBMS has the following characteristics:

- There is no immediate Layer 2 source-to-destination transition, resulting in transit of all multicast traffic at the GGSN.

- As GGSNs commonly are regional, triangular routing, distant entities and encapsulation this may cause a significant degradation of efficiency.

In 3GPP2 (3^{rd} Generation Partnership Project 2) [57], the MBMS has been extended to the Broadcast and Multicast Service (BCMCS) [58], which on the routing layer operates very similar to MBMS. In both 3GPP (3^{rd} Generation Partnership Project) and 3GPP2, multicast can be sent using either point-to-point (PTP) or point-to-multipoint (PTM) tunnels, and there is support for switching between PTP and PTM.

A mobile multicast node may change its point of Layer 2 attachment within homogeneous access technologies (horizontal handover) or between heterogeneous links (vertical handover) [59]. In [60] has modified PIM-SM to support handover latency and keeping connection. By proposed multicast routing protocol named MC-PIM-SM by extended from PIM-SM protocol [61]. Mobility applications transport for MIH are required as an abstraction for Layer 2 multicast service transfer in an Internet context [45] and are specified in [62].

Functions required for MIH include:

- Service context transfer.

- Service discovery.
- Service invocation.
- Service context transformation.

In [63] is shown the amount of multicast packet loss, when handover occur at the mobile node in equation. Suppose $\varphi_P^{(BASE)}$ is the amount of multicast packet loss for the base multicast handover procedure. Let δ_s denote the average multicast session arrival rate per second at the mobile node. $\varphi_P^{(BASE)}$ is obtained as

$$\varphi_P^{(BASE)} = \delta_s E(S) L_{HO}^{(BASE)} \tag{2.6}$$

Where *E(S)* is the average session length in packets

This research will focus on mobile receiver problems and so methods to solve these problems have been proposed. The problem of achieving seamless mobile receiver multicast handover can be addressed by one of the following:

- ***Home subscription-based solution:***

  - *Mobile IP Home Subscription* or *bi-directional tunnelling:* this approach relies on the Mobile IP protocol and uses a local router in the home network as the multicast router for responses such as forwarding multicast group membership control messages to the mobile node even when it moves to a foreign network. However, tunnelling will create the process of encapsulation/ decapsulation and fragmentation problems.

  - *Multicast encapsulation*: that is encapsulation of multicast data packets to shield mobility and to enable access to remotely located data services such as from the home agent.

- ***Remote subscription-based solution:*** by forcing the mobile node to re-initiate multicast distribution following handover. However, this technique cannot support session persistence under multicast source mobility.

- *Agent assistance*: there are many protocols that are proposed for agent-assisted handover for host-based mobility such as Fast MIPv6 (FMIP6) and Hierarchical MIPv6 (HMIPv6).

- *Network-based mobility management:* Proxy MIPv6 (PMIPv6) [19] is multicast transparent in the sense that the MN experiences a point-to-point home link fixed at its LMA (Local Mobility Anchor). In [63] network based mobility management is deploying for the mobile nodes, also the tunnel between the LMA and itself for the MN. However, PMIPv6 still has a problem about MTU size from spanning tunnels at the receiver site.

➢ ***Hybrid architectures:*** that tries to find the methods, which avoid the complexity at the internet core network.

- *Hybrid shared tree:* [64] proposes the hybrid shared tree approach by introducing a mobility-agnostic multicast backbone on overlay.

- *Hierarchical local registration:* the network model has proposed hierarchical and local registration. The registration consists of having a root FA (Foreign Agent) and lower FAs. The MN registers its CoA with the root FA. All the FAs exchange summary reports that consist of the common multicast group of interest on the lower levels. However, this approach required an extra cost to select multicast service provider (MSP) [65].

➢ ***MLD Extensions:*** there are many methods by extended MLD message. Some of them modify an MN operating predictive handover such as FMIPv6.

2.6 Summary

The approach which is presented in this research aims to offer a smooth handover between the home network and the foreign network. The advantages of both home subscription-based and remote subscription-based solutions have been combined. A modification has been offered which responds to problems posed by mobility. However, this framework does not make use of bi-directional tunnelling, which means that the framework solves three problems: tunnel convergence, encapsulation / decapsulation overhead delay, and fragmentation problems.

The concept of a remote subscription-based solution has been applied to this framework by creating a reserve route to neighbouring networks but kept in standby. The FMIP6 protocol also uses the remote subscription but it still suffers a short lost-connection time of MN. Moreover, this approach does not require multicast tree reconstruction as many previous methods do. The design detail of this framework will be described in the next section.

Chapter 3 A New Framework for Multicast Mobility in WiFi Networks

3.1 Introduction

This chapter presents the designed framework that was produced during this research. The chapter describes about network architecture, protocol overview and the details of protocol process step by step.

3.2 Network Architecture

In this research, there is a requirement for a system that provides support for research scenarios in WiFi networks that are connected through the internet. According to the real world, the WiFi network will combine with an IP wired infrastructure network with a gateway router to route data through to the internet. There are a variety of clients in the network such as PCs, laptops and mobile devices [66].

Before starting to explain the detail in each process of the framework, it is first necessary to show the scope of the network architecture. Hence, in this chapter the network architecture being used in this research is described and shown in Figure 3-1.

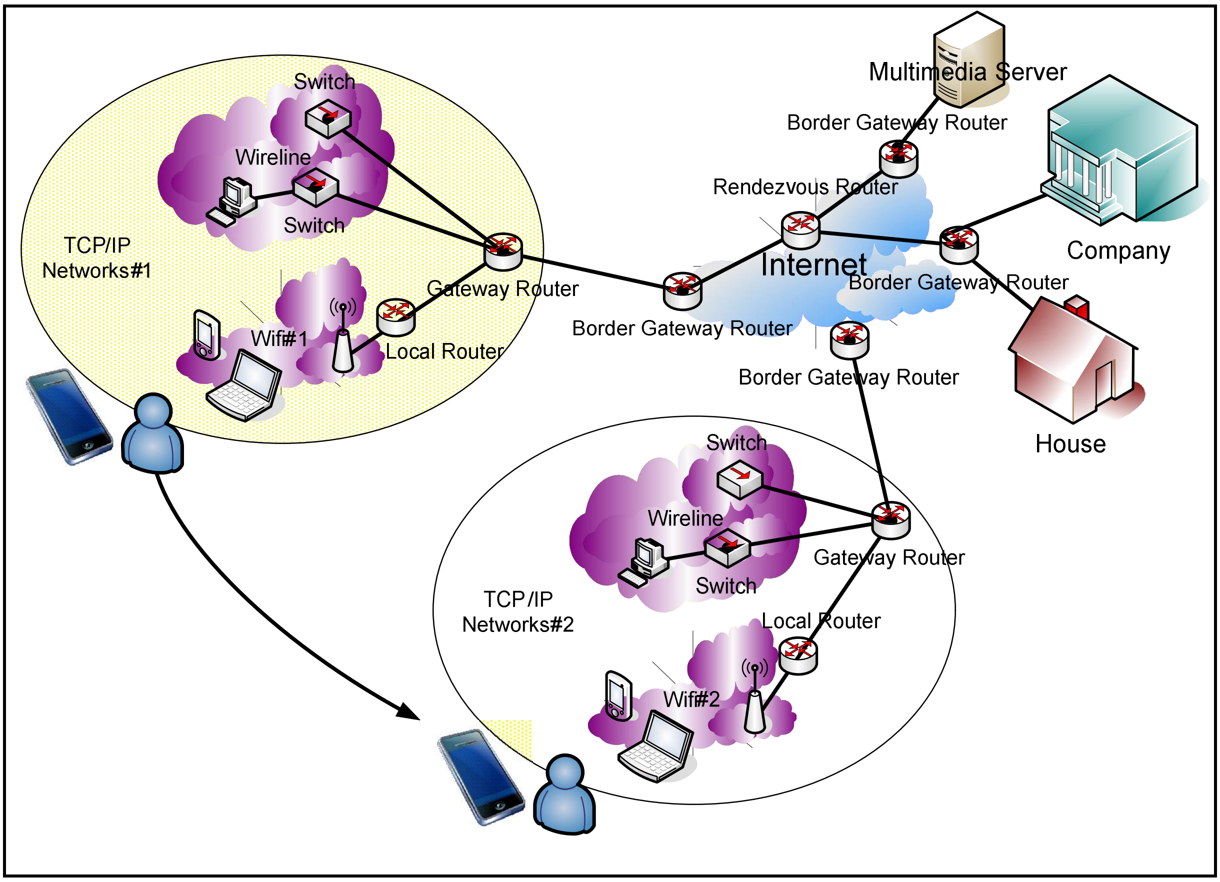

Figure 3-1 Network Architecture

Figure 3-1 is shown the stage which we consider WiFi to WiFi handover.

- Here, the architecture comprises only WiFi networks that comprise: a media server which represents the source server, which is sending the multicast packets to mobile devices in the network. The rendezvous router is a central router in a multicast tree.
- Border gateway router is a core router which is enabled for multicast delivery services and supports multicast routing protocols such as PIM (Protocol-Independent Multicast), MOSPF (Multicast Extensions to Open Shortest Path First), DVMRP (Distance Vector Multicast Routing Protocol) and so on.
- Gateway router is a core router within a company network which is connected to both WiFi and wireline networks, providing Internet access and supporting multicast routing protocols.
- Local router is a router that connects between a WiFi access point and other network devices in the local network.
- WiFi access point provides localized wireless coverage.

- Mobile nodes with embedded WiFi interface roam within this network.

3.3 Protocol Overview

In this section we will describe the process by which the new proposed framework is designed to improve handover performance for multicast services. Procedures and protocols already exist for handling handover from one WiFi network to another. However, such protocols, of which mobile IP is a key example, achieve handover by firstly making contact with a new WiFi base station, obtaining a new IP address and then re-routing traffic to that new address through a modified multicast tree. Unfortunately this leads to a loss of connectivity and hence, service whilst this process is taking place. Similarly, IGMP manages the distribution of multicast services through the establishment of a multicast tree which is maintained by the routers. When a mobile node, moves, this tree needs to be modified to accommodate the mobile node's new location. This therefore leads to further delay which handover takes place. Our approach is to complete as much of the existing handover process as possible before the physical handover actually takes place. This therefore will minimise the actual handover delay at the expense of having to establish several connections to neighbouring networks, most of which may never be activated.

The overall concept of our new framework is that a mobile node will use mobile IP but modified in such a way as to allow connectivity to multiple foreign networks. In so doing a mobile node will receive a Care of Address from each foreign network that is within range. These addresses are then used to compute multiple extensions to the existing multicast tree, which are then held in a standby mode until required. The standby route represented as a dash line in Figure 3-2 below. Once a mobile node actually completes handover by moving to a new WiFi base station, the Care of Address that has already been obtained and the associated change to the multicast tree are then activated with traffic being routed to that mobile node via this new route.

Hence, establishing the Care of Address and determining the required modifications to the multicast tree in advance of needing them reduces the handover delay to one of switching between the route currently being used within the multicast tree and the new one. The goal is to achieve this through the minimal modification to existing protocols and procedures. In this chapter it is shown how the new framework can be applied to a network that employs mobile IP and IGMP.

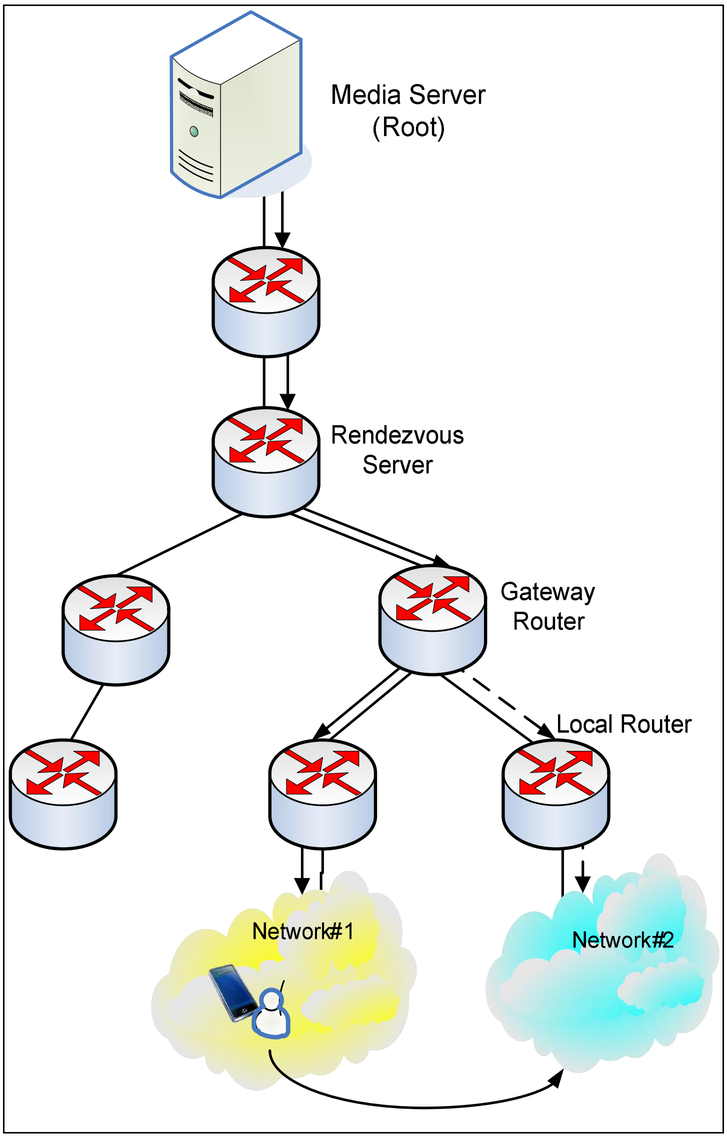

Figure 3-2 Multicast Route

3.4 Process Diagram

In this section we will describe a process diagram of the proposed framework in details. From the Figure 3-3 until 3-6, the modified messages or processes in the proposed framework will be represented by underlining the text, while standard protocol process will be represented by normal text.

The outline of our process is as follows [67]:

- The mobile device connects to the WiFi#1 as home network and receives a multicast data stream as usual.
- The mobile device sends the message to join WiFi#2 as a foreign network in advance by modified Mobile IP protocol.
- The mobile node uses its CoA address from WiFi#2 to establish a new multicast route with the same media server.
- After it receives the multicast message from the second route, it disconnects the second route at a point along the path between its local router for WiFi#1 and the rendezvous router.
- Local router of WiFi#2 network sends a modified PIM protocol message to keep its connection as a standby route.
- If handover occurs then the modified ICMP protocol message will be sent to reconnect the second route. This will minimize handover delay because the second route has already been configured and just needs to switch from standby to active.

Figure 3-3 is shown the initial steps of how a mobile node is able to register for receipt of a multicast stream being delivered by the media server.

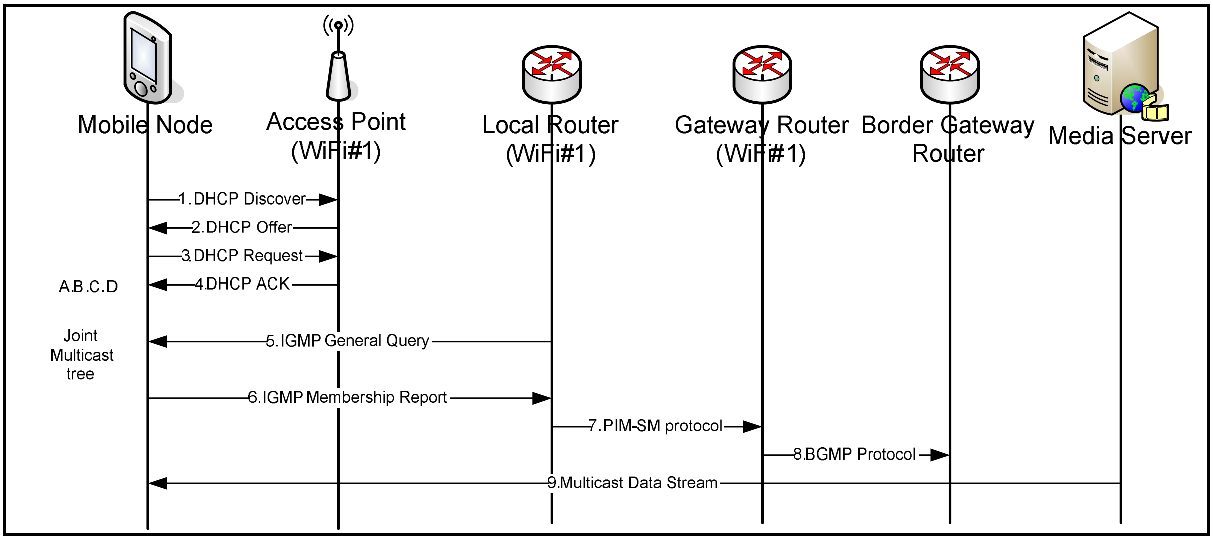

Figure 3-3 Starting connection process

Details of this process are as follows:

1. The mobile node sends a DHCP Discover to its local access point by broadcasting in order to discover the DHCP server that can supply it with an IP address.
2. In this case WiFi#1 will respond with a DHCP Offer including an IP address.
3. If the mobile node accepts that IP address, the mobile node will send a DHCP Request message including that IP address back to the access point for confirmation.
4. Access point WiFi#1 will send a DHCP ACK message back to the mobile and allow the mobile node access to the network. At this stage, the mobile node will has IP address for establish a connection to media server.
5. Usually, if a local router is enabled for multicast delivery it will send an IGMP General Query every 60 seconds within the network. Hence, at this stage WiFi#1 will send an IGMP General Query message out to the mobile node [68].
6. In this scenario, the mobile node wants to connect to the multicast tree and so it will send an IGMP Membership Report, including the IP multicast address to which it wants to connect.
7. The local router will then use its multicast routing protocol to connect to the appropriate multicast tree. In this case it will create a PIM-SM protocol message and send it to the gateway router within the WiFi#1 network.
8. Since in this case the media server is outside the local network and on the internet, BGMP will be used by the gateway router to connect to the multicast tree.
9. Once connected to the multicast tree, the mobile node will start to receive the multicast data stream.

When a mobile node moves then in effect its position on the multicast tree moves or, in some cases, the multicast tree will need to be extended to accommodate the mobile nodes' new location. Moving within a network also requires the mobile node to be issued with a care of address. Therefore, in order to improve handover efficiency, in our scheme we seek to obtain the care of address and modify the multicast tree ahead of the time when it is actually needed.

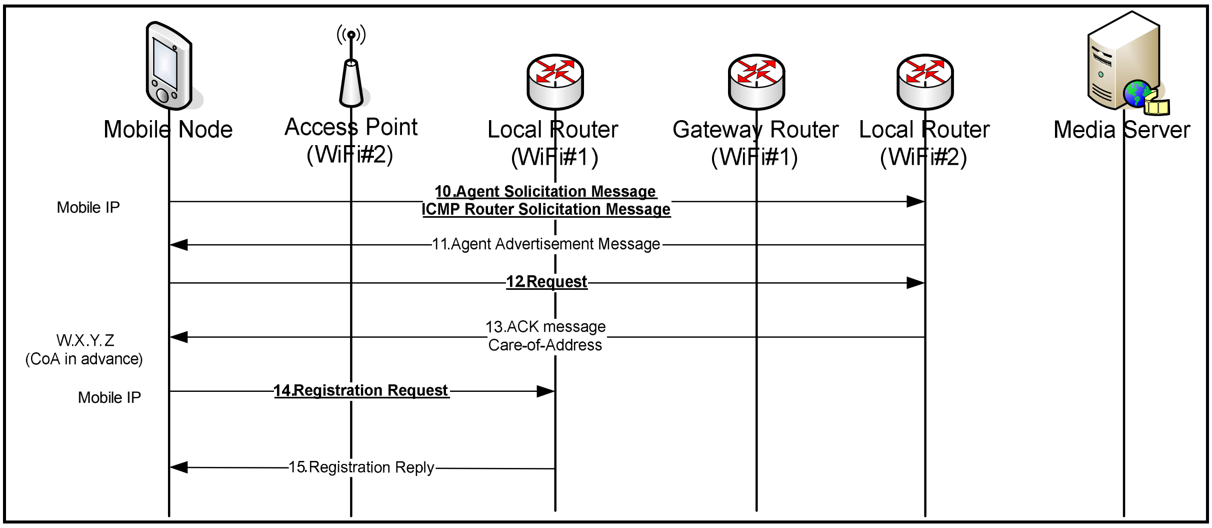

Figure 3-4 Request CoA address in advance process

Figure 3-4 is shown the process of obtaining a care of address in advance from access point WiFi#2 as the mobile node is preparing to move away from access point WiFi#1.

The procedure is described as follows:

1. When the mobile node received a signal from foreign agent which is Local Router on WiFi#2 network in Figure 3-4, that implies the mobile node is in range of foreign agent. In the new framework, we designed that the mobile node will broadcast a Mobile IP Agent Solicitation Message every 30 seconds for registering to the new foreign agent. This process Mobile IP does not use a new packet type for agent solicitation, it uses the router solicitation packet of ICMP.
2. After receiving Mobile IP Agent Solicitation Message, the local router (foreign agent) in the Wifi#2 network will send back an Agent Advertisement Message which includes the IP care of address (CoA) to the mobile node.
3. If the mobile node accepts this address, the mobile node will send a Request message to confirm to the local router in WiFi#2 network that it wants to use this IP address.
4. The local router will subsequently acknowledge this with an ACK message. At this stage, that means the mobile node had CoA address in advance before moving to WiFi#2 network. Also it means the mobile node can use this IP address if it handover into WiFi#2 network.
5. Normally, when a mobile node receives a CoA address it needs to register this with

its Home agent which, in this case, is the local router in WiFi#1 network.

6. After the home agent stores the information in their database, it will send a Registration Reply to confirm to the mobile node. At this stage, the mobile node has two addresses; one is an IP address from its home agent and the other is the CoA address from the foreign agent.

The next strategy is to establish a new multicast route from the foreign network to the multicast tree, the details of which are shown in Figure 3-5.

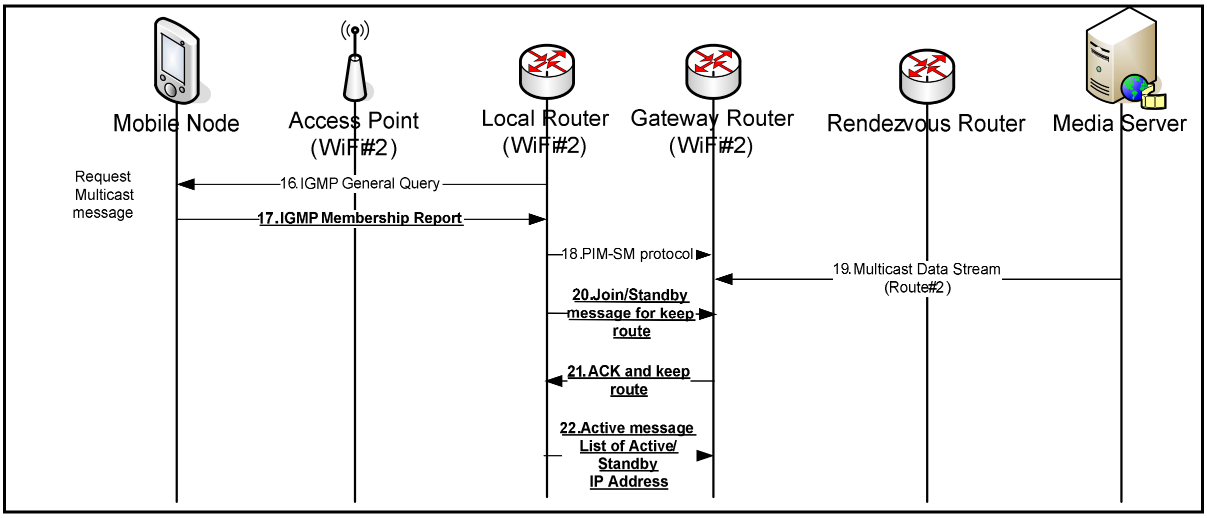

Figure 3-5 Request Multicast packets and keep route

The procedure is described as follows:

1. Normally, the local router will broadcast an IGMP General Query to the client in their network which in this case is the local router in the WiFi#2 network. For querying that there is any client would like to join any multicast tree in network.

2. If the mobile node wants to join the multicast tree it will send an IGMP Membership Report including the IP multicast address to the local router. In this framework the mobile node will use the CoA address to communicate.

3. The local router will use the PIM-SM protocol to communicate with the gateway router in the WiFi#2 network for creating the route to multicast tree by connecting to rendezvous router.

4. After joining the multicast tree, there is a multicast data stream from media server to

local router. However, at this stage, there are two routes connected to the same multicast tree but only one should be live. Therefore, we are proposing to modify the PIM-SM message to keep the route between the local router in the WiFi#2 network and the gateway router in a standby mode.

Figure 3-6 is shown details of the process when the handoff process occurs. The idea is to change the route at the foreign agent from standby to active. Here the secondary route which has been established using the CoA needs to be switched from standby to active and the currently active route which uses the home address needs to be switched from active to standby.

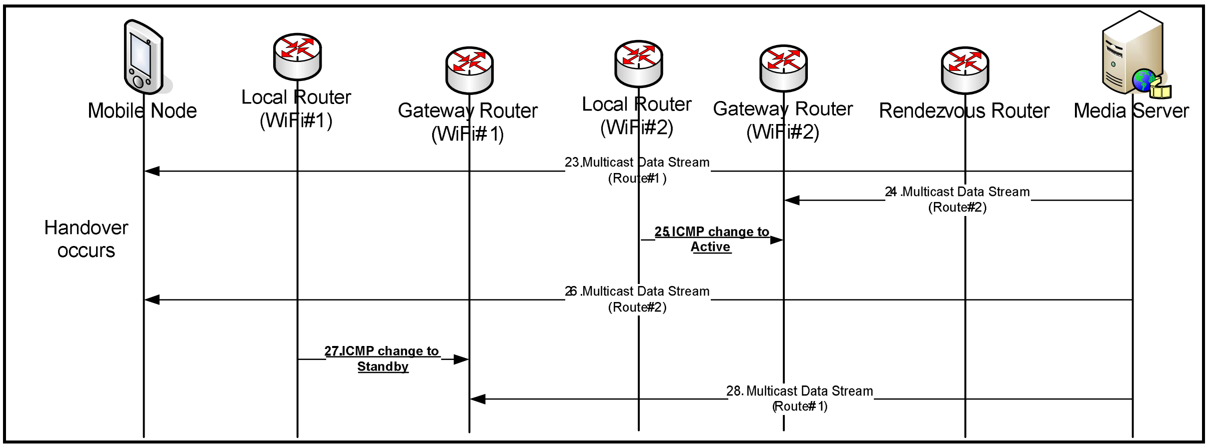

Figure 3-6 Handover process

The procedure for achieving this is as follows:

1. The local router in the WiFi#2 network will send a modified ICMP message to the gateway router to change mode from standby to action. This means that the new route for the multicast data stream is ready to connect. Moreover, this method should reduce handoff latency time.
2. After that the mobile node will start to receive multicast data stream via the new route.
3. The next step is to change the old route to standby mode by sending an ICMP change to Standby message to the gateway router in WiFi#1 network.
4. At this stage we still keep the old route from media server to the local router in WiFi#1 network in case the mobile node moves back to the old network. However, it will

have setting timeout for delete route.

3.5 Modified Protocol Message

In this research, there are some protocol messages that have been modified to support our designing framework, which are:

3.5.1 PIM Protocol Message

The message that we have been modified in PIM protocol is *"Join/Prune message"*. The format of Join/Prune message is shown in Figure 3-7. This message modified for keeping status join/standby message information which sends between local router in WiFi#2 and rendezvous router in the process number 20 in the Figure 3-5.

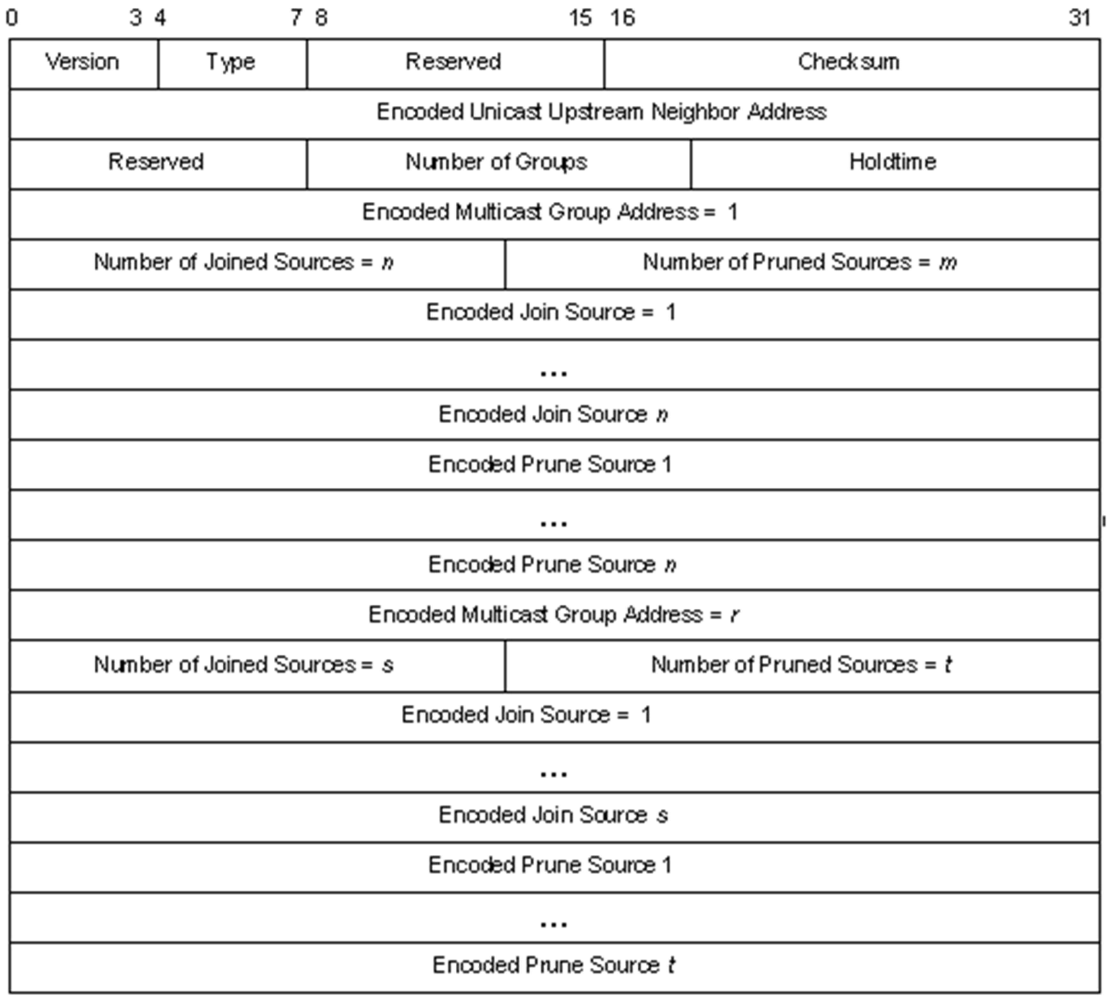

Figure 3-7 Join/Prune messages format [20]

3.5.2 ICMP Message

The ICMP message has been modified and adapted for controlling and changing the status of a second route when the handover happened. In our framework when handover occurs, local router in WiFi#2 will send the message to change the status to become active as in process number 25 in Figure 3-6. After that, the mobile node can continuously receive the multicast message from multicast tree without rebuilding the tree. We modified ICMP to support this strategy. The 3-8 is shown the standard ICMP message.

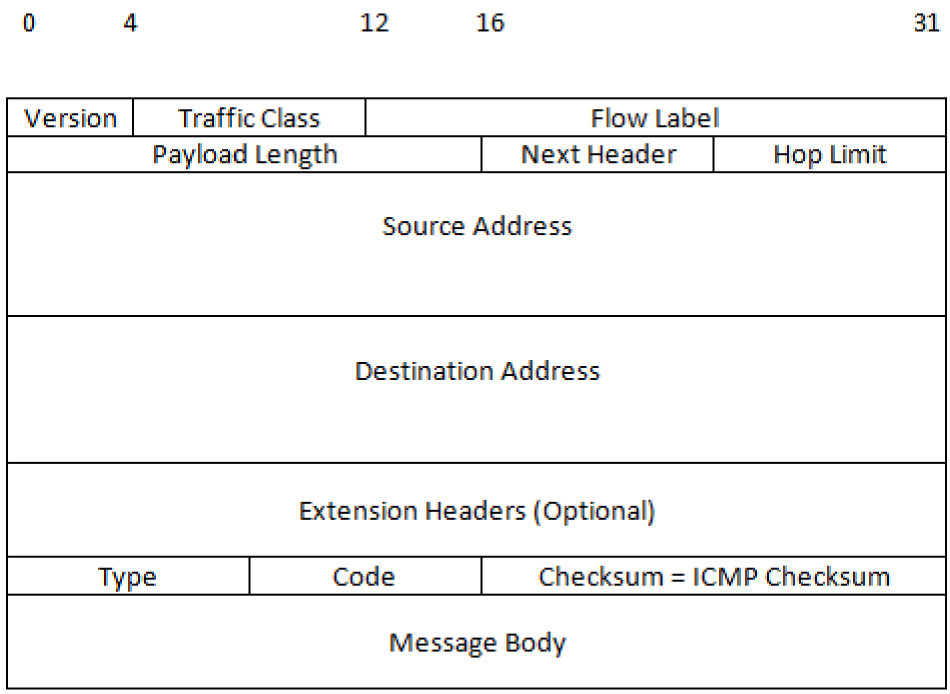

Figure 3-8 ICMP message format [35]

3.5.3 Mobile IP Message

The Mobile IP message is modified for sending to the local router in WiFi#2 network to ask for CoA address in advance. In process number 10 in Figure 3-4, we will modify an Agent solicitation Message of Mobile IP to ask for CoA address from local router in WiFi#2 network by changing the flag H to active. The format of Mobile IP message is shown in Figure 3-9.

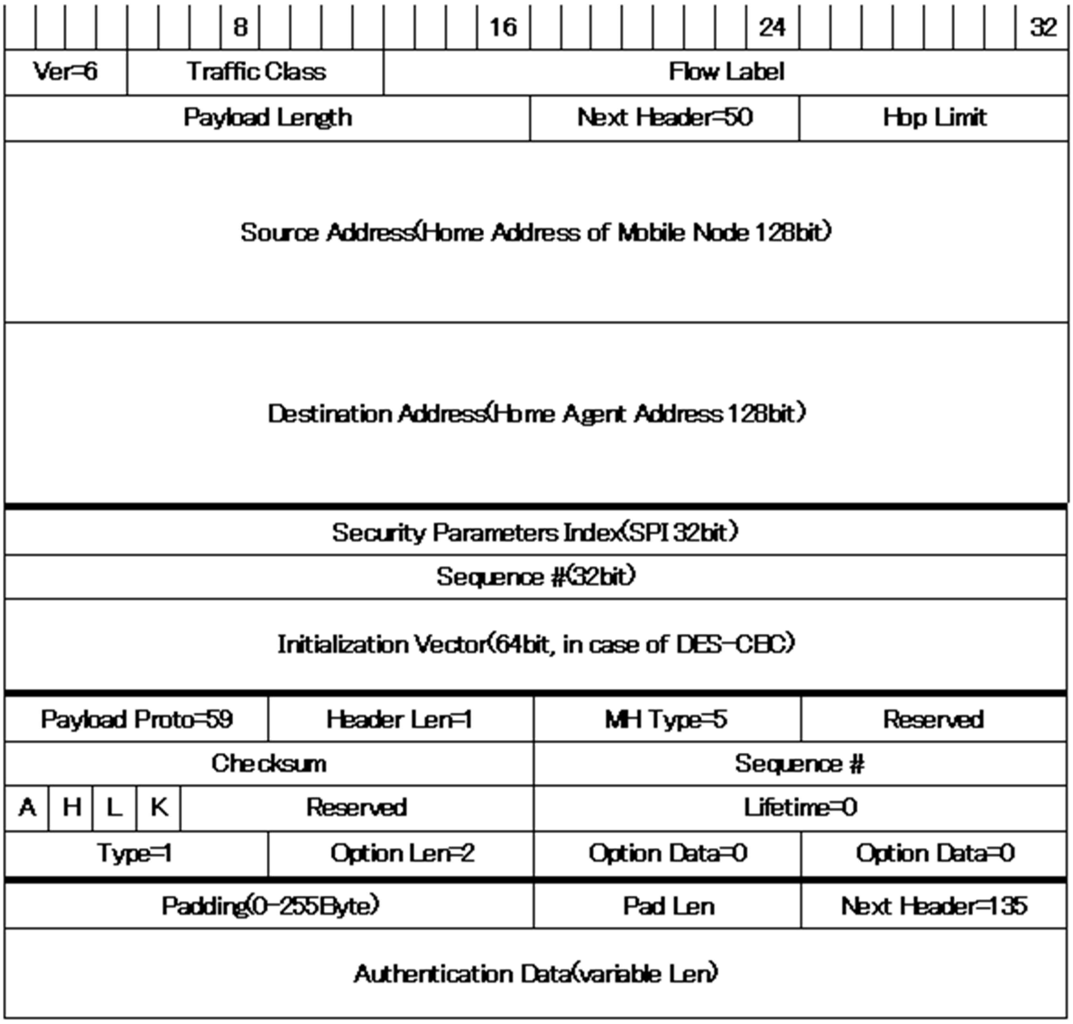

Figure 3-9 Mobile IP message format [32]

3.5.4 IGMP Message

In the designing framework, IGMP protocol has been modified to do a process of joining multicast tree for the second route before handover take place. This IGMP message will carry an *"IGMP Membership Report"* message as show in the process number 17 in Figure 3-5. The standard format of IGMP message is shown in Figure 3-10.

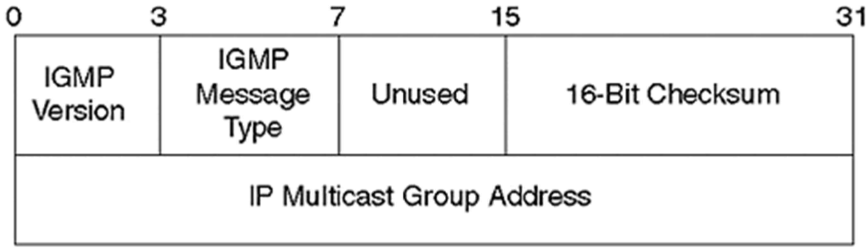

Figure 3-10 IGMP message format [6]

3.6 Summary

This chapter describes the network requirement, network architecture, the protocol process and modifying messages in detail. According to our goal we are trying to minimize the handover process in multicast mobile. The framework creates the reserve routes via neighbour zones, which are connected to the same multicast tree in advance. When handover occurs, the system only changes the status from standby to active and from active to standby mode.

We predict this strategy can reduce handover latency time because the network already has reserve paths, which are connected to neighbour zones. The delay will become only the time for changing the status of the reserve route. Hence, the handover latency time will become the time from local router in WiFi#2 to be detected by the mobile node including sending ICMP message to change the status until mobile node receives the multicast message. If it is compared with the previous methods that have been proposed, this strategy avoids many problems such as reconstruction of the multicast tree, network inactivity because it knows the new route in advance, multicast encapsulation/decapsulation because this method does not use tunnelling and so on.

Chapter 4 A Framework Simulation in OPNET Modeler

4.1 Introduction

In this chapter will present about a designed framework which simulating on OPNET Modeler software. The research is simulating those techniques and designed processes on OPNET Modeler software which is a licensing at University of Salford. OPNET Modeler software is a network simulation software and solution. This software provides for application and network management issues.

4.2 Network simulation

For doing research in the network field, network simulation software is a very useful and important tool. As researchers or protocol designers have to design and testing the system in simulation software before using it in a real network. There are many network simulations that widely used in networking research such as OMNET++ (Objective Modular Network Testbed in C++), NS-2 (Network Simulator version 2) and QualNet [69].

OPNET Modeler is generally used by researchers, developing protocol designers and so on. The OPNET software was funded in 1986 by Alain Cohen. OPNET stands for Optimizing Network Engineering Tools [70]. OPNET Modeler provides a comprehensive development environment which is powerful for instance simulation, data analysis, model design and etc. also it can support lot of technologies including local area network (LAN), mobile network, sensor network, wireless network and so on.

4.2.1 Basic Structure within OPNET Modeler

This is the workflow for OPNET Modeler. Normally, the researcher use these steps to build a network model, create the traffic, choose statistics and then run simulations.

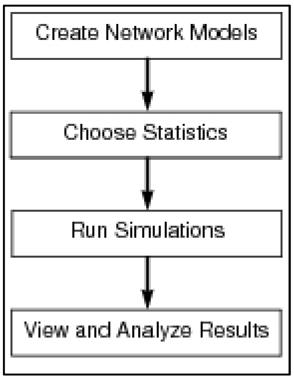

Figure 4-1 Basic step for creating network simulation

These 4 steps in Figure 4-1 consist of creating network environments which is including network devices and traffic, and then choose statistics that we want to study. Next step is run simulations. Finally, view and analyze the results. To complete these 4 steps, OPNET Modeler provides variety kinds of editor to support users as show below.

- ***The Project Editor***

This is a main area of OPNET simulation. We use this area to create network topology, generate traffic within network and view the results via this editor. Moreover, this area still covers about choosing statistics and running simulations.

- ***The Node Editor***

The user can define the behavior of each network object via *"Node Editor"*. In Node Editor of each model, the behavior is defined using different modules for example data storage, data creation, etc. A network object in OPNET Modeler is typically building up from multiple modules which define that object. The user can add their modules into the network object via Node Editor.

- ***A Network Model in the Project Editor***

The OPNET Modeler let user to design and create any elements of network as they wish. For instance, user can create node, link model, process models and build packet formats. Also, the user can create filters and parameters that they want to analyze.

➢ *Node Model*

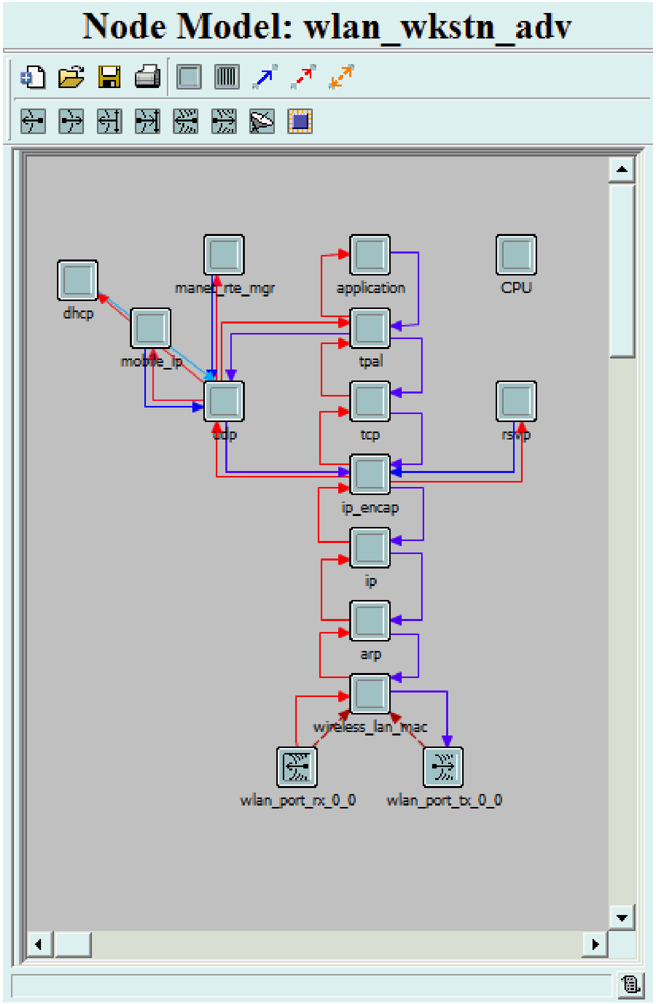

Figure 4-2 Node Mode example

➢ The Process Model Editor

The OPNET Modeler let user design and creates their process models via the *"Process Editor"*. The user can start from create node model in Node Editor and then they can build process model, which control the functionality of that node mode.

➢ Process Model

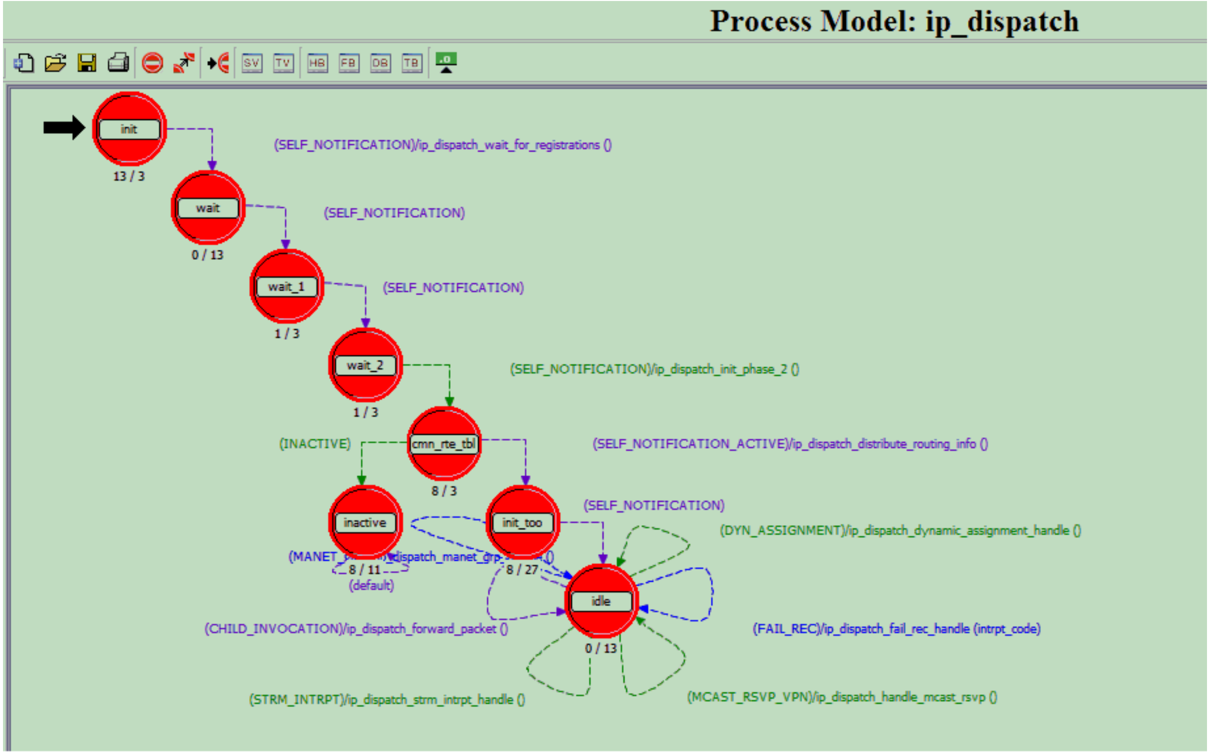

Figure 4-3 Process Mode example

4.3 Implementation of the Proposed Framework in OPNET Modeler

Due to the implementation of this research has been simulation environments and testing the performance of designing on OPNET Modeler software version 16.0 which is not supported multicast communication over IPv6 WiFi environment. Hence, the implementation and development of this thesis has been modifying based on IPv4 environment. However, the concept and designed of this framework can adapt to WiFi network both on IPv4 and IPv6 Networks.

4.3.1 Network Architecture

Normally, the first thing that has to start network simulation is OPNET Modeler is to create network architecture. The common start network topology that is used in this research is shown in Figure 4-4.

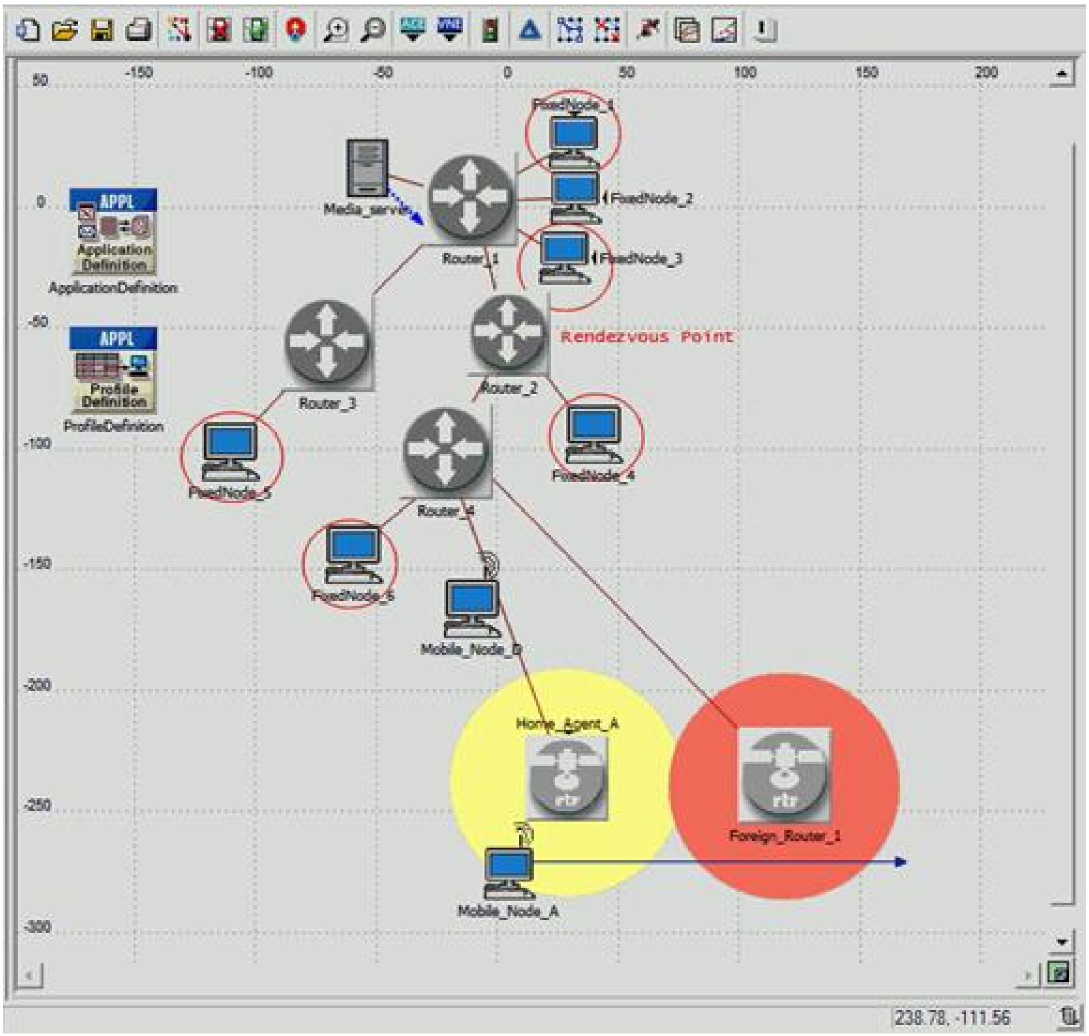

Figure 4-4 the common network topology that is used in this research.

4.3.2 Process Model

Therefore, we have to create a network environment and add some processes into standard protocol. So, we have to deal with process model many times.

4.3.2.1 Asking CoA in Advance process

In the designed framework, a mobile node will send an Agent Solicitation Message to foreign agent to ask for a CoA in advance. This process happens via the Mobile IP protocol. However in OPNET Modeler, this method will happen by creating an extra state and adding into the process model of mobile_ip_mn.

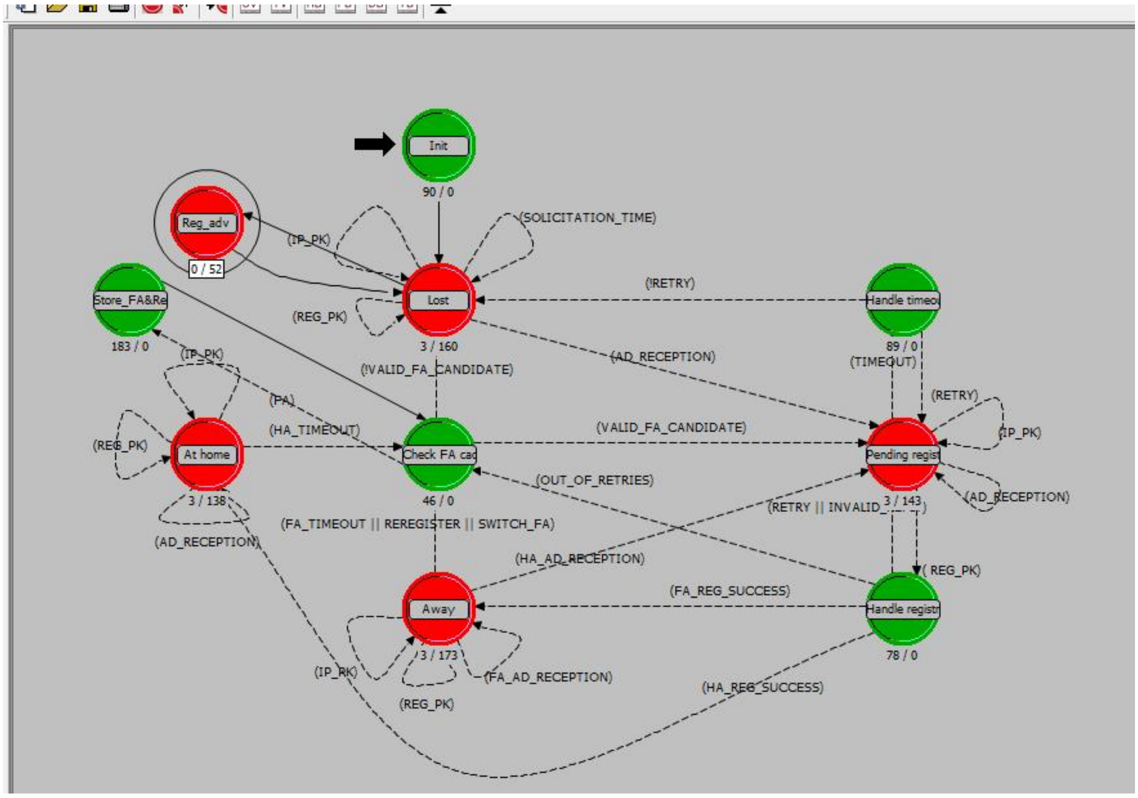

Figure 4-5 Process Model: mobile_ip_mn

Figure 4-5 is shown the Process Model of Mobile IP protocol within mobile node. To achieve the process of asking CoA in advance, state name "Reg_adv" have to be added. The coding of this state is shown in the Figure 4-6.

```c
static void
mip_mn_agent_solicit_pk_send_adv (void)
    {
    Packet*     solicit_pkptr_adv;
    double      solicit_interval;

    /** PURPOSE: Send the ICMP agent solicitation packet.**/
    /** REQUIRES: none. **/
    /** EFFECTS: Packet will be given to IP to handle.**/
    FIN (mip_mn_agent_solicit_pk_send (void));

    /* Time to send out the solicitation. */

    usleep(10000000); // 10 secs
    solicit_pkptr_adv = op_pk_create_fmt ("mobile_ip_irdp_solicit");

    /* Send the packet out. */
    module_data->ip_ptc_mem.child_pkptr = mip_sup_irdp_pkt_encapsulate
        (solicit_pkptr_adv, home_address, subnet_bcast_addr, IcmpC_Type_IRDP_Sol);

    /* Record some stats. */
    op_stat_write (irdp_sent_pkts_sh, 1.0);
    op_stat_write (irdp_sent_bits_sh, op_pk_total_size_get (module_data->ip_ptc_mem.child_pkptr));
    op_stat_write (g_irdp_sent_bits_sh, op_pk_total_size_get (module_data->ip_ptc_mem.child_pkptr));
    op_stat_write (g_irdp_sent_bits_sh, 0.0);

    /* Invoke IP to handle the packet. */
    op_pro_invoke (proc_info_struct_ptr->ip_phndl, OPC_NIL);

    /* Schedule the next transmission. */
    if (++solicit_count > 3)
        {
        solicit_interval = MipC_MN_Solicit_Min_Interval * pow (2.0, (double) (solicit_count - 3));

        if (solicit_interval > MipC_MN_Solicit_Max_Interval)
            {
            solicit_interval = MipC_MN_Solicit_Max_Interval;
            }
        }
    else
        {
        solicit_interval = MipC_MN_Solicit_Min_Interval;
        }

    solicit_timer_ehndl = op_intrpt_schedule_self (op_sim_time () + solicit_interval, MipC_MN_Timer_Solicit);

    FOUT;
    }
```

Figure 4-6 The coding of “Reg_adv” state

4.3.2.2 Joining Multicast using CoA Address

This process model has been called after the mobile node received CoA in advance and then tries to build another route to multicast tree by using CoA address. This stage has been extended from IGMP process model. To achieve this, we have to create a process state adding into IGMP process model name “JOIN_ADV” as is shown in Figure 4-7. The coding of process state is presented in Figure 4-8.

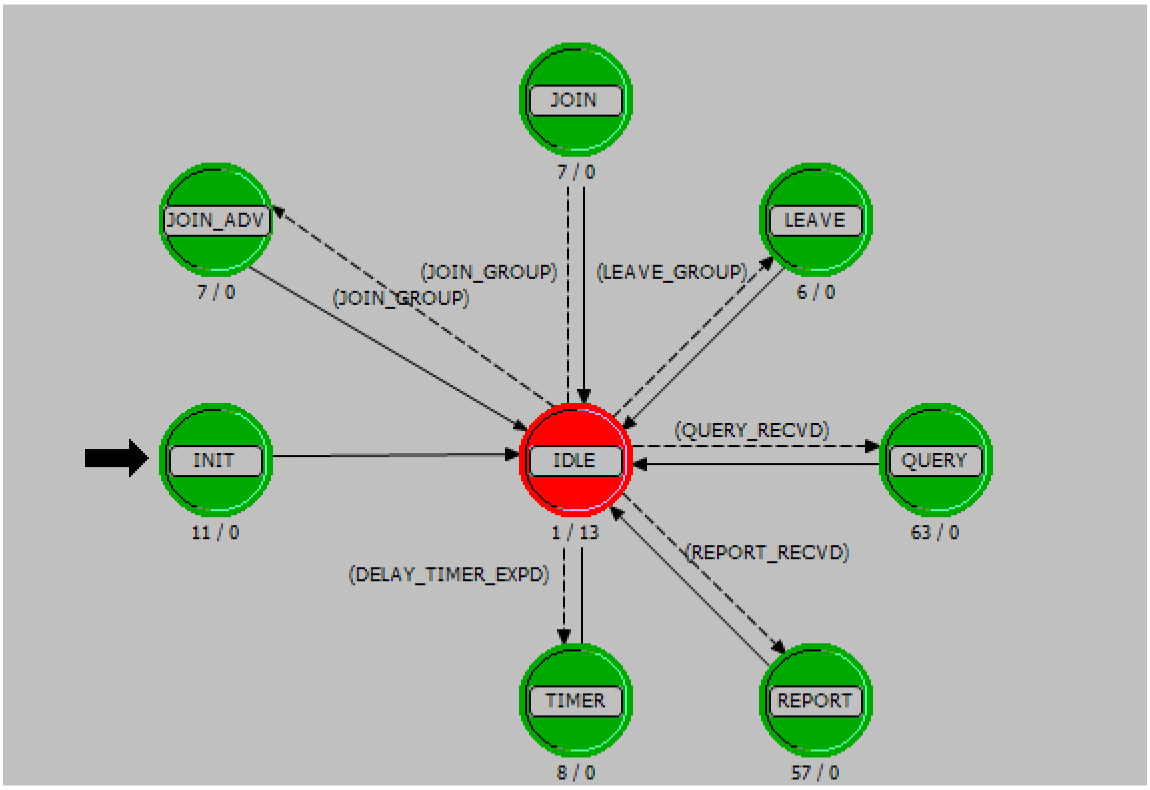

Figure 4-7 Process Model: IGMP host

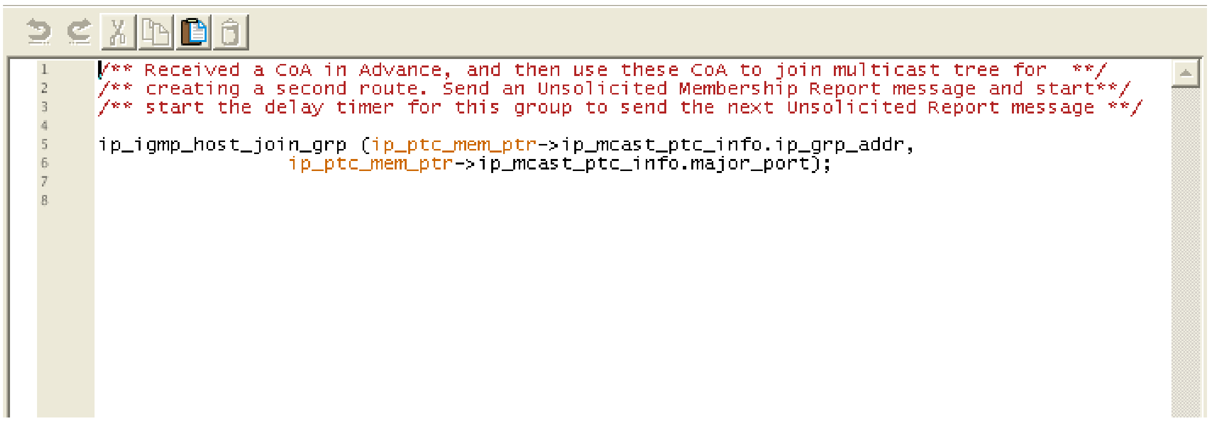

Figure 4-8 The coding of “JOIN_ADV” state

4.3.2.3 Re-join Multicast

When the mobile node has been realized that, now it is in a foreign agent. The process re-join will call multicast joining state in process model “ip_igmp_rte_grp” to send a message to join multicast group again.

```
/** time. Notify the router and start the group membership timer    **/

/* Generate trace messages  */
if (LTRACE_IGMP)
    {
    ip_address_print (ip_addr_str, ip_grp_addr);
    sprintf (msg0, "IP Group Address        :       %s", ip_addr_str);
    sprintf (msg1, "IP Interface            :       %d", ip_interface);
    op_prg_odb_print_major ("Received an IGMP Membership Report message for: ", msg0, msg1, OP
    op_prg_odb_print_major ("A local host has joined the above group. Notifying PIM-SM router
    }

/* Notify the router */
ip_igmp_rte_grp_notify_router (IpC_Igmp_Rte_Pim_Sm_Notify_Plus);

/* Start the group membership timer only if simulation efficiency is disabled */
if (igmp_attrs_ptr->igmp_sim_efficiency == OPC_FALSE)
    {
    /* Generate trace messages  */
    if (LTRACE_IGMP)
        {
        sprintf (msg2, "Starting the Group Membership timer with value, %d for the above group
        op_prg_odb_print_major (msg2, OPC_NIL);
        }
    ip_igmp_rte_grp_start_timer (IPC_IGMP_RTE_GRP_TIMER, igmp_attrs_ptr->grp_member_interval);
    }

/* Tag this connection process for debugging purposes.  */
if (op_sim_debug () == OPC_TRUE)
    {
    ip_address_print (ip_addr_str, ip_grp_addr);

    sprintf (msg0, "Group address: %s\tInterface index: %d", ip_addr_str, ip_interface);

    op_pro_tag_set (op_pro_self (), msg0);
    }
```

Figure 4-9 the coding for re-join multicast

4.3.2.4 Keeping Multicast Route

After the mobile node created other multicast routes, the mobile node have to keep other routes become Standby mode, only one route at a time being Active mode. To achieve this goal, we modified Join/Prune message in PIM protocol to keep these multicast routes alive. The coding of this process state is shown in Figure 4-11.

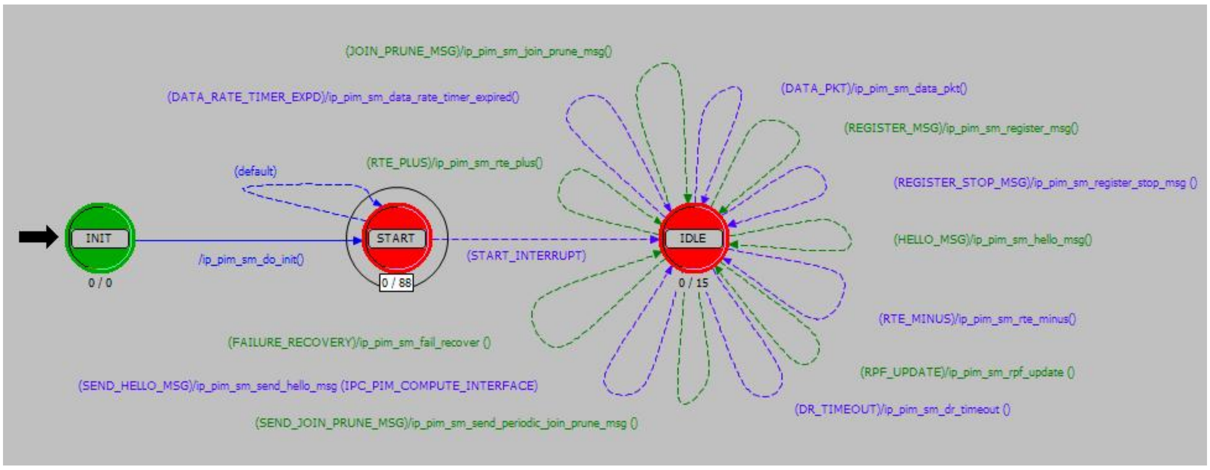

Figure 4-10 Process Model: PIM-SM protocol

```
/* Send the hello messages.                        */
ip_pim_sm_send_hello_msg (IPC_PIM_ALL_INTERFACES);

/* Join/Standby message to keep multicast route.    */
/* Bootstrap candidate RP information. */
/* static and auto-rp configuration.                */
if ((boostrap_support == OPC_TRUE) && (bootstrap_rp_lptr != OPC_NIL))
    {
    ip_pim_rp_lists_merge (bootstrap_rp_lptr);
    }
else if ((boostrap_support == OPC_FALSE) && (bootstrap_rp_lptr != OPC_NIL))
    {
    /* There were some candidate RPs configured for bootstrap   */
    /* However, there is no preferred BSR. Log a message.        */
    ipnl_protwarn_mcast_bootstrap_not_supp ();
    }

if (log_call_scheduled == OPC_FALSE)
    {
    /* Keep Standby the route.  */

    op_intrpt_schedule_call (OPC_INTRPT_SCHED_CALL_ENDSIM, 0,
        ipnl_protwarn_mcast_endsim_log_write, OPC_NIL);

    log_call_scheduled = OPC_TRUE;
    }

/* Check for the list of Active/Standby mode    */
ip_pim_sm_rp_table_to_ot_export_schedule ();

/* Check if Group mapping has been configured to export to OT reports   */
ip_pim_sm_group_table_to_ot_export (OPC_NIL, 0);
}
```

Figure 4-11 The coding for keeping multicast route

4.3.2.5 Store CoA addresses

When the mobile node moves, the mobile node starts to receive CoA address along the path. However, in the large network which consists of many routers in different zones, the mobile node also receive multiple CoA address from foreign router. Hence, the mobile node needs to have a process of store and process multiple CoA addresses. To solve this issue, we had modified Mobile IP protocol to have the process of store and retrieve CoA address on the mobile node. We have created the state named “Store_FA&Re” state which is shown in Figure 4-5 to solve this issue.

```
/* Access the packet. */
reg_pkptr = op_pk_get (input_strm);

/* Create an ICI for communication to Mobile IP process. */
reg_ici_ptr = op_ici_create ("mobile_ip_reg_ici");

/* First the packet format. */
op_pk_format (reg_pkptr, pk_format_name);

if (!strcmp (pk_format_name, "mobile_ip_reg_req"))
    {
    /* Request! */
    reg_type = MipC_Reg_Type_Req;

    /* Retrive information from the packet. */
    op_pk_nfd_access (reg_pkptr, "Care-of Address", &care_of_address);
    op_pk_nfd_access (reg_pkptr, "Lifetime", &lifetime_req);
    op_pk_nfd_access (reg_pkptr, "S", &simultaneous_binding);

    /* Set the values on the ici to mobile ip. */
    op_ici_attr_set (reg_ici_ptr, "care_of_address", care_of_address);
    op_ici_attr_set (reg_ici_ptr, "lifetime_req", lifetime_req);
    op_ici_attr_set (reg_ici_ptr, "s", simultaneous_binding);
    }
else
    {
    if (!strcmp (pk_format_name, "mobile_ip_reg_reply")) /* strcmp function is for compare
        {
        /* Reply! */
        reg_type = MipC_Reg_Type_Reply;

        /* Retrive information from the packet. */
        op_pk_nfd_access (reg_pkptr, "Lifetime", &lifetime_grant);
        op_pk_nfd_access (reg_pkptr, "Extension", &care_of_address); /* Use extension for F
        op_pk_nfd_access (reg_pkptr, "Code", &reply_code);

        /* Set the values on the ici to mobile ip. */
        op_ici_attr_set (reg_ici_ptr, "lifetime_grant", lifetime_grant);
        op_ici_attr_set (reg_ici_ptr, "extension", care_of_address);
        op_ici_attr_set (reg_ici_ptr, "reply_code", reply_code);
        }
    else
        {
        op_sim_end ("Unknown packet format received.","","","");
        }
    }
```

Figure 4-12 the coding of “store_FA&Re” state

State “store_FA&Re” has two processes of designed framework in there. That is keeping CoA from foreign router, and another process is retrieving CoA address. Some part of coding state is shown in Figure 4-12.

```
/* Structure of each CoA that storage in Mobile node. */
reg_ici_ptr = op_intrpt_ici ();
op_ici_attr_get (reg_ici_ptr, "reg_type", &reg_type);
switch (reg_type)
    {
    case MipC_Reg_Type_Req:
        {
        /* get additional attributes from ici. */
        op_ici_attr_get (reg_ici_ptr, "care_of_address", &care_of_address);
        op_ici_attr_get (reg_ici_ptr, "lifetime_req", &lifetime_req);
        op_ici_attr_get (reg_ici_ptr, "s", &simultaneous_binding);


        reg_pkptr = op_pk_create_fmt ("mobile_ip_reg_req");
        op_pk_nfd_set (reg_pkptr, "Care-of Address", care_of_address);
        op_pk_nfd_set (reg_pkptr, "Lifetime", lifetime_req);
        op_pk_nfd_set (reg_pkptr, "S", simultaneous_binding);

        break;
        }

    case MipC_Reg_Type_Reply:
        {
        /* get additional attribute from ici. */
        op_ici_attr_get (reg_ici_ptr, "lifetime_grant", &lifetime_grant);
        op_ici_attr_get (reg_ici_ptr, "dest_address", &dest_address);
        op_ici_attr_get (reg_ici_ptr, "reply_code", &reply_code);

        reg_pkptr = op_pk_create_fmt ("mobile_ip_reg_reply");
        op_pk_nfd_set (reg_pkptr, "Lifetime", lifetime_grant);
        op_pk_nfd_set (reg_pkptr, "Extension", dest_address);
        op_pk_nfd_set (reg_pkptr, "Code", reply_code);

        break;
        }
```

Figure 4-13 the coding of structure of CoA address list

Figure 4-13 presents the coding of the process in “Store_FA&Re” state. This part is shown the structure of each CoA information that has been stored in the mobile node responsible by Mobile IP message.

```
/* Each CoA address do Building update to Home Agent*/
/* Also take care of identification field as well. */
op_ici_attr_get (reg_ici_ptr, "identification", &identification);
op_pk_nfd_set (reg_pkptr, "Identification", identification);

/* Record received stats. */
op_stat_write (reg_sent_bits_sh, op_pk_total_size_get (reg_pkptr));
op_stat_write (reg_sent_pkts_sh, 1.0);

/* Access the necessary common attributes. */
op_ici_attr_get (reg_ici_ptr, "home_address", &home_address);
op_ici_attr_get (reg_ici_ptr, "home_agent", &ha_address);
op_ici_attr_get (reg_ici_ptr, "dest_address", &dest_address);

/* Set the values on the packet. */
op_pk_nfd_set (reg_pkptr, "Home Address", home_address);
op_pk_nfd_set (reg_pkptr, "Home Agent", ha_address);

/* Send it on to UDP module for delivery. */
op_ici_attr_set (command_ici_ptr, "local_port", MipC_Reg_UDP_Port_Num);
op_ici_attr_set (command_ici_ptr, "rem_port", MipC_Reg_UDP_Port_Num);
op_ici_attr_set (command_ici_ptr, "rem_addr", dest_address);

/* Install the ici and send.              */
op_ici_install (command_ici_ptr);
op_pk_send_forced (reg_pkptr, output_strm);

/* Get the status indication from the ici */
op_ici_attr_get (command_ici_ptr, "status", &ind);

/* Clean up. */
op_ici_destroy (reg_ici_ptr);
```

Figure 4-14 the coding of Binding Update multiple CoA address to home agent

Normally, the mobile node has to report every CoA address that received from foreign agent to home agent. Hence, in the framework the mobile node has to do the same but in advance. So, this process is part of reducing handover latency because the binding update process had been doing in advance before handover occur. The coding of this process is shown in Figure 4-14.

4.4 Summary

This chapter presents the fundamentals of OPNET Modeler software, the implementation that has been made to achieve our designed framework, the state diagram, some coding of the process. The full coding of this research has been attached in Appendix B.

However, some process of the designed framework, we do not need coding program. We can handle it by setting a value of attribute of the protocol such as IP parameter and wireless parameter. Also, the designed process setting a second multicast route to Standby/Active mode can be done by changing the parameter flag at gateway router.

Chapter 5 Simulation Scenarios, Results and Evaluation

5.1 Introduction

The previous chapter described the implementation of a framework, which is simulated using OPNET Modeler software. In order to evaluate the performance of the designed network framework, we have to simulate a network environment within network simulation software, as it is impossible to test the designed network in the real network environment. This chapter will present the simulation scenarios, results and evaluation performance of the framework from the research project comparing it with the standard network. In the area of computer networking research, OPNET Modeler software is widely and reliably used for testing, debugging and performance evaluation of extended protocol and developing networks.

To prove that this framework can reduce handover latency and reduce packet delay within a wireless network, a variety of network scenarios have been produced. Some scenarios were created to measure robustness on the network, some of them for testing about scalability and so on. Every scenario will be compared with the standard network environment.

5.2 Scenario 1: The Performance of Unicast and Multicast Mechanism

5.2.1 Scenario1: Scenario Description

At first, we evaluated the performance of the unicast and multicast mechanism. The statistical parameters that we focused on are throughput in the links and load at wireless router.

Normally, the streaming video formats can range from 128*120 pixels to beyond 1920*1080 pixels for HD (High Definition) standards. The popular streaming video service such as

YouTube has 320*240 pixel resolution. For the video frame inter-arrival rates can be from 10 up to 30 frames per second [37]. However, the higher the video resolution, the higher the raw video content size, so that means it will affect the bandwidth on the network and packet delays.

In this simulation model, a media server that connects within the network exports the multicast streaming video traffic to the clients. The frame size of video is 128*120 pixels and the video frame inter-arrival rate is 10 frames/sec (fps). In the wireless network, there are four subnets and each subnet has only one wireless router. For the sake of simplicity, we consider that there is only one multicast group in the network. Also, there are fixed and mobile node clients in this scenario. During the simulation, clients can join or leave the multicast group at any time.

5.2.2 Scenario1: Simulation Topology

At the beginning, the network topology combines with wireline and wireless networks as presented in Figure 5-1.

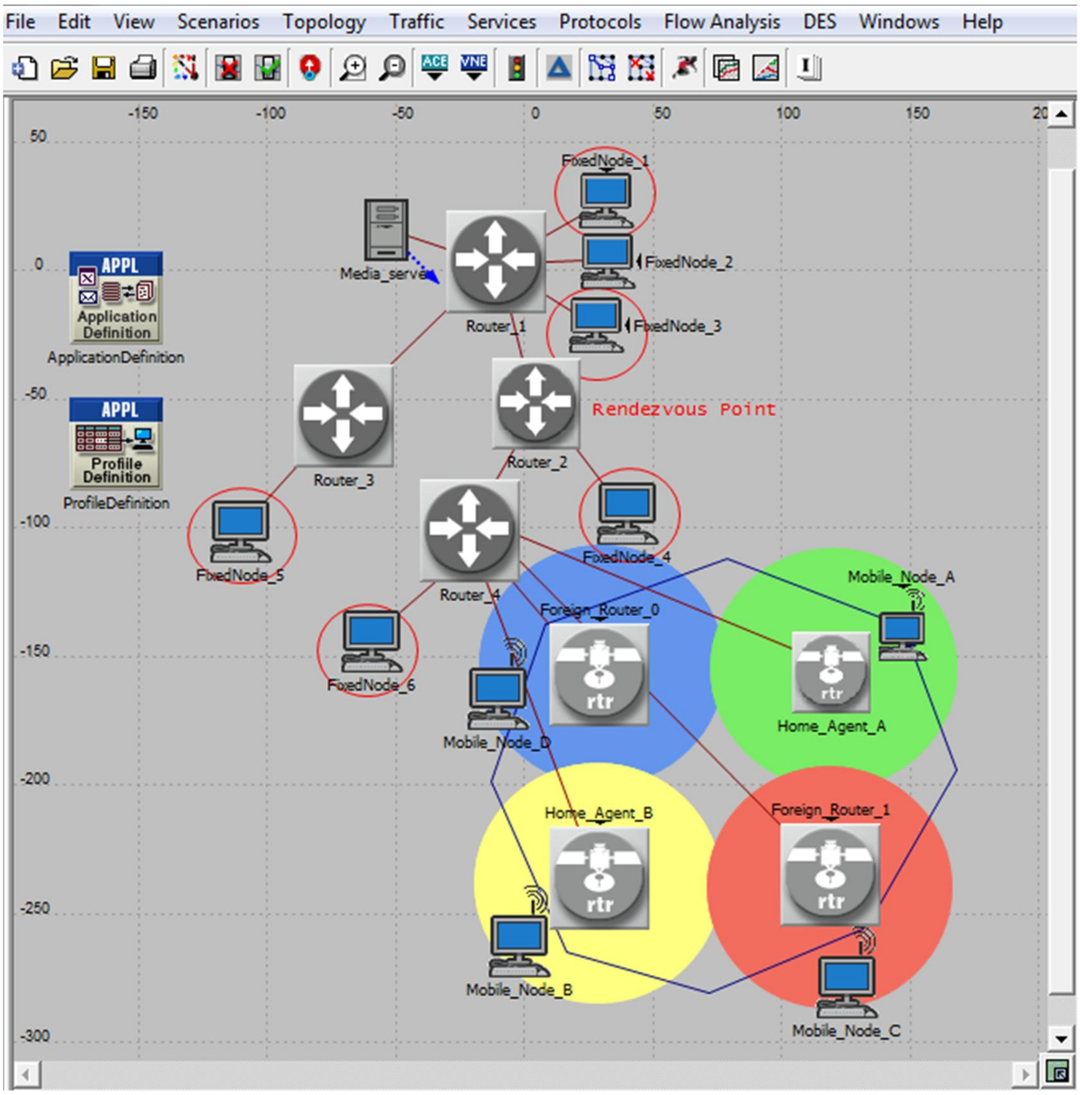

Figure 5-1 Scenario 1: Simulation Topology

In order to examine the performance, we have moved mobile node B and C to connect to the network via Home Agent_A access point. Hence the experiment at this stage is that the mobile nodes A, B and C are connected to Home Agent_A access point and are required to receive the data from the media server.

5.2.3 Scenario1: Simulation Results and Evaluation

In the simulation, all mobile nodes join the multicast group at 100 seconds and leave at the end of the simulation. During simulation time, we assume that all packets are delivered correctly to all receivers without any disruptions to the service. The correlation between unicast and multicast are tested. The results are presented in Figure 5-2 and 5-3.

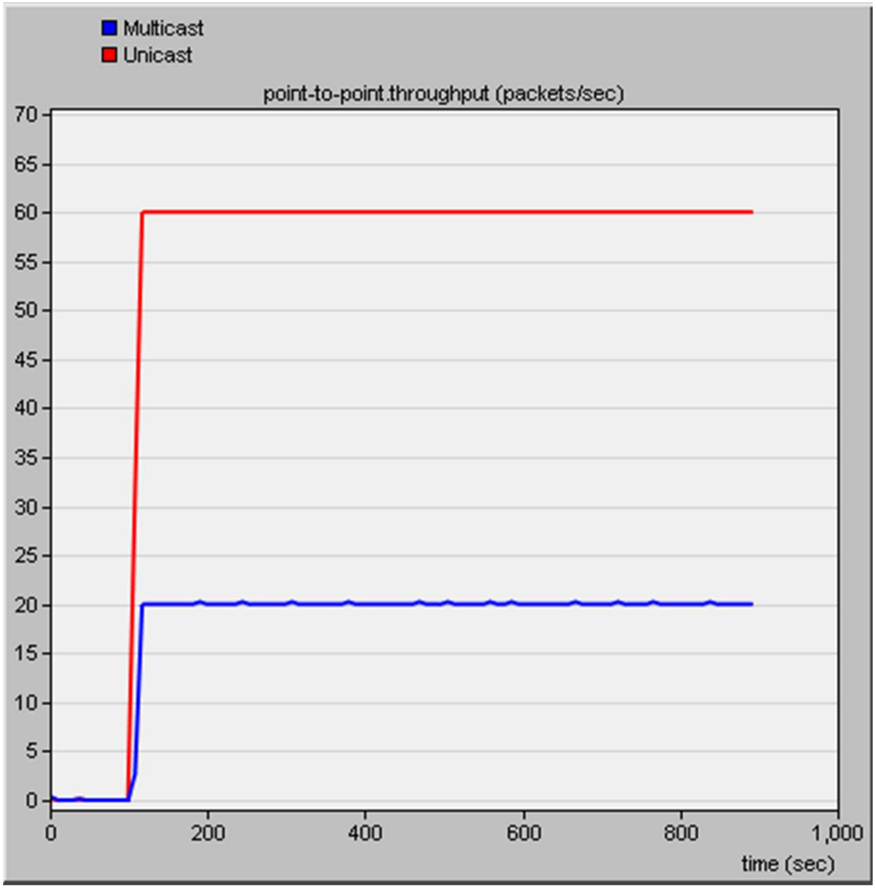

Figure 5-2 Throughputs between Router4 and Home Agent_A access point

We can observe from Figure 5-2 that, the throughput of multicast transmission is only one third compared with the throughput in the same links when multiple unicast transmission is used. This means that the bandwidth consumption increases. It is the most important benefit of the multicast data delivery.

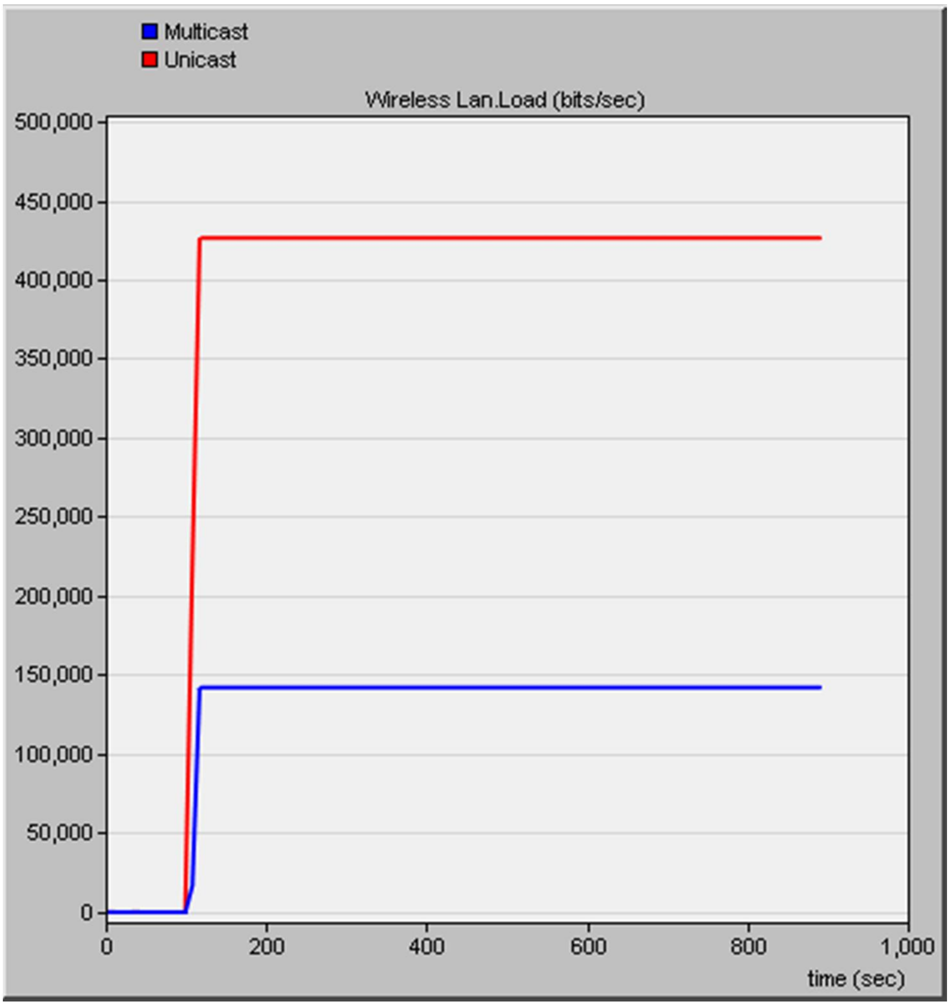

Figure 5-3 Load at Home Agent_A access point

Figure 5-3 is shown the load at Home Agent_A access point compared between unicast and multicast transmission in units of bits/sec. It can be seen from the data that multicast mechanism reduces traffic and load at access point. Figure 5-4 illustrates CPU utilization at access point. It is apparent from this graph that CPU utilization of unicast is higher than the multicast mechanism. This result may be explained by the fact that the access point of unicast has to process 3 unicast copies while multicast processes is concerned with only one set of data. However, the other processes such as encapsulate/ decapsulate packet, routing process and forwarding packet are still the same method. This can explain why CPU utilization of unicast differs from multicast by only 0.04 %.

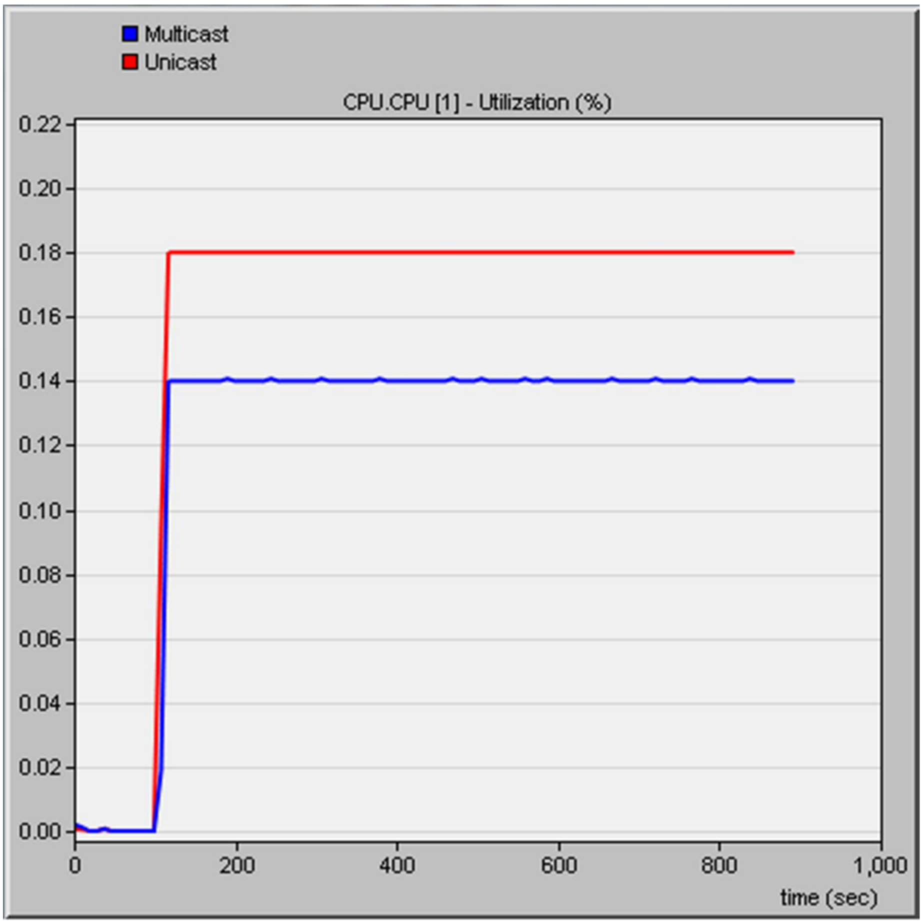

Figure 5-4 CPU Utilization at Home Agent_A access point

Another aspect that we examined was when a mobile node moves while receiving multicast data. From Figure 5-1, mobile node A requests to receive video multicast from the media server at 100 seconds. During the simulation, mobile node A moves in a counter-clockwise trajectory roaming through all four access points in the network (the path represented in a blue line in Figure 5-1). At 800s, the mobile node A leaves the multicast group. The other mobile nodes did not move and also received multicast data through the access point as in the figure. Figure 5-5 compares the video traffic received at mobile node A and B.

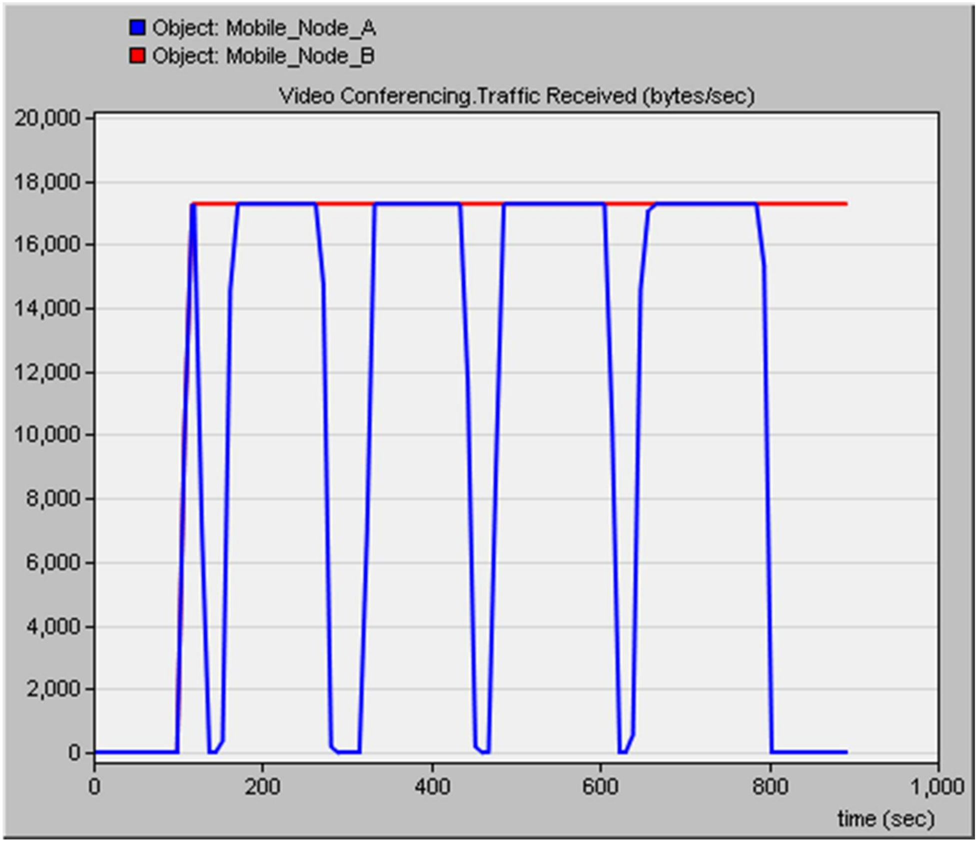

Figure 5-5 Video traffic received at mobile node A compared with B

Figure 5-5 depicts clearly that mobile node B, which is a stationary node, continuously received the multicast traffic until the end of the simulation. On the other hand, mobile node A moves through all four access points, and when handover occurred the traffic received dropped significantly.

In this case, it may completely disconnect the access point from the link layer. Thereafter, it needs to restart the process of performing an IP reconfiguration in the network layer and binding updates to home agent to its infrastructure. Until completion of all these operations the mobile node is likely to experience disruptions or disturbances of application, as the results of packet loss, jitter and delay increase. After the handover process is finished, the node will pick up multicast data again [47].

$$T_{handoff} = T_{L2} + T_{local\text{-}IP} + T_{BU} \tag{5-1}$$

The T_{BU} is Binding update latency time. In order to examine the multicast service continuity during the mobile node mobility procedures, therefore we simulated the following scenario. From Figure 5-1 we move the mobile node D to connect to Home Agent _B access point. Now there is no client in Foreign_Router_1 subnet. The mobile node A still moves in the same path. The result of this scenario is shown in Figure 5-6.

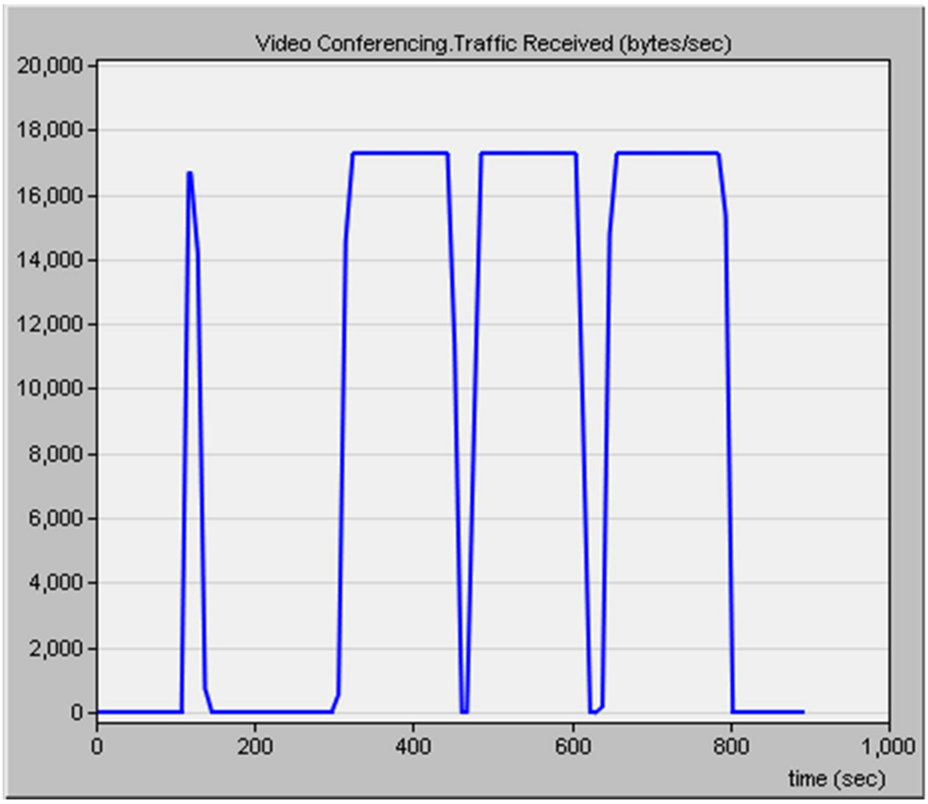

Figure 5-6 Video traffic received at mobile node A

Figure 5-6 is shown that mobile node A did not receive the multicast traffic whilst connected to Foreign_Router_1 zone. This means that mobile node A did not re-join the multicast group during that time. The mobile node A only joins the multicast group once the simulation starts, which is in Home_Agent A zone. Moreover, the mobile node A will receive multicast traffic if that zone already has a member in the same multicast group.

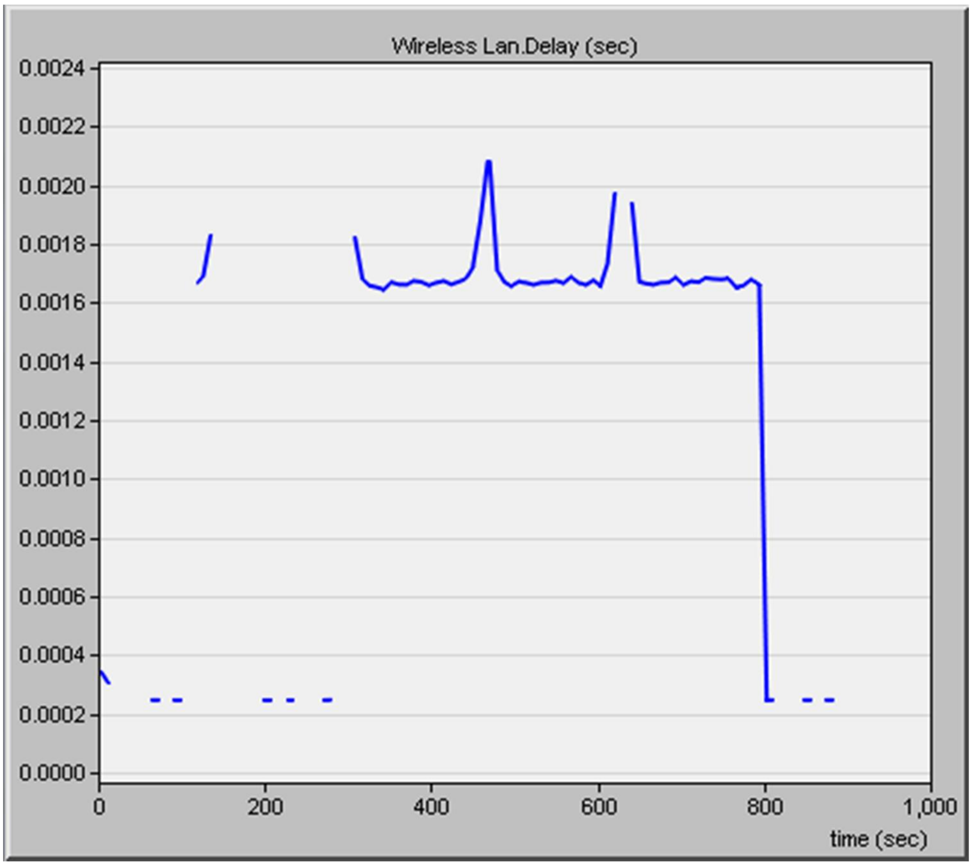

Figure 5-7 Delay at mobile node A

Figure 5-7 is shown the delay on mobile node A. It depicts clearly that the delay increased when the handover process happened. From this result it supports our idea that if we want to reduce the handover process time and also improve the performance, it should have the process of registering the multicast group in advance.

5.3 Scenario 2: Simple Network

5.3.1 Scenario2: Scenario Description

In the second scenario, we focused on multicast mobility handover occurring between modified framework and comparing with standard network environments. In this scenario we have included the process of multicast re-join in the design framework. Therefore, the aim of this scenario is to measure how much difference of re-join process affects the performance of the network.

5.3.2 Scenario 2: Network Topology

The network topology of this scenario still combines the wireline and wireless network. There are two WiFi zones in this topology, which are Home Agent_A and Foreign_Router_1. In Home_Agent_A zone, Mobile_Node_A is a member in this zone. Mobile_Node_A receives multicast data from Media_Server, and then moves to the Foreign_Router_1 zone. During mobility, Mobile_Node_A still continuously receives multicast packets from the Media_Server. In this scenario, there is no function of Mobile IP protocol involved. Figure 5-8 is shown the network topology of this scenario.

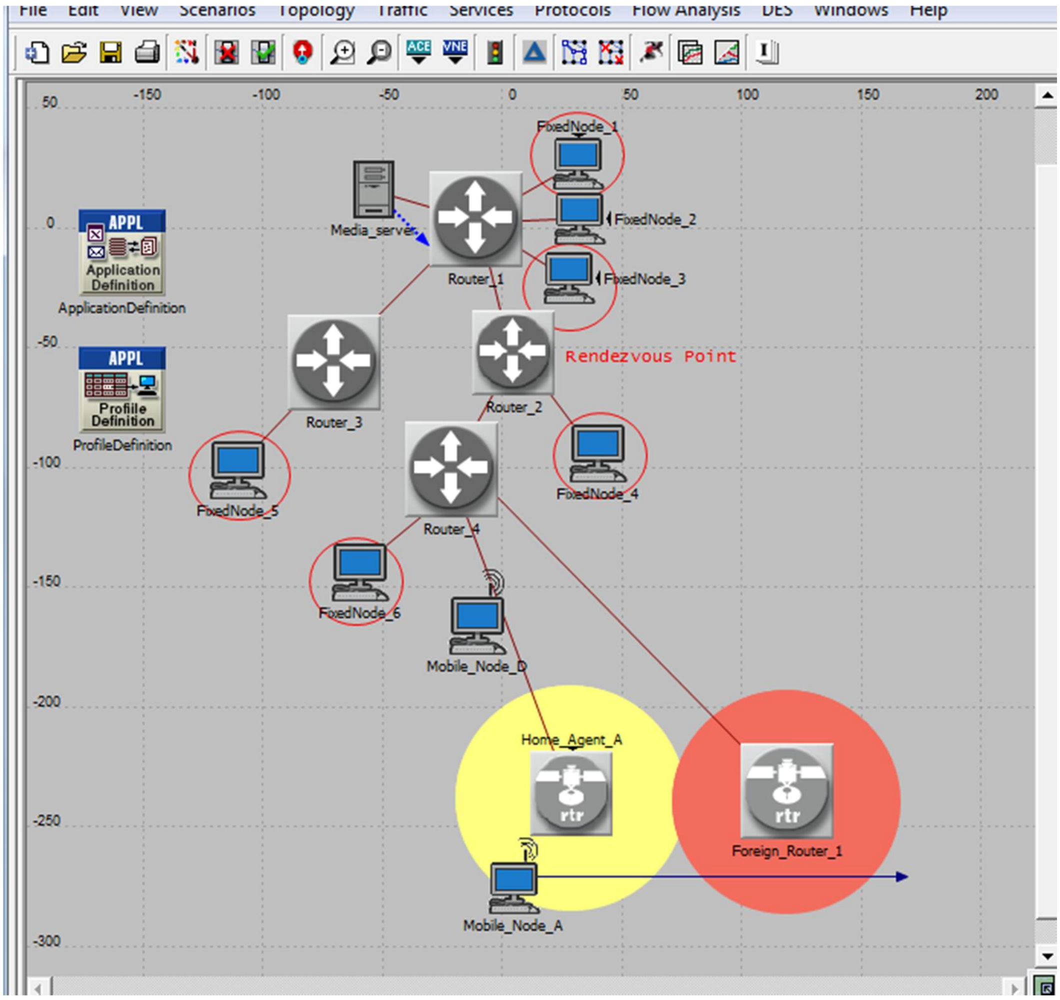

Figure 5-8 Scenario 2: Network Topology

5.3.3 Scenario2: Simulation Results and Evaluation

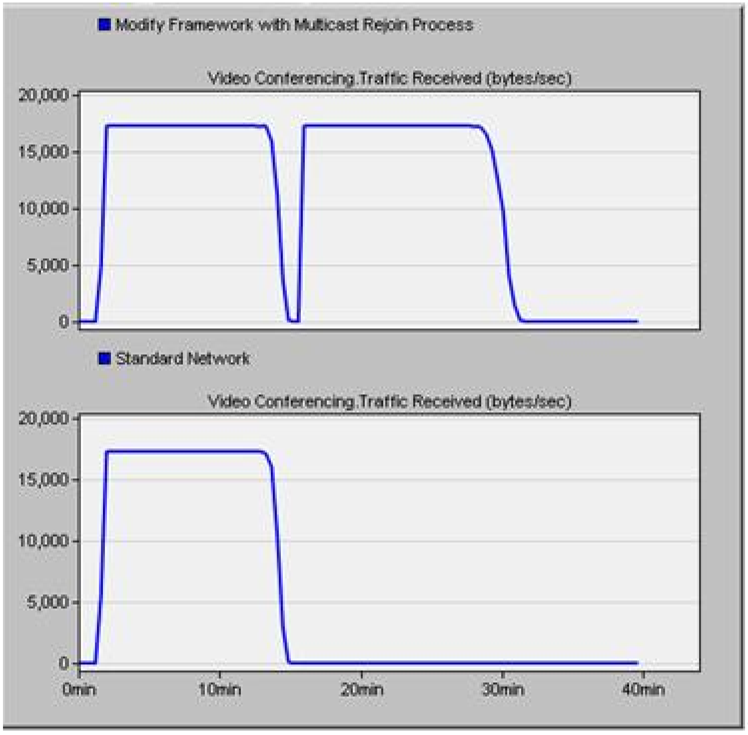

Figure 5-9 Scenario 2: Traffic received at Mobile_Node_A

The first graph in Figure 5-9 is multicast data that has been received at Mobile_Node_A during the simulation. A gap in the graph happens when Mobile_Node_A moves from Home Agent_A to Foreign Router_1 networks. The result is shown that Mobile_Node_A completely lost connection when handover occurred. However, the modified framework has the multicast re-join process, so a joining message has been sent to Foreign Router_1. Mobile_Node_A can get back to receive multicast data until it leaves multicast group. What is interesting in this data is that there is not the multicast re-join process in the standard network, hence Mobile_Node_A cannot receive traffic when leaving from the home agent network.

5.4 Scenario 3: Mobile IP and multicast Re-join

5.4.1 Scenario3: Scenario Description

This scenario intends to inspect the effect of Mobile IP protocol when handover occurs. The infrastructure network of scenario 3 is still similar to the previous scenario. However, some parameters might change. In the modified framework, the Mobile IP protocol has been enabled. Also, there it still has the process of multicast re-join in this scenario. The result will be compared with the standard network.

5.4.2 Scenario3: Simulation Results and Evaluation

The blue graph in Figure 5-10 is the result of the video conferencing traffic that Mobile_Node_A received. It can be seen that it still has a gap of handover. However, the gap is narrower than in scenario 2 and does not completely lose the connection. This is because Mobile IP protocol has a mobility function to support connection. In the red graph, there are no functions of multicast re-join and Mobile IP was not enabled. Hence, the connection was completely lost in this case.

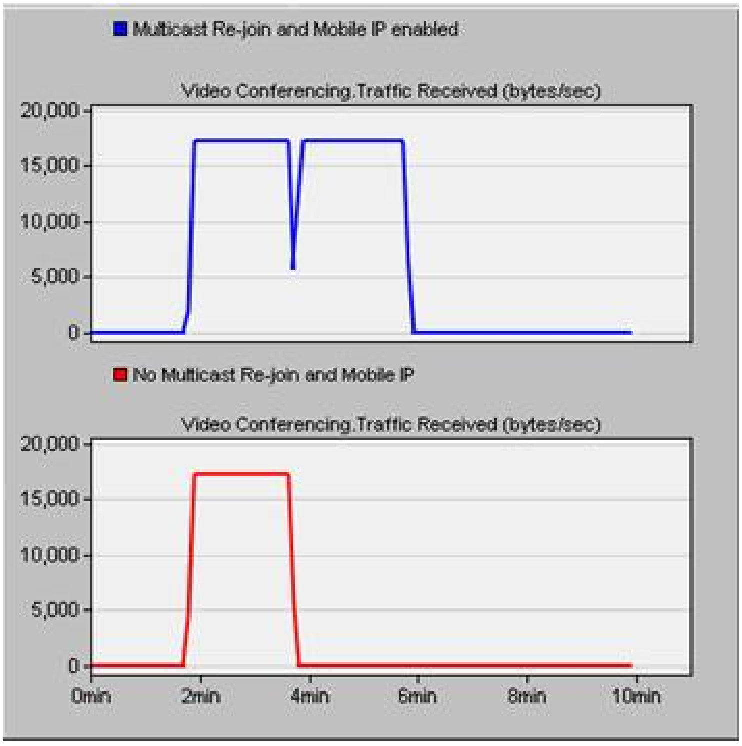

Figure 5-10 Scenario 3: Traffic received at Mobile_Node_A

5.5 Scenario 4: Care of Address in Advance

5.5.1 Scenario4: Scenario Description

This scenario focused on the advantage of the process of registering a CoA address in a foreign network in advance. The method can reduce the waiting time in the process of assigning IP address to the network membership. The network topology in this scenario is still the same in scenario 2.

5.5.2 Scenario4: Simulation Results and Evaluation

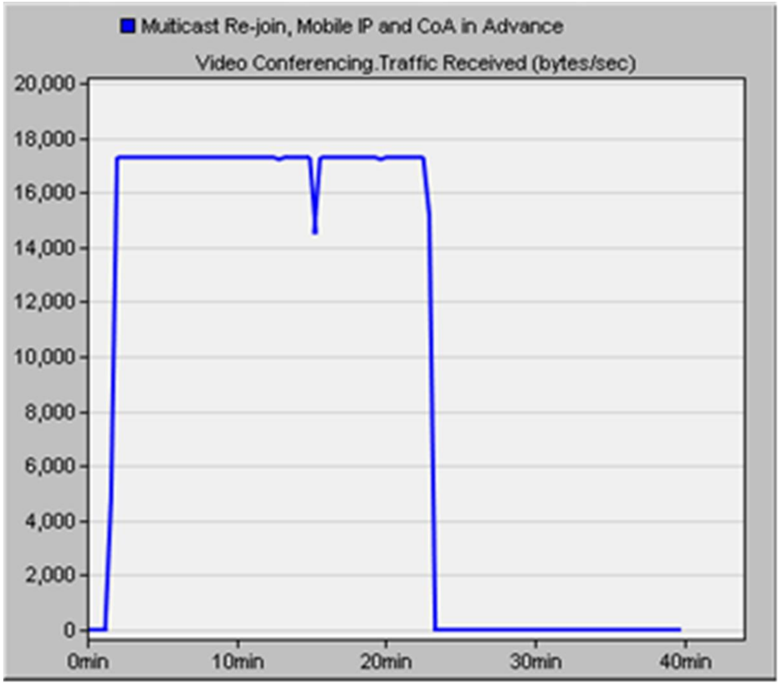

Figure 5-11 Scenario 4: Traffic received at Mobile_Node_A

The result is shown that when the foreign agent knows the new member node in advance, it helps to reduce the handover latency time on the network. Also, it increases the performance and throughput on the wireless network. The foreign router will establish the connection earlier, including the process of joining and distributing the multicast tree in advance. Furthermore, the method of binding update will happen before the handover occurs.

5.6 Scenario 5: Same Multicast Group

5.6.1 Scenario5: Scenario Description

The purpose of the current scenario was to determine the performance of the network when the foreign network is already a member of the multicast group.

5.6.2 *Scenario5: Network Topology*

The network topology of this scenario is slightly different from the previous network. In this infrastructure, the foreign network already has a mobile node, which is a member of the multicast group that Mobile_Node_A wants to join. In Figure 5-12, Mobile_Node_B is already a member of the multicast group from the media server.

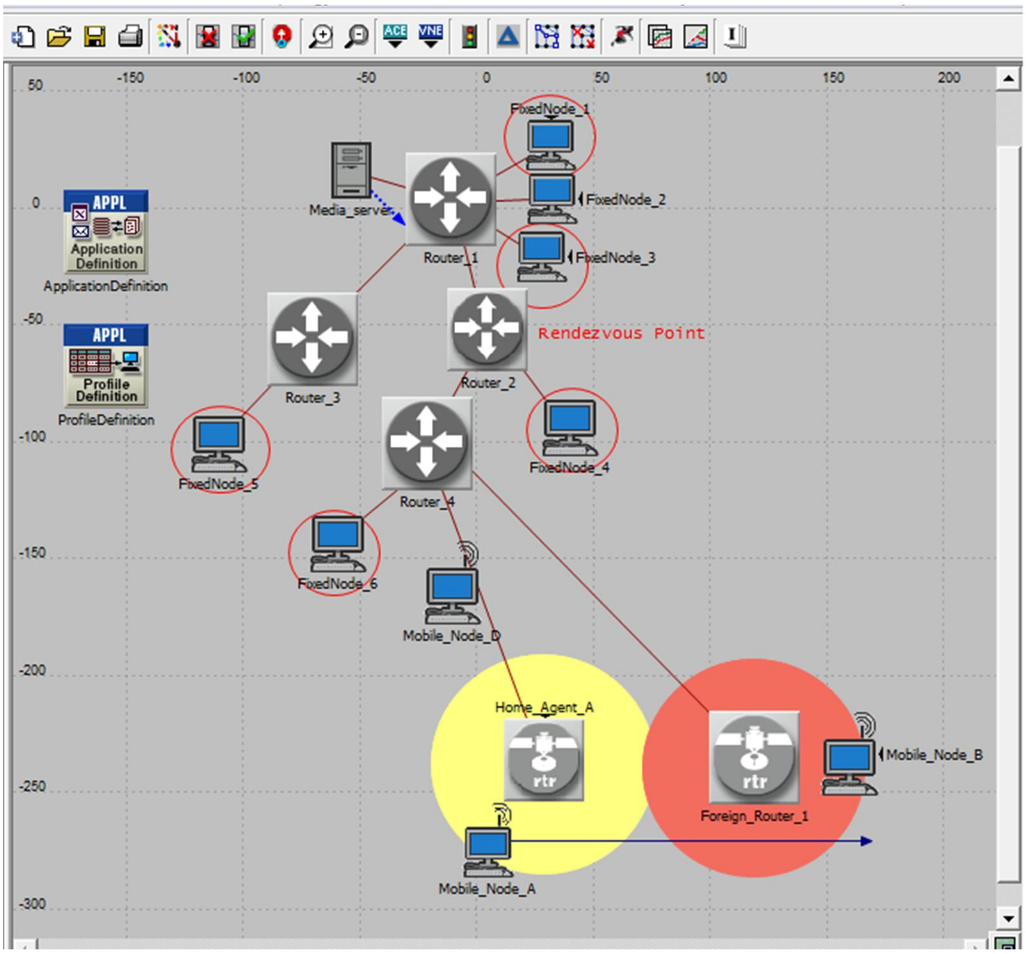

Figure 5-12 Scenario 5: Network Topology

5.6.3 Scenario5: Simulation Results and Evaluation

The graph below illustrates that if the foreign network is already a member of the multicast group the handover latency time will be reduced. This is because the new mobile node just deals with IP addressing and then picks up the multicast signal within the network. There are not any methods for multicast communication.

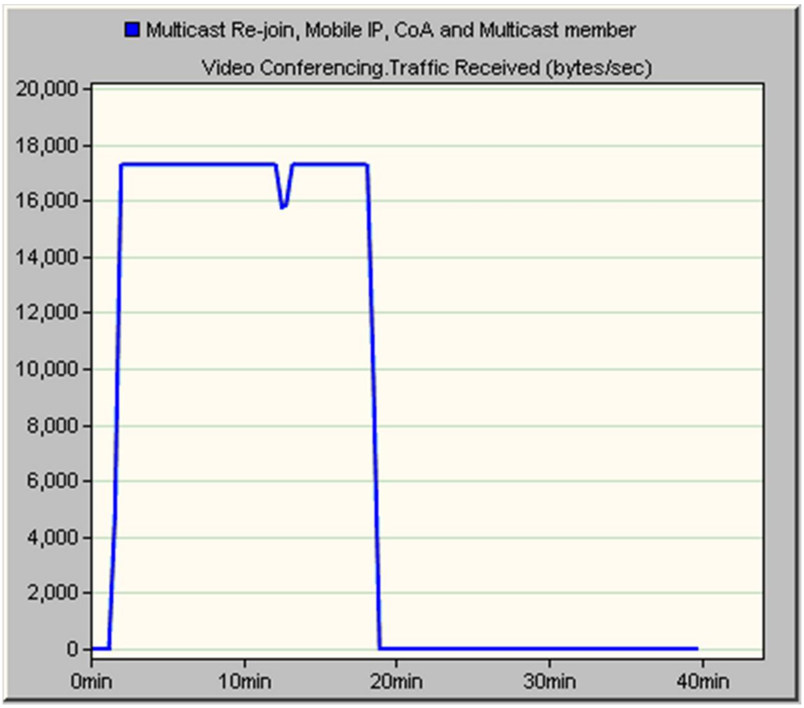

Figure 5-13 Scenario 5: Traffic received at Mobile_Node_A

5.7 Scenario 6: Multi-Hops

5.7.1 Scenario 6: Scenario Description

In this scenario, we increase a distance between home and foreign network. This study set out to determine the effects of multi-hops on the handover latency and packet delay.

5.7.2 Scenario 6: Network Topology

In the new topology, the foreign agent connects to Router_2. However, they are still connecting to the same multicast group.

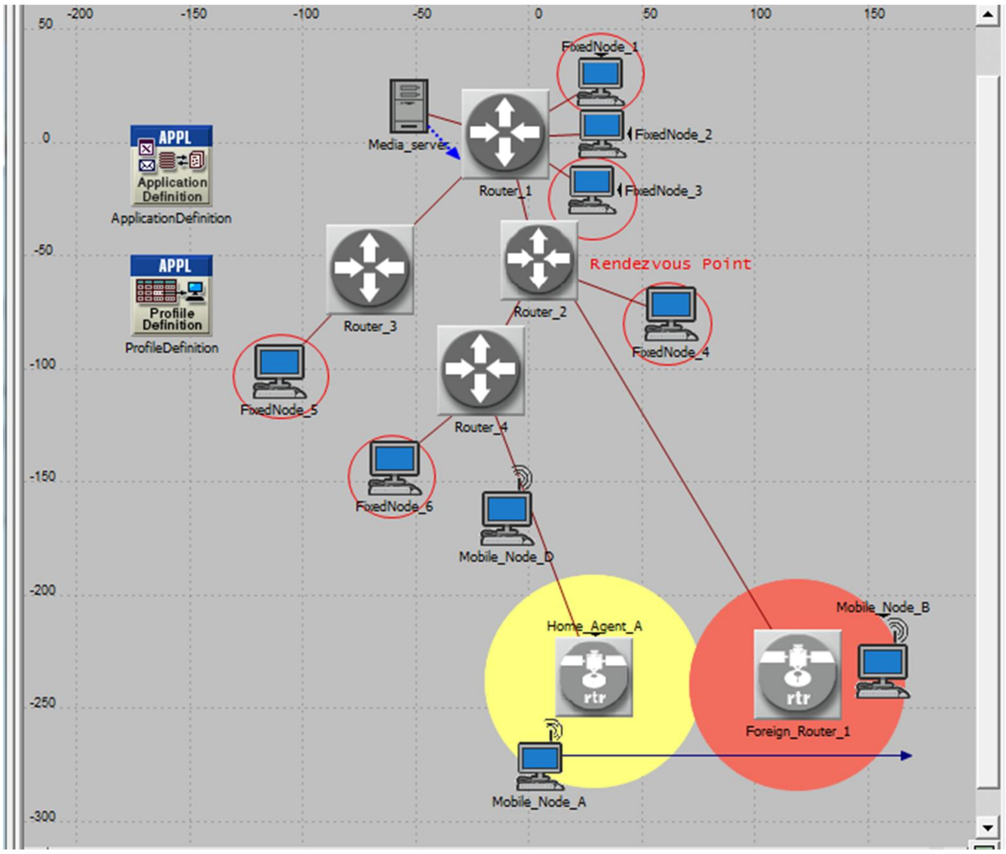

Figure 5-14 Scenario 6: Network Topology

5.7.3 Scenario 6: Simulation Results and Evaluation

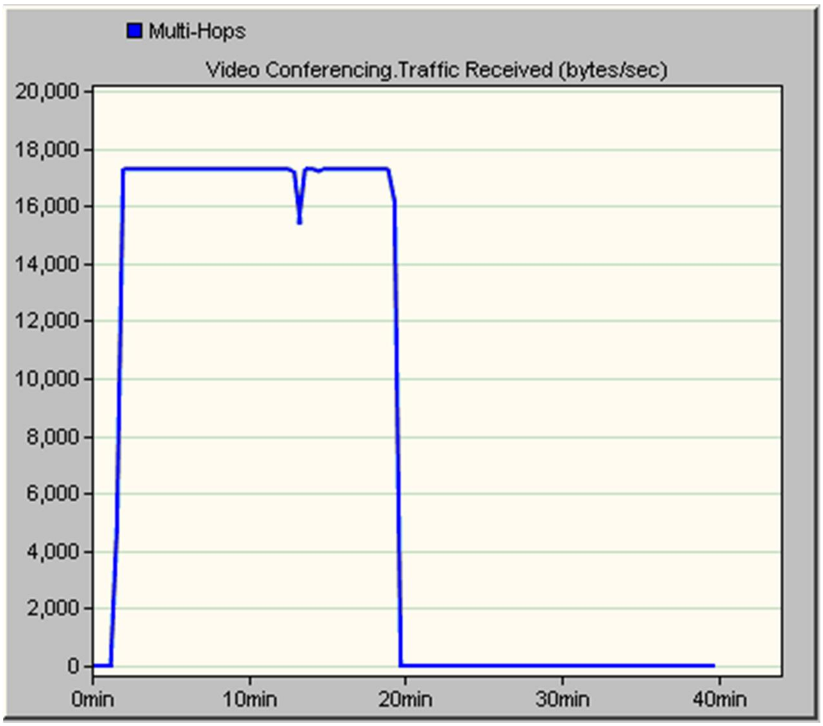

Figure 5-15 Scenario 6: Traffic received at Mobile_Node_A

The output result on Figure 5-15 is shown that the handover latency time slightly increases because the route path between both networks had changed. However, in this topology the foreign router connected to Router_2, which is one of the members of the multicast tree. Moreover, in this network, Router_2 is a Rendezvous point of the multicast tree. That is why the output result is only slightly different.

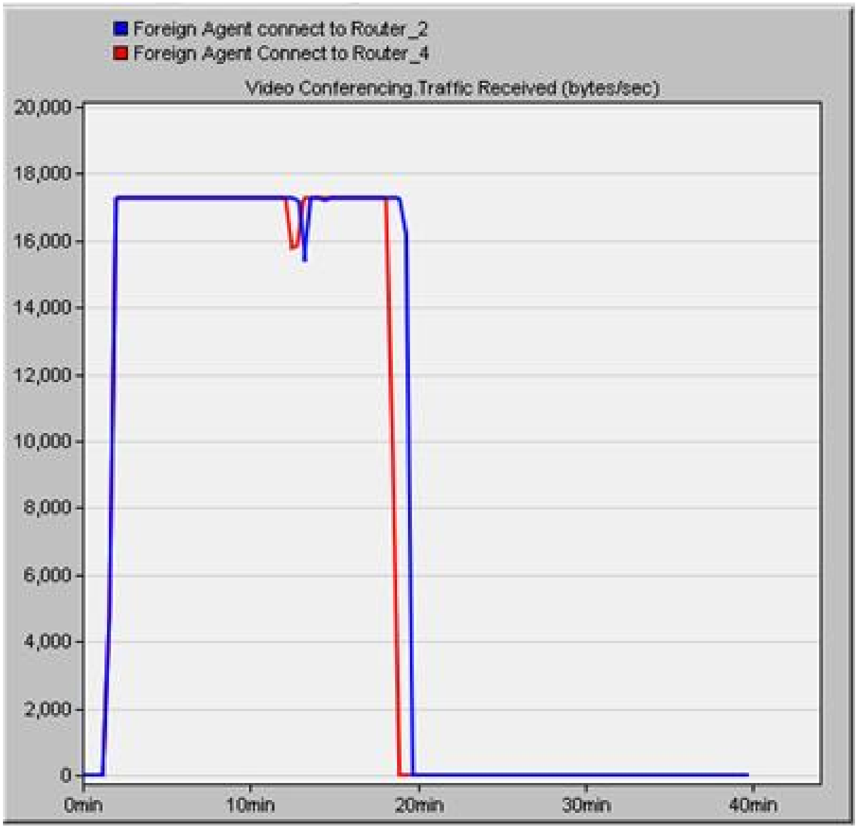

Figure 5-16 Scenario 6: Combined traffic received at Mobile_Node_A

Figure 5-16 combines the graph in Figure 5-13 and 5-15 in order to compare the results. The aim of this graph is to examine how hop distance affects handover latency in our designed framework. It can be seen from the data in the graph that the handover latency time slightly increases in this network topology.

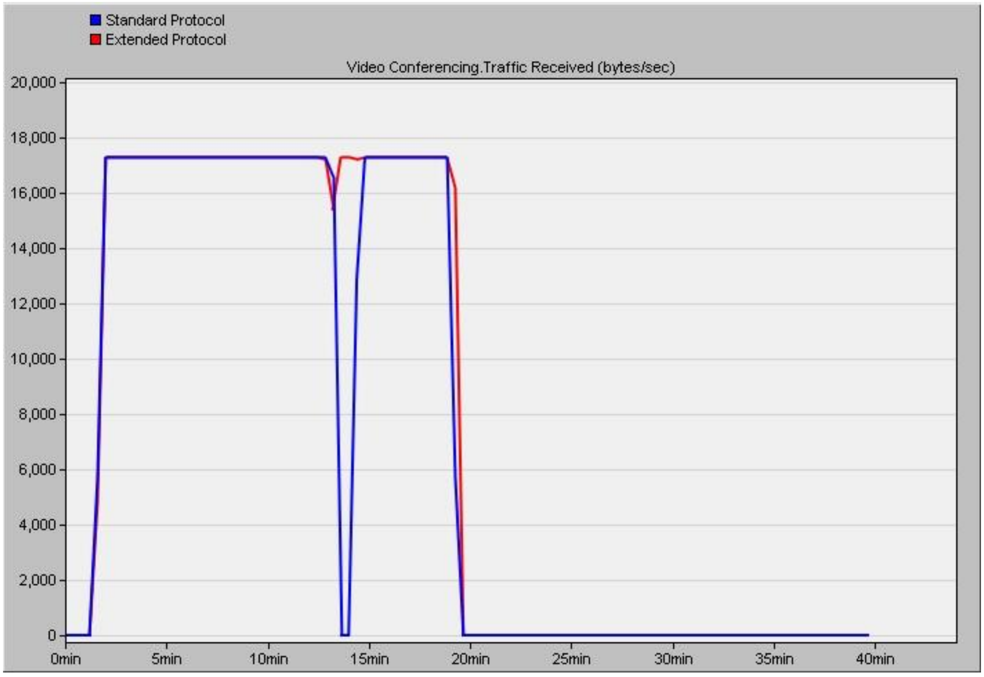

Figure 5-17 Is shown how much the extended protocols can reduce handover latency compared with standard protocols

The graph above illustrates the comparison between the graphs in Figure 5-15, which is the result from the modified framework, compared to the result from the standard framework. In the graph the handover latency of extended protocols is 48 seconds and 120 seconds for standard protocols. What is interesting in this data is that in a standard framework, mobile node completely disconnects until the process of IP readdressing and joining multicast are finished. The results of this study indicate that the modified framework can reduce the handover latency 60% when compared with the standard network.

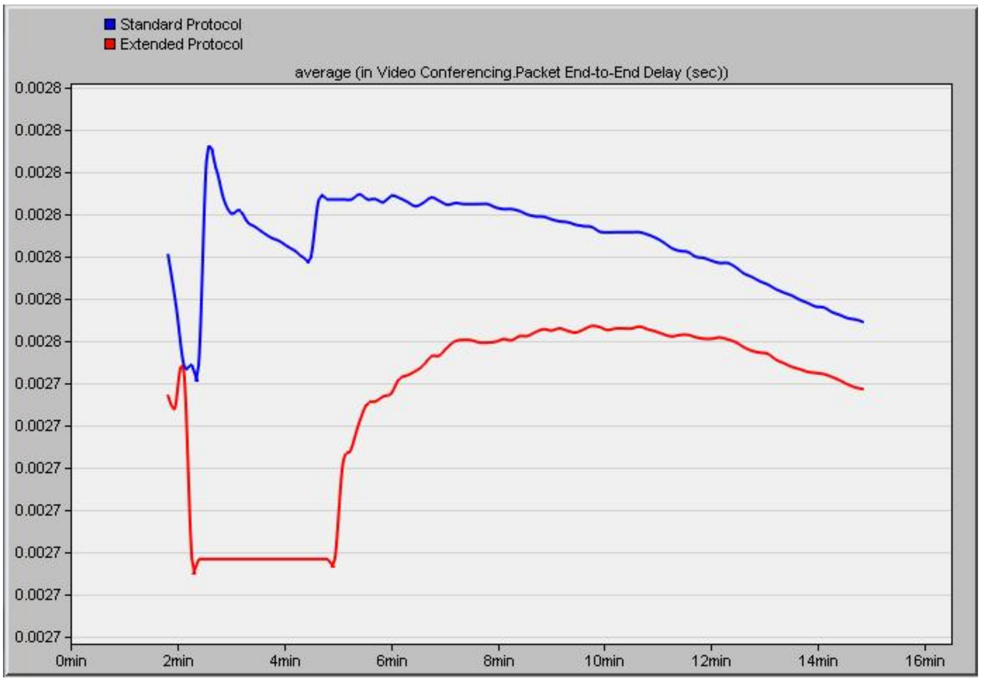

Figure 5-18 Is shown how much the extended protocols can reduce packet delay compared with standard protocols

From the network topology in Figure 5-14, the purpose of the current study was to assess packet delay in both networks. The above Figure is shown comparison of the output result of packet end-to-end delay in terms of average. It is apparent from this graph that the packet delays in the modified framework are lower than the original framework. On average the modified framework can reduce packet delay by approximately 3.5 – 10 ms throughout the simulation.

5.8 Scenario 7: Handover

5.8.1 Scenario 7: Scenario Description

In this scenario, we increase a scale of network topology for testing performance of proposed framework when mobile node joins in many WiFi networks along the path. Also, this scenario has shown that the mobile node can discover and connect to the new network when the mobile node moves.

5.8.2 Scenario 7: Network Topology

In this topology when the simulation starts, the Mobile_Node_A is a member of Home_Agent_A and receives multicast data from Media_server. Then Mobile_Node_A starts to move in the direction of the blue arrow from Foreign_Router_1 throughout Foreign_Router_4, as presented in Figure 5-19.

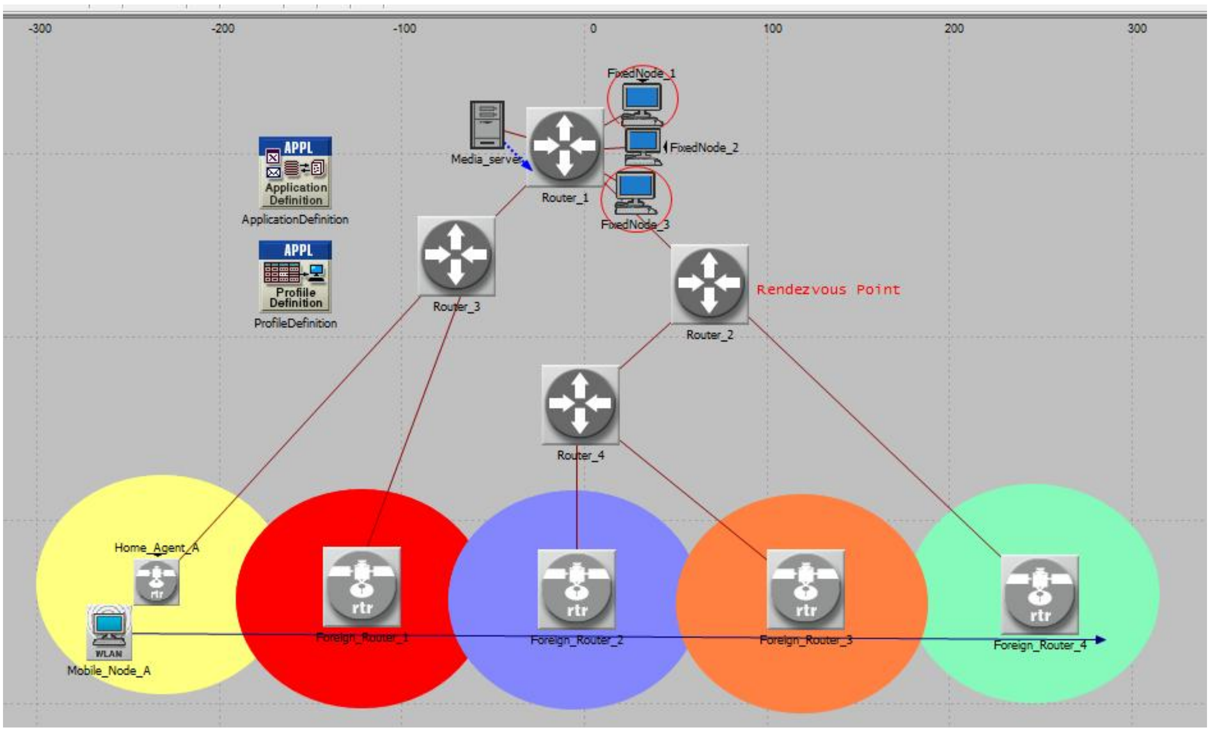

Figure 5-19 Scenario 7: Network Topology

5.8.3 Scenario 7: Simulation Results and Evaluation

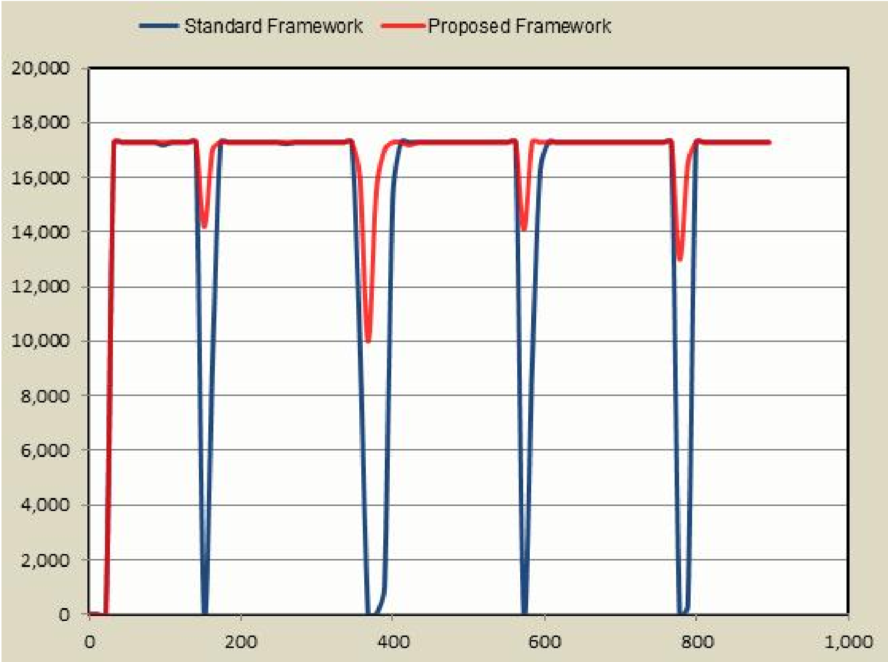

Figure 5-20 Scenario 7: Traffic received at Mobile_Node_A

As Figure 5-20 is shown, there is a significant difference between the two lines. The x-axes show the time in units of seconds. The y-axes represent traffic received in units of bytes/sec. From the graph above we can see that the proposed framework can significantly minimize handover latency more than the standard framework, especially when handover occurred between multi hops such as from Foreign_Router_1 to Foreign_Router_2 in Figure 5-19. In that time, the standard framework completely disconnects and has to restart every process from the beginning while Mobile_Node_A in the proposed framework can receive multicast traffic around 10,000 bytes/sec.

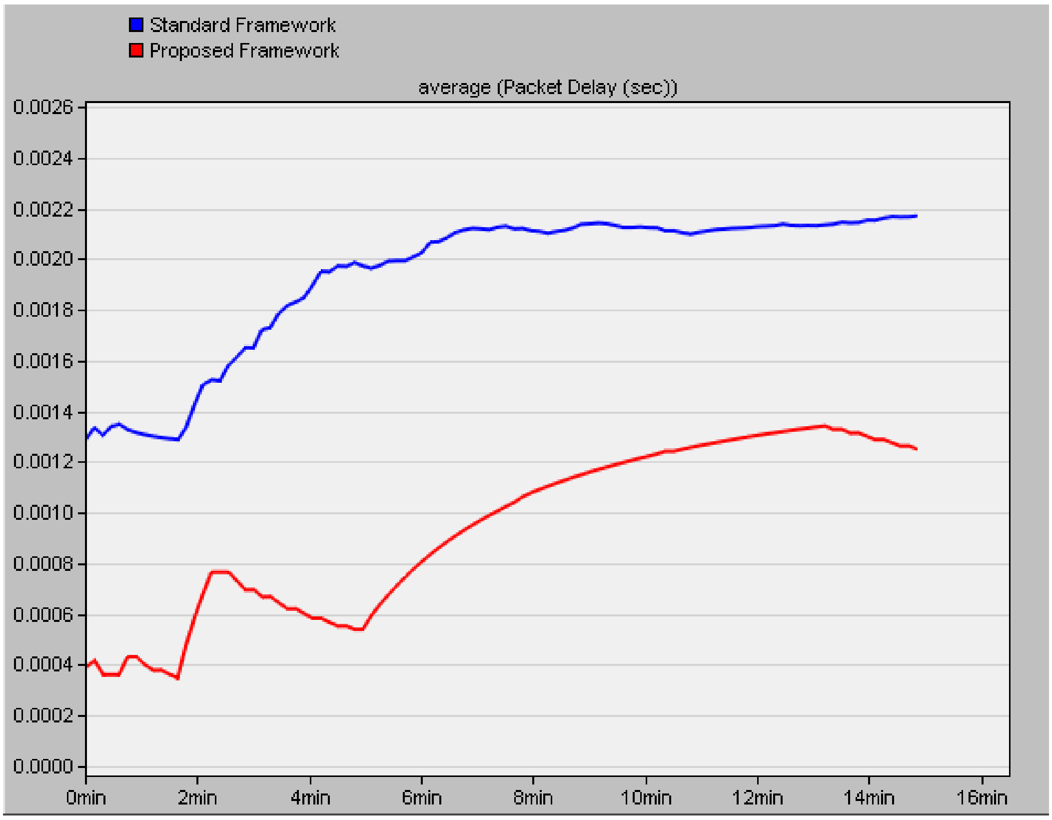

Figure 5-21 Scenario 7: Packet delay

Figure 5-21 represents the end-to-end delay of all the data packets that are successfully received by the mobile node. The graph is shown value in terms of average. There is a clear trend of lower packet delay in the design framework throughout the simulation. Further analysis showed that the designed framework reduced packet delay by approximately 0.7-1.5 ms in this scenario.

5.9 Scenario 8: Multiple Networks

5.9.1 Scenario 8: Scenario Description

In this scenario, we increase a group of WiFi zone to evaluate the performance of the proposed framework in the process of registering CoA address in the foreign network in advance. In this topology Mobile_Node_A will register the CoA address to all of the foreign agents in advance as soon as the mobile node moves into their coverage area. Moreover, this scenario will study how mobile nodes handle all CoA addresses after they are received.

5.9.2 Scenario 8: Network Topology

In this topology when the simulation starts, the Mobile_Node_A is a member of Home_Agent_A and receives multicast data from Media_server. Then Mobile_Node_A starts to move with constant speed in the direction of the blue arrow from Home_Agent_A to Foreign_Router_2 and then moves to Foreign_Router_4, as presented in Figure 5-22. There are 7 WiFi zones in this scenario.

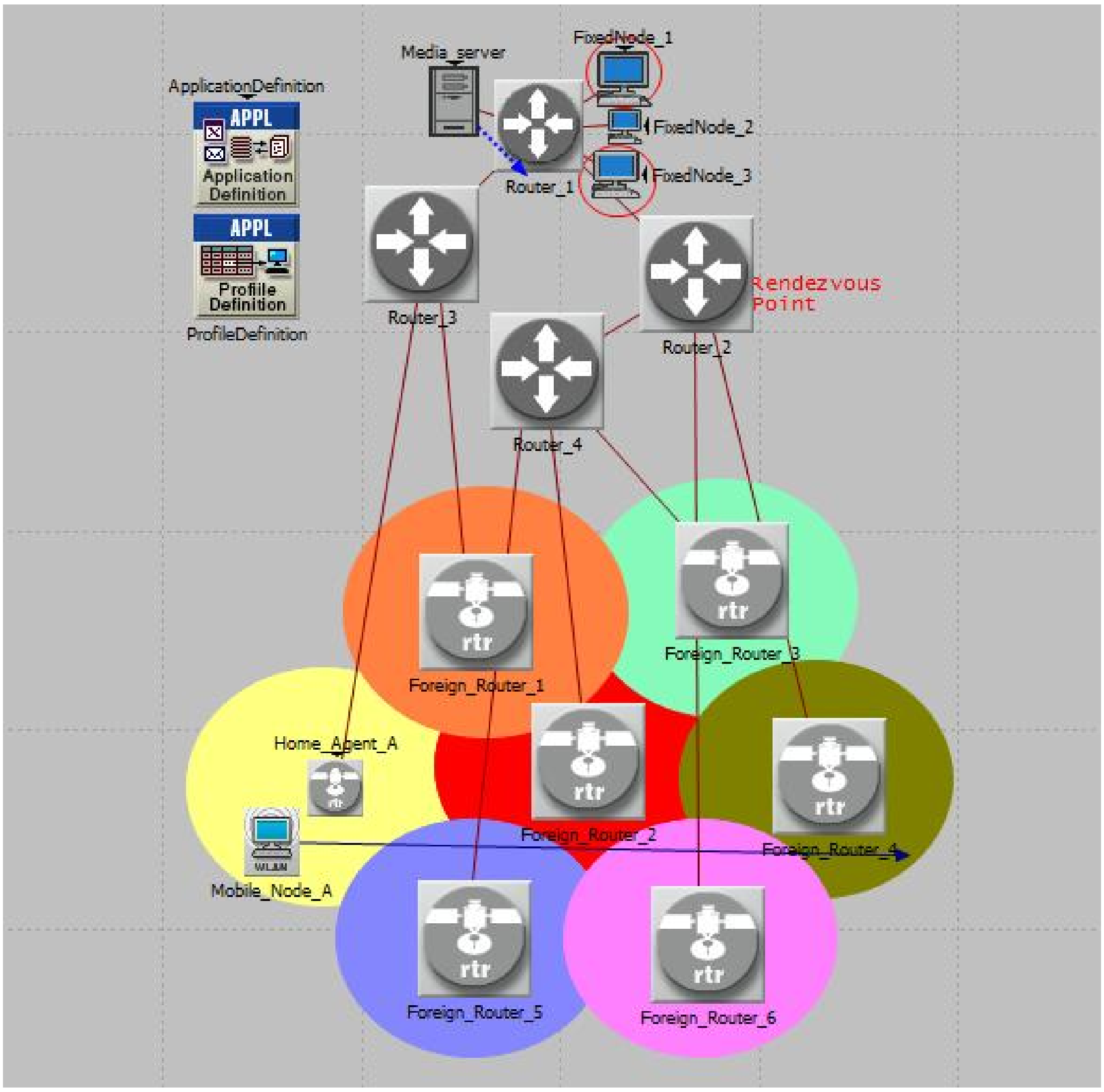

Figure 5-22 Scenario 8: Network Topology

5.9.3 Scenario 8: Simulation Results and Evaluation

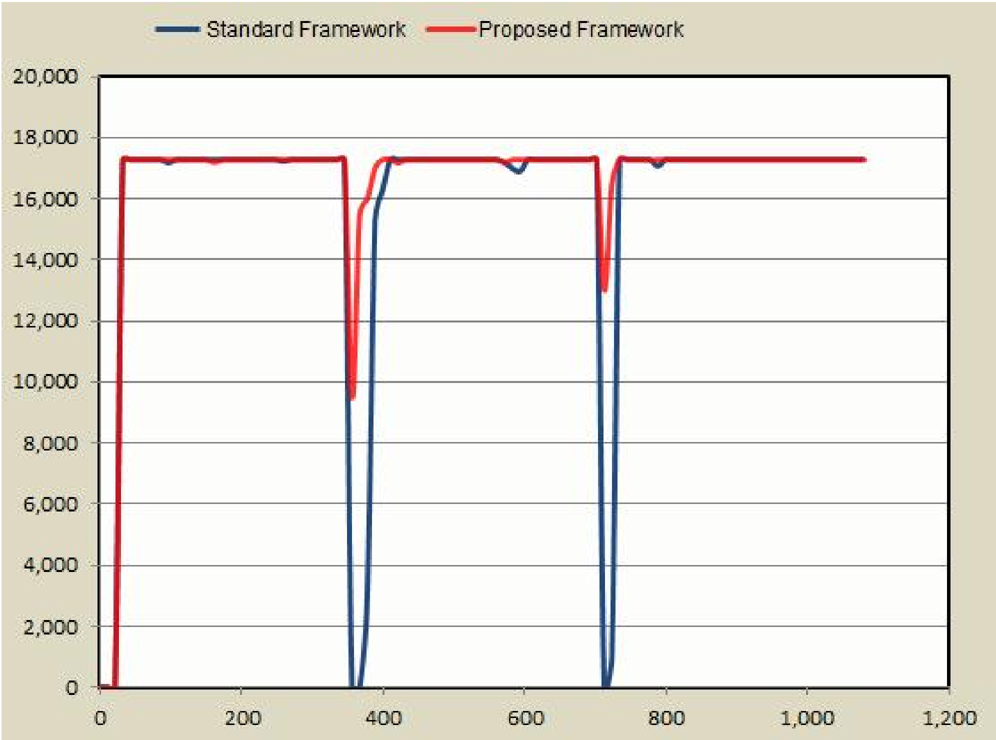

Figure 5-23 Scenario 8: Traffic received at Mobile_Node_A

The results, as shown in Figure 5-23, indicate that the proposed framework can reduce handover latency for the mobile node. Also, on the designed framework the mobile node continuously received multicast data and did not lose the connection as the standard framework does. In addition, this scenario confirms that the mobile node can store those CoA addresses from foreign agents and use them to connect to the multicast tree in advance. This method will help foreign routers establish the connection and distribute the multicast tree earlier.

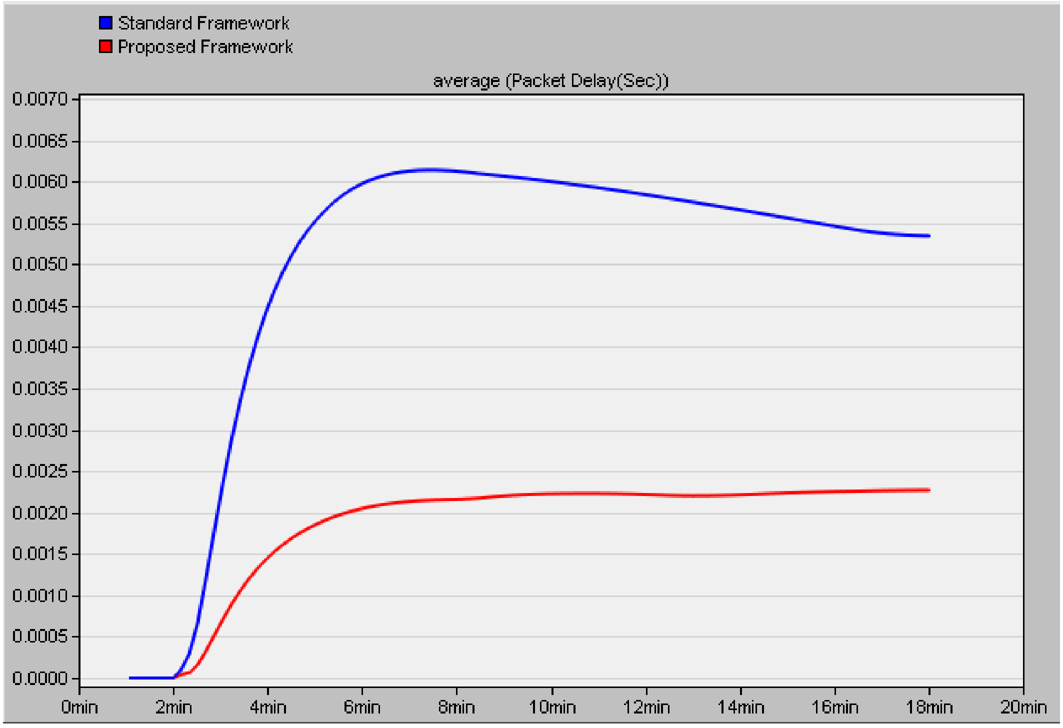

Figure 5-24 Scenario 8: Packet delay

The above figure represents the end-to-end delay on average at the mobile node. Further analysis showed that the designed framework reduced packet delay by approximately 1.5-40 ms in this scenario.

5.10 Scenario 9: Complex Networks

5.10.1 Scenario 9: Scenario Description

To increase the reliability of the proposed framework, we created more groups of WiFi zone to evaluate the performance. Also, in order to study the effectiveness of registering the CoA address in an advance process, the mobile node should register only the foreign network that the mobile node is in the coverage area of. Hence, in this scenario the direction of the mobile node was changed during simulation.

5.10.2 Scenario 9: Network Topology

In this topology when the simulation starts, the Mobile_Node_A is a member of Home_Agent_A and receives multicast data from Media_server. Then Mobile_Node_A starts to move with constant speed in the direction of the blue arrow from Home_Agent_A to Foreign_Router_5 and then moves to Foreign_Router_6, until it stops at Foreign_Router_8 at the end of the simulation, as presented in Figure 5-25.

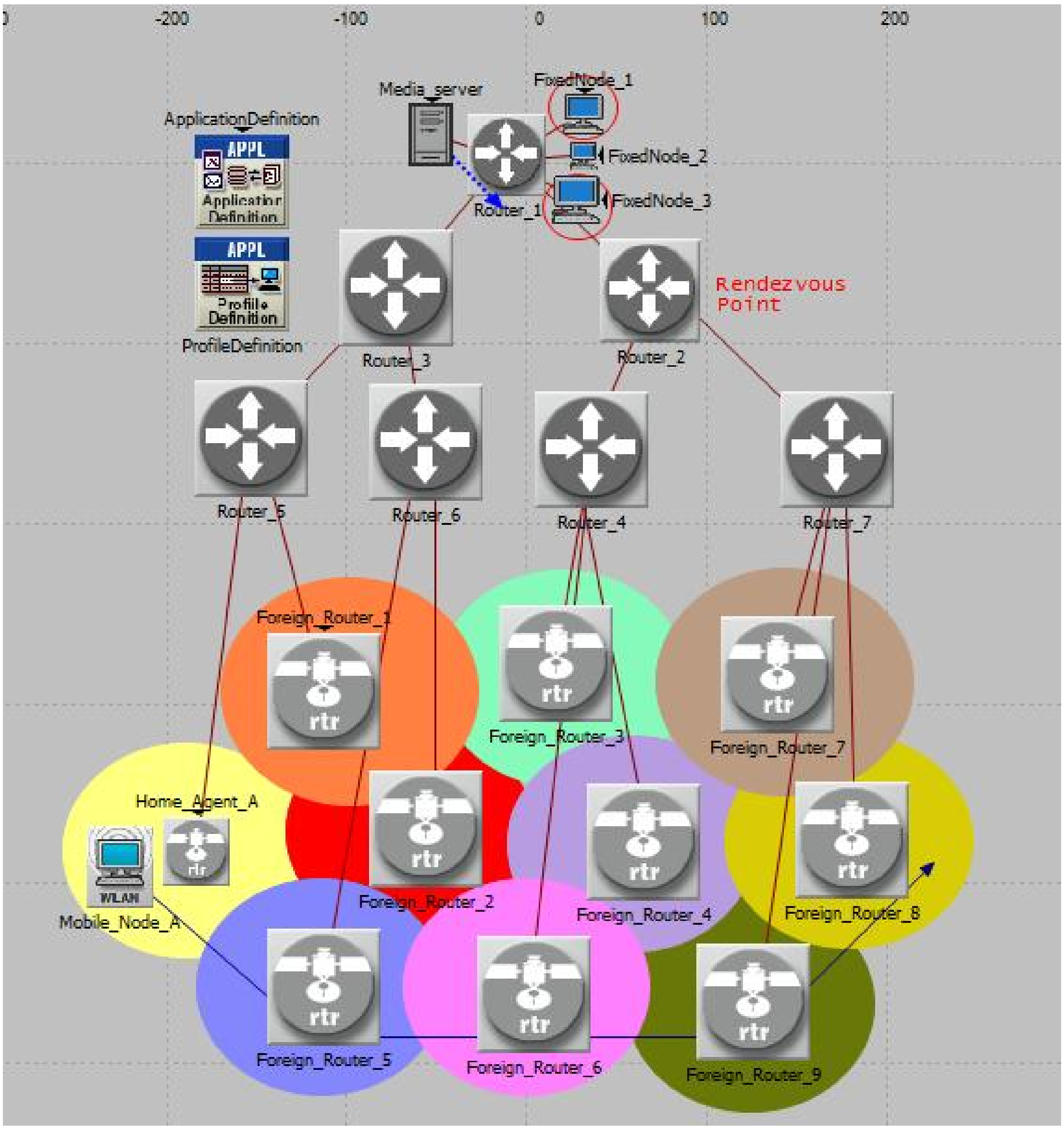

Figure 5-25 Scenario 9: Network Topology

5.10.3 Scenario 9: Simulation Results and Evaluation

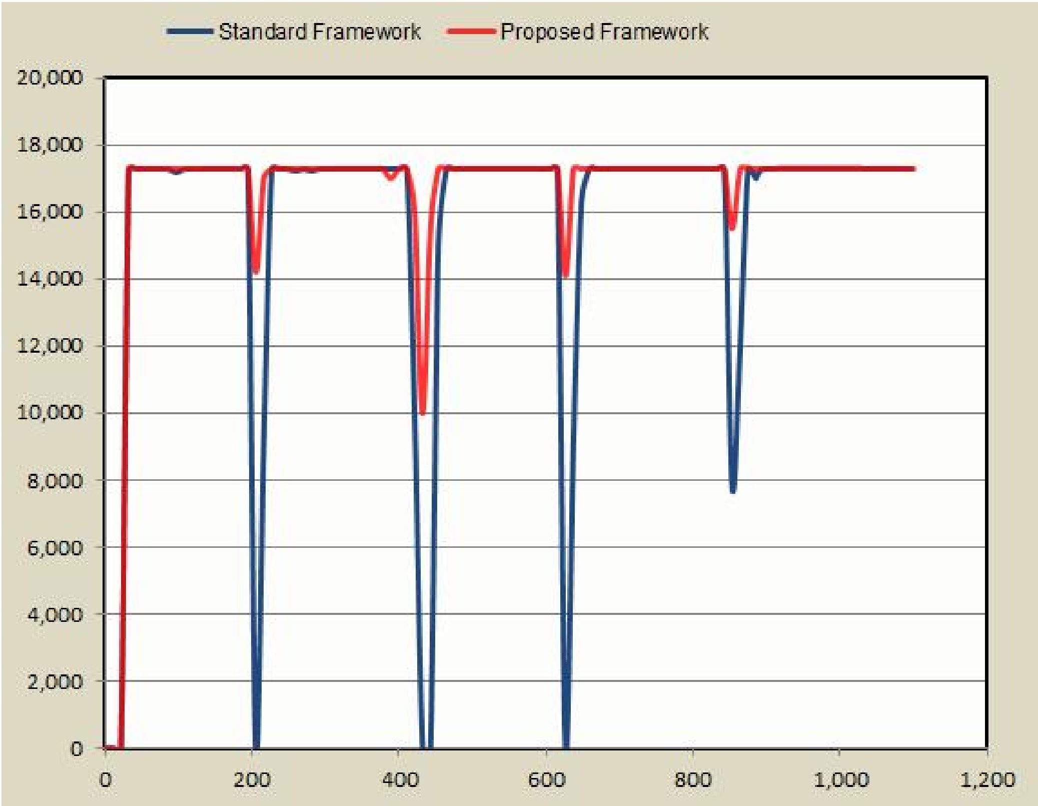

Figure 5-26 Scenario 9: Traffic received at Mobile_Node_A

The findings of the current network topology are consistent with the previous scenario that the proposed framework can minimize handover latency time. However, the performance of this framework depends on the structure and topology of the network. It can show that the framework can reduce handover latency time that happen in network and transport layer. But it is still create handover latency time that happen in data link layer.

5.11 Scenario 10: Internet

5.11.1 Scenario 10: Scenario Description

To increase the reliability of the proposed framework, we have applied our framework to the internet. In this scenario, the network will be similar to scenario 9, however, Media server and all access points connect to each other and the multicast tree via the internet.

5.11.2 Scenario 10: Network Topology

The network topology in this scenario is similar to that of scenario 9. The mobile node moves in the same direction. However, in this scenario the mobile node received multicast data from Media_server via the internet. In Figure 5-27, the IP cloud represents the internet. In the simulation we have created some traffic such as email, www, and ftp traffic etc. within the internet as random traffic to make it more like reality. Moreover, the aim of this scenario is to evaluate the performance of the designed framework when it is a part of the internet.

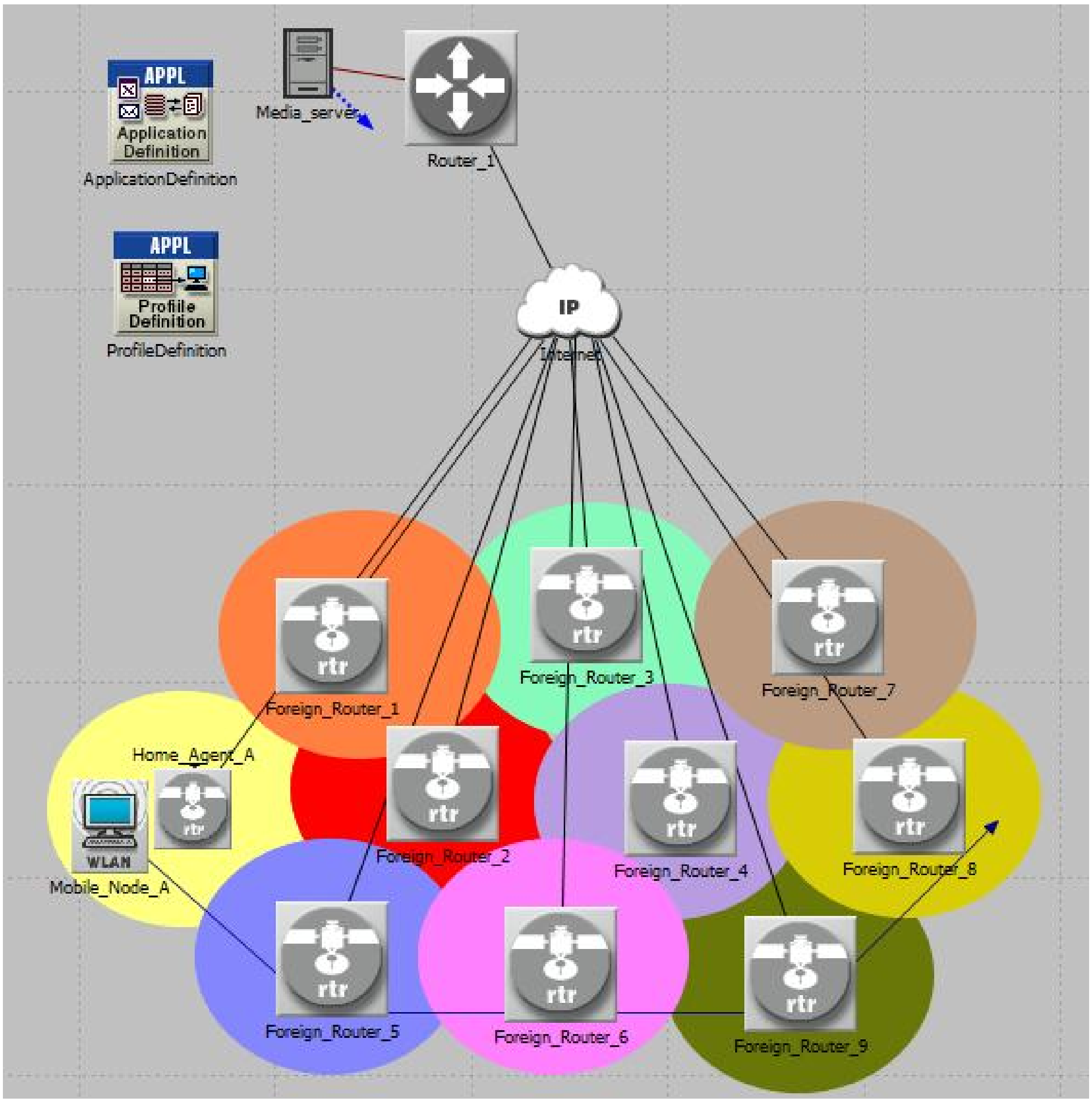

Figure 5-27 Scenario 10: Network Topology

5.11.3 Scenario 10: Simulation Results and Evaluation

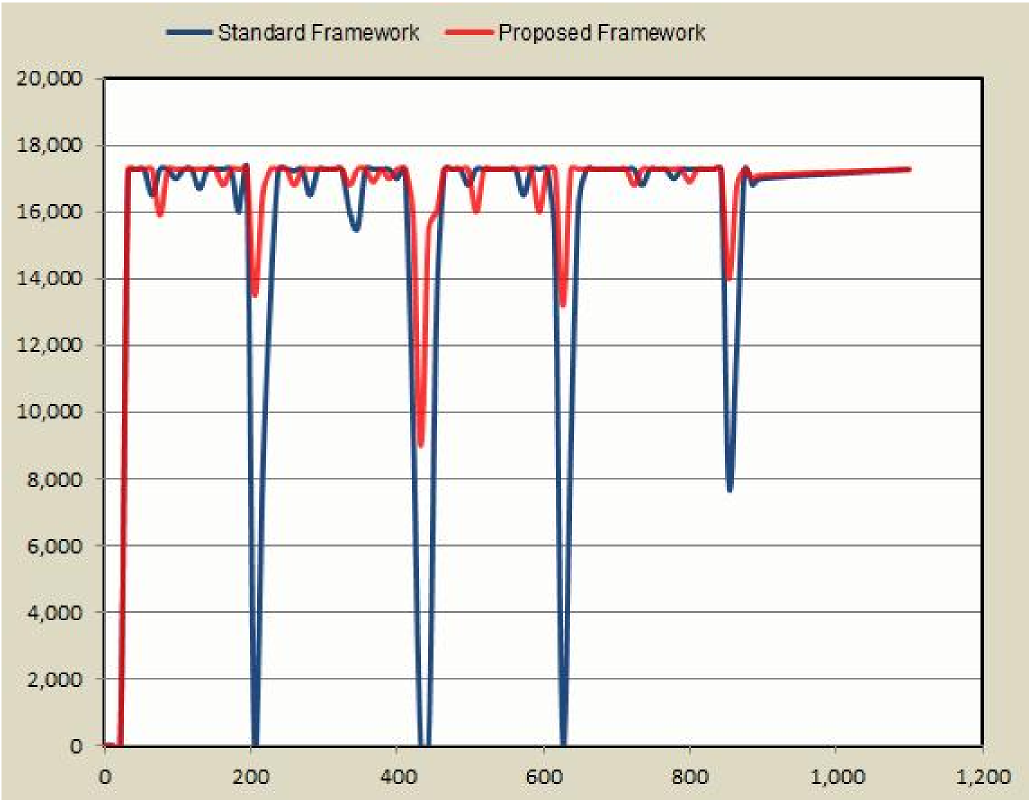

Figure 5-28 Scenario 10: Traffic received at Mobile_Node_A

The current result found that the traffic on the internet slightly affects handover latency time. However, the traffic on the internet directly affects multicast data received at the mobile node because there is traffic on the internet. The most interesting finding was that the proposed framework could be applied for using on the internet and it did not affect the performance.

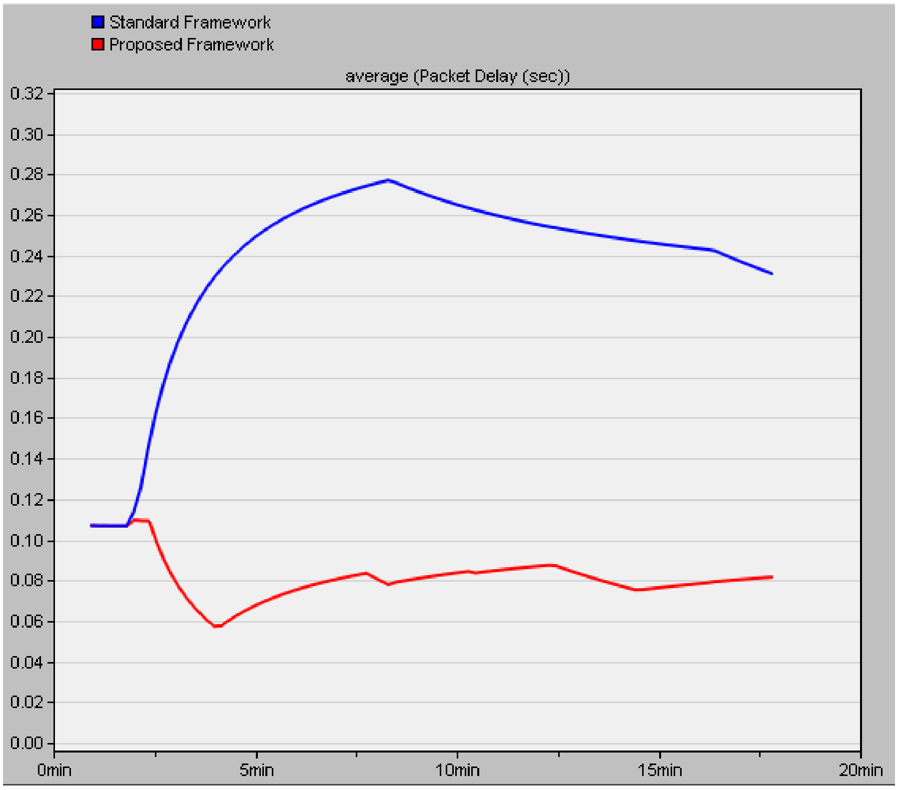

Figure 5-29 Scenario 10: Packet Delay

The above figure represents the end-to-end Packet delay of video conferencing on average at mobile node. The packet delay increased significantly when connected through the internet. Further analysis showed that the designed framework reduced packet delay approximately 80 - 150 ms in this scenario.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In this thesis, a framework for supporting multicast mobility has been designed. The concept of this framework can support WiFi network both on IPv4 and IPv6 networks. The contribution of this thesis is to find the way to reduce handover latency time, which means including handover delay packet loss and jitter.

However, from the output result on OPNET Medeler network simulation software in chapter 5, it can be confirmed that the designed framework of this research achieves the aim of the research in this thesis.

Those methods and processes within the designed framework are a key factor that can produce an output result for achieving the research aim that can be analysed and classified into:

- Foreign agent arranges an IP address for a new mobile node before it becomes a member within the network. This way it helps the foreign agent to reduce the time for searching an available IP address on their database, including negotiation time between foreign router and new mobile node. It is affected from registering CoA in advance module.
- As it knows a new mobile node member in advance the foreign router can establish a path to multicast group early.
- The results of this research support the idea that when mobile node connected to the multicast tree early, the process of rebuild multicast tree can happen early also. This saves time for connecting to multicast tree.
- Helps Mobile IP protocol to do the process of Binding update to home agent early.
- The results of this investigation show that mobile node can use the new IP address to join multicast tree in advance due to already having the new IP address.

- Reduces time in the process of becoming a member in a new multicast group.

6.2 Recommendation for Future Work

From this research, there are many things that can be improved, modified and developed for making this framework more powerful, such as:

- Include the algorithm or process for predicting potential foreign agent in advance. This is to save the resource and bandwidth on the wireless network. Further work is required to establish this.
- Design a new algorithm or method for finding higher performance of the foreign agent.
- Find the way to store CoA address, in case of receiving lots of CoA in advance.
- Improve the performance of framework in terms of robustness such as when there are lots of mobile nodes within the wireless network.
- Increase the performance of framework in terms of scalability for supporting more mobile nodes. This is an important issue for future research.
- Improve framework for supporting large scale multicast tree.
- Improve framework for supporting a variety of applications and multimedia sizes.
- Extend framework for supporting many-to-many delivery applications. Future studies on this topic are therefore recommended.

Appendices

Appendix A: Essential Kernel Procedures

This appendix presents the most-used kernel procedures (KPs) and functions in OPNET Modeler. They are grouped by the following areas of functionality:

- Attribute Access
- Distributions
- Dynamic Processes
- Events and Time
- Identification and Discovery
- Interface Control Information (ICIs)
- Interrupt Processing
- Packet Generation and Processing
- Statistic Recording

Attribute Access Get or set attribute values. (Simulation attributes are "global" to the simulation model.)

• *op_ima_obj_attr_get_<type>* –> completion code *<type>* = *color*, *dbl*, *int32*, *objid*, *str*, *toggle*

• *op_ima_obj_attr_set_<type>* –> completion code *<type>* = *color*, *dbl*, *int32*, *objid*, *str*, *toggle*

• *op_ima_sim_attr_get_<type>* –> completion code *<type>* = *color*, *dbl*, *int32*, *str*

Distributions Load distributions by name; Obtain outcomes from loaded distributions.

• *op_dist_load (dist_name, dist_arg0, dist_arg1)* –> distribution handle

• *op_dist_outcome (dist_ptr)* –> outcome

• *op_dist_uniform (limit)* –> outcome (between 0.0 and limit)

Dynamic Processes Create a new “child” process of a given type; Destroy a process.

• *op_pro_create (model_name, ptc_mem_ptr)* –> process handle

• *op_pro_destroy_options (pro_handle, options)* –> completion code Identify the current process.

• *op_pro_self ()* –> handle for this process Invoke another process (cause it to execute now). As an invoked process, get optional state that is passed.

• *op_pro_invoke (pro_handle, argmem_ptr)* –> completion code

• *op_pro_argmem_access()* –> argument pointer

Events and Time

Cancel an event.

• *op_ev_cancel (evhandle)* –> completion code

Obtain current simulation time.

• *op_sim_time ()* –> current simulation time in seconds

Terminate simulation.

• *op_sim_end (line0, line1, line2, line3)* –> (no return value)

Identification and Discovery

Find the containing object.

• *op_id_self ()* –> object ID of containing object

Find the parent of an object.

• *op_topo_parent (child_objid)* –> object ID of parent

Find an object’s descendants in the hierarchy.

• *op_topo_child_count (parent_objid, child_type)* –> number of children of specified type

• *op_topo_child (parent_objid, child_type, child_index)* –> object ID of the i’th child meeting criteria

Find an object’s peers. “objmtype” is one of an enumerated set; “direction” is IN

or OUT. Possible use: how many links am I connected to; then, give me the i’th

link.

• *op_topo_assoc_count (objid, direction, objmtype)* –> number of associations of given direction and type

• *op_topo_assoc (objid, direction, objmtype, index)* –> object ID of the i’th association meeting the direction and type criteria

Interface Control Information (ICIs)

Create or destroy an ICI.

• *op_ici_create (fmt_name)* –> new ICI

• *op_ici_destroy (iciptr)* –> (no return value)

Get or set ICI attribute values.

• *op_ici_attr_get_<type>*, <type> = *dbl*, *int32*, *int64*, *ptr* –> completion code

• *op_ici_attr_set_<type>*, <type> = *dbl*, *int32*, *int64*, *ptr* –> completion code

Associate an ICI with a particular interrupt.

• *op_ici_install (iciptr)* –> previously installed ICI

Interrupt Processing Schedule an interrupt for this object or another at a given time. Optionally pass a “code”.

• *op_intrpt_schedule_self (time, code)* –> event handle for interrupt

• *op_intrpt_schedule_remote (time, code, mod_objid)* –> event handle for interrupt

Obtain various attributes of the current interrupt.

• *op_intrpt_type ()* –> type (such as packet arrival, statistic change, self interrupt)

• *op_intrpt_strm ()* –> stream for packet arrivals

• *op_intrpt_ici ()* –> control information passed with an interrupt (arbitrary structure)

Packet Generation and Processing

Create, copy, or destroy a packet.

• *op_pk_copy (pkptr)* –> pointer to new copy of packet

• *op_pk_destroy (pkptr)* –> (no return value)

Get or send a packet. (with optional delay)

• *op_pk_send_delayed (pkptr, outstrm_index, delay)* –> (no return value)

Get and set named fields of a packet.

• *op_pk_nfd_set_<type>*, <type> = *dbl*, *info*, *int32*, *int64*, *objid*, *pkid*, *pkt*, *ptr* –> completion code

• *op_pk_nfd_get_<type>*, <type> = *dbl*, *int32*, *int64*, *objid*, *pkid*, *pkt*, *ptr* –> completion code

Get certain properties of a packet.

• *op_pk_creation_time_get (pkptr)* –> simulation time at which packet was created

• *op_pk_total_size_get (pkptr)* –> size of packet in bits (sum of field sizes) Insert or remove a packet from a specified subqueue.

• *op_subq_pk_remove (subq_index, pos_index)* –> pointer to packet removed from the specified subqueue

Statistic Recording Obtain a handle for a statistic, given its name. Type is Global or Local.

Optionally specify an index when a single statistic name encompasses multiple independent time series.

• *op_stat_reg (stat_name, stat_index, type)* –> statistic handle

Write a new value to a particular statistic. (new value is assumed to be recorded at the current time)

• *op_stat_write (stat_handle, value)* –> (no return value)

Appendix B: C++ Code

This appendix presents the source code of the research that had been developed in OPNET Modeler simulation software.

```
/* This variable carries the header into the object file */
const char mobile_ip_mn_pr_c [] = "MIL_3_Tfile_Hdr_ 30A op_runsim_dev 7
4F427647 4F427647 1 Khanista-PC Khanista 0 0 none none 0 0 none 0 0 0 0 0 0 0
0 4871 6
";
#include <string.h>

/* OPNET system definitions */
#include <opnet.h>

/* Header Block */

#include    "ip_rte_support.h"
#include    "mobile_ip_support.h"
#include    "ip_addr_v4.h"
#include    "ip_mcast_support.h"
#include    "mobility_support.h"
#include    "ip_igmp_support.h"
#include    "ip_pim_sm_support.h"
#include    "udp_api.h"
#include    "ip_icmp_pk.h"
#include    "ip_dgram_sup.h"
#include    "math.h"

#define     IP_PK                   is_ip_pk
#define     REG_PK                  is_reg_pk
#define     AD_RECEPTION            is_ad_reception
#define     HA_AD_RECEPTION         is_ha_ad_reception
#define     FA_AD_RECEPTION         is_fa_ad_reception
#define     SOLICITATION_TIME       is_solicitation_time
#define     VALID_FA_CANDIDATE      is_valid_fa_candidate
#define     HA_TIMEOUT              is_ha_timeout
#define     FA_TIMEOUT              is_fa_timeout
#define     TIMEOUT                 is_timeout
#define     RETRY                   is_retry
#define     FA_REG_SUCCESS          is_fa_reg_success //Me
#define     HA_REG_SUCCESS          is_ha_reg_success //Me
#define     OUT_OF_RETRIES          is_out_of_retries
#define     INVALID_REPLY           is_invalid_reply
#define     REREGISTER              is_reregister
#define     SWITCH_FA               is_switch_fa

#define     TIMER                   is_timer //Me
#define     SOLICIT                 is_solicit //Me

#define     MipC_MN_Rereg_Buffer    1.0

#define     MipC_MN_Solicit_Max_Interval  60.0
#define     MipC_MN_Solicit_Min_Interval  1.0
```

```
#define     IP_DEFAULT_TTL                32
#define     IPC_PIM_SM_RPF_TIMER_OFFSET   50
#define     IPC_PIM_SM_NOT_BSR_CAND       0
#define     IPC_PIM_SM_TOS                0
#define     IPC_PIM_SM_DATA_RATE_TIMER    1
#define     IPC_PIM_SM_START              2
#define     IPC_PIM_SM_RPF_UPDATE         3
#define     IPC_PIM_SM_SEND_JOIN_PRUNE_MSG      4
#define     IPC_PIM_SM_DR_TIMEOUT_OFFSET        1000
#define     IPC_PIM_SM_SEND_HELLO_MSG_OFFSET    2000

/****** Transition Macros ******/
#define     HELLO_MSG                  (transition_code ==
IpC_Pim_Sm_Hello_Msg_Recvd)
#define JOIN_PRUNE_MSG                 (transition_code ==
IpC_Pim_Sm_Join_Prune_Msg_Recvd)
#define DATA_PKT                       (transition_code ==
IpC_Pim_Sm_Data_Pkt_Recvd)
#define RTE_PLUS                       (transition_code == IpC_Pim_Sm_Rte_Plus)
#define RTE_MINUS                      (transition_code == IpC_Pim_Sm_Rte_Minus)
#define REGISTER_MSG                   (transition_code ==
IpC_Pim_Sm_Register_Msg_Recvd)
#define REGISTER_STOP_MSG              (transition_code ==
IpC_Pim_Sm_Register_Stop_Msg_Recvd)
#define DATA_RATE_TIMER_EXPD  (transition_code ==
IpC_Pim_Sm_Data_Rate_Timer_Expd)
#define SEND_HELLO_MSG                 (transition_code ==
IpC_Pim_Sm_Send_Hello_Msg)
#define SEND_JOIN_PRUNE_MSG            (transition_code ==
IpC_Pim_Sm_Send_Join_Prune_Msg)
#define DR_TIMEOUT                     (transition_code ==
IpC_Pim_Sm_Dr_Timeout)
#define RPF_UPDATE                     (transition_code ==
IpC_Pim_Sm_RPF_Update)
#define FAILURE_RECOVERY               (transition_code ==
IpC_Pim_Sm_Failure_Recovery)

#define START_INTERRUPT                (invmode == OPC_PROINV_DIRECT)

#define DELAY_TIMER_EXPD               (transition_code ==
IpC_Igmp_Host_Delay_Timer_Expd)
#define REPORT_RECVD                   (transition_code ==
IpC_Igmp_Host_Report_Recvd)
#define QUERY_RECVD                    (transition_code ==
IpC_Igmp_Host_Query_Recvd)
#define LEAVE_GROUP                    (transition_code ==
IpC_Igmp_Host_Leave_Grp)
#define JOIN_GROUP                     (transition_code ==
IpC_Igmp_Host_Join_Grp)

#define ICMP_IP_PROCESS_INVOKE         -1
#define ECHO_REQUEST_GEN               (intrpt_type == OPC_INTRPT_SELF) &&
(pkt_from_ip == OPC_FALSE)
#define ECHO_REQUEST_RCVD      (icmp_message_type == IpC_Icmp_Echo_Request)
#define ECHO_REPLY_RCVD      (icmp_message_type == IpC_Icmp_Echo_Reply)
#define IP_ICMP_DEST_ADDR_UNSPECIFIED      "0.0.0.0"
```

```
#define IPC_ICMP_ECHO_PKSIZE_BITS         64

/****** Macro for Traces *******/
#define LTRACE_PIM_SM               (op_prg_odb_ltrace_active ("pim-sm") ||
op_prg_odb_trace_active ())
#define LTRACE_PIM_SM_ALL_BUT_DATA  (op_prg_odb_ltrace_active ("pim-
sm_all_but_data"))
#define LTRACE_PIM_SM_JOIN_PRUNE    (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_join_prune"))
#define LTRACE_PIM_SM_HELLO         (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_hello"))
#define LTRACE_PIM_SM_TIMERS        (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_timers"))
#define LTRACE_PIM_SM_DATA          (LTRACE_PIM_SM ||
op_prg_odb_ltrace_active ("pim-sm_data"))
#define LTRACE_PIM_SM_FAIL_RECOVER  (LTRACE_PIM_SM ||
op_prg_odb_ltrace_active ("pim-sm_fail_recover"))
#define LTRACE_IGMP           (op_prg_odb_ltrace_active ("igmp") ||
op_prg_odb_trace_active ())

static Boolean                log_call_scheduled = OPC_FALSE;
static int                    ip_pim_sm_efficiency_mode = -1;

/* Global RP lists.     */
List*       bootstrap_rp_lptr = OPC_NIL;
List*       auto_rp_lptr = OPC_NIL;
Boolean     boostrap_support = OPC_FALSE;
Boolean     auto_rp_agent_found = OPC_FALSE;
static Log_Handle       ip_igmp_host_config_warn_loghndl;
static Log_Handle       ip_igmp_host_lowlevel_error_loghndl;
static Boolean          ip_igmp_host_loghndls_init = OPC_FALSE;

typedef struct
      {
      int         interval;
      int         retry;
      int         req_lifetime;
      }
MipT_MN_Reg_Info;

typedef enum
      {
      MipC_MN_Timer_Rereg,
      MipC_MN_Timer_Agent,
      MipC_MN_Timer_Solicit,
      MipC_MN_Timer_Retry
      }
MipC_MN_Timer;

typedef enum IpT_Icmp_Echo_Message_Type
      {
      IpC_Icmp_Unspec = -1,
      IpC_Icmp_Echo_Reply = 0,
      IpC_Icmp_Echo_Request = 8
      } IpT_Icmp_Echo_Message_Type;
```

```
typedef struct
      {
      Stathandle        pkts_sent_stathandle;
      Stathandle        pkts_rcvd_stathandle;
      Stathandle        resp_time_stathandle;
      } IpT_Icmp_Stats;

typedef struct
      {
      IpT_Address       ip_grp_addr;
      int               interface;
      Evhandle          delay_timer_evh;
      int               delay_timer_id;
      Boolean           timer_on_flag;
      int               unsolicit_msg_count;
      Boolean           report_sent_flag;
      } IpT_Igmp_Host_Grp_Elem;

static Log_Handle       ip_igmp_host_config_warn_loghndl;
static Log_Handle       ip_igmp_host_lowlevel_error_loghndl;
static Boolean          ip_igmp_host_loghndls_init = OPC_FALSE;

typedef struct
      {
      InetT_Address     address;
      double            lifetime;
      int               pref_level;
      IpT_Interface_Info *incoming_intf_ptr;
      }
MipT_MN_Agent_Info;

typedef struct {

      Evhandle    expire_time;
      Objid       DUID;
      Objid       IAID;
      unsigned char     assign_type;

      InetT_Address_Range     assignment;             /* Holds the v4 addr or
v6 addr/prefix */
      InetT_Address                 link_local_addr;  /* The address used to
unicast back to the client */
} DhcpT_Srv_Assignment;

/* Function declarations. */
static void mip_mn_register (int, MipT_MN_Agent_Info, Boolean);
static void mip_mn_agent_timer_update (double);
static void mip_mn_tunneled_pk_stat_write (Packet*);
static void mip_mn_agent_solicit_pk_send (void);
static void mip_mn_agent_cache_update (MipT_MN_Agent_Info*, InetT_Address,
double, int, MipT_Invocation_Info*);
static void mip_mn_ip_pk_handle (MipT_Invocation_Info*);
static void mip_mn_ad_packet_parse (Packet*, int*, int*, InetT_Address*,
int*, int*);

void  dhcp_parse_msg();
int   dhcp_msg_server_duid_match(PrgT_List* rcv_pkt_opts);
```

```
int   dhcp_get_free_addr(int iface_index);
int   dhcp_get_free_prefix(int iface_index);
int   dhcp_get_iface_info_from_index(int index);

static void mip_mn_agent_solicit_pk_send_adv (void);//me

static void             ip_icmp_sv_init (void);
static void             ip_icmp_ping_specs_parse (IpT_Icmp_Temp_Ping_Specs*
ip_temp_ping_specs_ptr);
static void             ip_icmp_initial_echo_requests_schedule (void);
static Packet*          ip_icmp_echo_request_packet_create (int
req_index);//Me
static Packet*          ip_icmp_pkt_encapsulate (Packet* icmp_req_pkptr, int
req_index);
static void             ip_icmp_echo_reply_create (Packet* ip_dgram_pkptr,
Packet* ip_icmp_pkptr, Packet* icmp_reply_pkptr);
static void             ip_icmp_ip_process_invoke (Packet* ip_dgram_pkptr);
EXTERN_C_BEGIN
static void             ip_icmp_ip_process_invoker (void* state_ptr, int
code);//Me
EXTERN_C_END
static void             ip_icmp_ip_process_invoke_schedule (Packet*
ip_dgram_pkptr);
static double           ip_icmp_reply_stats_update (Packet* icmp_reply_pkptr,
int req_index);
static void             ip_icmp_next_echo_request_schedule (int req_index);
static void             ip_icmp_ping_stats_register (int stat_index, char*
dest_host_name);
static void             ip_icmp_sim_log_init ();
static void             ip_icmp_ip_dgram_discard (Packet* ip_dgram_pkptr);
EXTERN_C_BEGIN
static void             ip_icmp_request_timeout (void *state_ptr, int
index);//Me
EXTERN_C_END
static IpT_Icmp_Temp_Ping_Specs *
                        ip_icmp_ping_traffic_list_generate (Objid
node_objid);
static void             ip_icmp_dgram_fdstruct_update_for_reply
(IpT_Dgram_Fields* ip_dgram_fd_ptr);

static void             ip_pim_sm_do_init (void);
static void             ip_pim_sm_rte_plus (void);
static void             ip_pim_sm_data_pkt (void);
static void             ip_pim_sm_join_prune_msg (void);
static void             ip_pim_sm_data_rate_timer_expired (void);
static void             ip_pim_sm_register_msg (void);
static void             ip_pim_sm_register_stop_msg (void);
static void             ip_pim_sm_hello_msg (void);
static void             ip_pim_sm_rte_minus (void);
static void             ip_pim_sm_rpf_update (void);
static void             ip_pim_sm_fail_recover (void);

/****** Procedures *************/
static void ip_igmp_host_sv_init (void);
static IpT_Igmp_Host_Transition ip_igmp_host_get_transition_code (Packet**
igmp_pkt_pptr, Boolean* is_gs_query_msg_ptr);
```

```
static IpT_Igmp_Host_Grp_Elem*      ip_igmp_host_get_grp_elem (IpT_Address
ip_grp_addr, int interface);
static void ip_igmp_host_join_grp (IpT_Address ip_grp_addr, int interface);
//Me: Joinning multicast group by using CoA address.
static void ip_igmp_host_leave_grp (IpT_Address ip_grp_addr, int interface);
static void       ip_igmp_host_start_timer_for_grp (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr, double max_resp_time);
static void       ip_igmp_host_start_timer_for_grps_on_intf (int interface,
double max_resp_time);
static void       ip_igmp_host_cancel_timer_for_grp (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr);
static void       ip_igmp_host_timer_expired (int timer_code);
static IpT_Igmp_Host_Grp_Elem*      ip_igmp_host_grp_elem_alloc (void);
static void       ip_igmp_host_grp_elem_dealloc (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr);
static void       ip_igmp_host_error (const char* msg1, const char* msg2,
const char* msg3);
static void       ip_igmp_host_log_handles_init (void);
static void       ip_igmp_host_log_found_no_grp_info (const char*
ip_addr_str, int ip_intf_num);
static void       ip_igmp_host_grp_info_print (List* grp_lptr, Boolean
short_version);
static void       ip_igmp_host_igmp_msgs_sent_stat_update (OpT_Packet_Size
size); //Me

EXTERN_C_BEGIN
static void
      ip_igmp_host_ip_process_invoke (void *state_ptr, int
interface_to_send);
EXTERN_C_END

/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef      BIN
#undef      BOUT
#define     BIN         FIN_LOCAL_FIELD(_op_last_line_passed) = __LINE__ -
_op_block_origin;
#define     BOUT  BIN
#define     BINIT FIN_LOCAL_FIELD(_op_last_line_passed) = 0; _op_block_origin
= __LINE__;
#else
#define     BINIT
#endif /* #if !defined (VOSD_NO_FIN) */

/* State variable definitions */
typedef struct
      {
      /* Internal state tracking for FSM */
      FSM_SYS_STATE
      /* State Variables */
      MipC_Node_Type                      mip_node_type
;     /* Type of MN I am (MN or MR) */
      MipT_Proc_Info*                     proc_info_struct_ptr
;     /* General info shared between parent and me */
      MipT_MN_Reg_Info                    reg_info
;     /* registration information for this MN or MR */
```

```
	InetT_Address				subnet_bcast_addr
;	/* brodcast address of the mobile interface */
	InetT_Address				ha_address
;	/* IP address of the home agent interface */
	InetT_Address				home_address
;	/* local interface ip address */
	double					time_to_reregister
;	/* sim time for next registration */
	Evhandle				reregister_timer_ehndl
;	/* eventhandle for reregistration */
	int					reg_id
;	/* identification for pending registration */
	int					retry_counter
;	/* current number of registration retries */
	InetT_Address				agent_address
;	/* Current agent address serving me */
	Boolean					direct_reg
;	/* direct with HA or indirect through FA */
	MipT_MN_Agent_Info			latest_fa_info
;	/* Last FA information on FA other than the currnt one */
	Evhandle				agent_timer_ehndl
;	/* event handle for agent timeouts */
	MipT_MN_Agent_Info			latest_ha_info
;	/* Latest info on the HA from advertisements */
	IpT_Rte_Module_Data*			module_data
;	/* node wide IP module info */
	Stathandle				tunneled_pk_rcvd_sec_sh
;
	Stathandle		tunneled_bit_rcvd_sec_sh			;
	Evhandle		solicit_timer_ehndl				;
	/* Event handle for sending solicitation packet. */
	Boolean			solicitation					;
	/* Whether or not the MN/MR solicit when lost */
	int			solicit_count					;
	/* number of times solicitation was sent out before getting an ad */
	Stathandle		irdp_sent_pkts_sh				;
	Stathandle		irdp_sent_bits_sh				;
	Objid			node_objid					;
	Boolean			simultaneous_binding				;
	/* Flag indicating if this MN/MR will be asking for HA to keep */

	/* simulatneous binding.						*/
	Evhandle		reg_retry_timer_ehndl				;
	/* Event handle for registration retry in case of failure. */
	Stathandle		g_irdp_sent_bits_sh				;
	Boolean			loopback_intf					;
	/* flag indicating if this MN/MR is configured on the loopback
interface */
	int			current_agent_pref_level			;
	/* preference level of the current FA */
	IpT_Interface_Info*	current_roaming_intf				;
	Boolean			default_gateway					;
	/* Flag to indicate, have a default gateway setup */
	IpT_Address		last_default_addr				;
	} mobile_ip_mn_state;

typedef struct
```

```
      {
      /* Internal state tracking for FSM */
      FSM_SYS_STATE
      /* State Variables */
      Objid                  module_objid                                    ;
      Objid                  node_objid                                      ;
      OmsT_Pr_Handle         my_proc_handle                                  ;
      Objid                  udp_objid                                       ;
      IpT_Rte_Module_Data*     ip_support_module_ptr                         ;
      Ici*                   command_ici_ptr                                 ;
      int                    num_dhcp_interfaces                             ;
      /* Number of interfaces DHCP will be listening on. */
      DhcpT_Srv_Interface*     interfaces_config                             ;
      int                      input_strm                                    ;
      int                      output_strm                                   ;
      InetT_Address           server_multicast_addr                  ;  // Me
      DhcpT_Stathandles*       global_stats                                  ;
      DhcpT_Stathandles*       local_stats                                  ;
      int                      intrpt_code                                   ;
      Boolean                  logging                                      ;
      /* Boolean to indicate if global DHCP logging is turned on/off */
      Log_Handle              new_log_handle                                 ;
      Log_Handle              renew_log_handle                               ;
      Log_Handle              error_log_handle                               ;
      Log_Handle              expire_log_handle                              ;
      } dhcp_server_state;

static enum
      {IpC_Pim_Grp_Tbl_Grp_Addr = 0, IpC_Pim_Grp_Tbl_RP_Addr,
      IpC_Pim_Grp_Tbl_Src_Addr, IpC_Pim_Grp_Type,
      IpC_Pim_Grp_Tbl_In_Iface, IpC_Pim_Grp_Tbl_Out_Iface,
      IpC_Pim_Grp_Tbl_Num_Columns
      } IpC_Pim_Grp_Tbl_Table_Column_Index;

typedef struct
      {
      OpT_Int8          pim_sm_status;
      double            hello_period;
      double            hello_holdtime;
      IpT_Pim_Intf_Stat_Handle*     stat_handle_ptr;
      Evhandle                      hello_evhandle;
      int   priority;
      } IpT_Pim_Intf;

#define mip_node_type                    op_sv_ptr->mip_node_type
#define proc_info_struct_ptr             op_sv_ptr->proc_info_struct_ptr
#define reg_info                         op_sv_ptr->reg_info
#define subnet_bcast_addr                op_sv_ptr->subnet_bcast_addr
#define ha_address                       op_sv_ptr->ha_address
#define home_address                     op_sv_ptr->home_address
#define time_to_reregister               op_sv_ptr->time_to_reregister //Me
#define reregister_timer_ehndl           op_sv_ptr->reregister_timer_ehndl
#define reg_id                           op_sv_ptr->reg_id
#define retry_counter                    op_sv_ptr->retry_counter
#define agent_address                    op_sv_ptr->agent_address
#define direct_reg                       op_sv_ptr->direct_reg
#define latest_fa_info                   op_sv_ptr->latest_fa_info
```

```
#define agent_timer_ehndl              op_sv_ptr->agent_timer_ehndl
#define latest_ha_info                 op_sv_ptr->latest_ha_info
#define module_data                    op_sv_ptr->module_data
#define tunneled_pk_rcvd_sec_sh        op_sv_ptr->tunneled_pk_rcvd_sec_sh
#define tunneled_bit_rcvd_sec_sh       op_sv_ptr->tunneled_bit_rcvd_sec_sh
#define solicit_timer_ehndl            op_sv_ptr->solicit_timer_ehndl
#define solicitation                   op_sv_ptr->solicitation
#define solicit_count                  op_sv_ptr->solicit_count
#define irdp_sent_pkts_sh              op_sv_ptr->irdp_sent_pkts_sh
#define irdp_sent_bits_sh              op_sv_ptr->irdp_sent_bits_sh
#define node_objid                     op_sv_ptr->node_objid
#define simultaneous_binding           op_sv_ptr->simultaneous_binding
#define reg_retry_timer_ehndl          op_sv_ptr->reg_retry_timer_ehndl
//Me
#define g_irdp_sent_bits_sh            op_sv_ptr->g_irdp_sent_bits_sh
#define loopback_intf                  op_sv_ptr->loopback_intf
#define current_agent_pref_level       op_sv_ptr->current_agent_pref_level
#define current_roaming_intf           op_sv_ptr->current_roaming_intf
#define default_gateway                op_sv_ptr->default_gateway
#define last_default_addr              op_sv_ptr->last_default_addr

#define module_objid                   op_sv_ptr->module_objid
#define node_objid                     op_sv_ptr->node_objid
#define my_proc_handle                 op_sv_ptr->my_proc_handle
#define udp_objid                      op_sv_ptr->udp_objid
#define ip_support_module_ptr          op_sv_ptr->ip_support_module_ptr
#define command_ici_ptr                op_sv_ptr->command_ici_ptr
#define max_sol_tmout                  op_sv_ptr->max_sol_tmout
#define max_req_tmout                  op_sv_ptr->max_req_tmout
#define max_req_retries                op_sv_ptr->max_req_retries
#define init_renew_tmout               op_sv_ptr->init_renew_tmout
#define max_renew_tmout                op_sv_ptr->max_renew_tmout
#define init_rebind_tmout              op_sv_ptr->init_rebind_tmout
#define max_rebind_tmout               op_sv_ptr->max_rebind_tmout
#define send_iface                     op_sv_ptr->send_iface
#define last_trans_id                  op_sv_ptr->last_trans_id
#define rapid_commit                   op_sv_ptr->rapid_commit
#define server_rapid_commit            op_sv_ptr->server_rapid_commit
#define server_id                      op_sv_ptr->server_id
#define srv_str                        op_sv_ptr->srv_str
#define interfaces_config              op_sv_ptr->interfaces_config
#define gateway_node                   op_sv_ptr->gateway_node
#define input_strm                     op_sv_ptr->input_strm
#define output_strm                    op_sv_ptr->output_strm
#define snd_pkt_ptr                    op_sv_ptr->snd_pkt_ptr // Me
#define snd_pkt_opts                   op_sv_ptr->snd_pkt_opts // Me
#define snd_msg_type                   op_sv_ptr->snd_msg_type // Me
#define rcv_pkt_ptr                    op_sv_ptr->rcv_pkt_ptr  // Me
#define rcv_pkt_opts                   op_sv_ptr->rcv_pkt_opts // Me
#define rcv_msg_type                   op_sv_ptr->rcv_msg_type // Me
#define rcv_msg_trans                  op_sv_ptr->rcv_msg_trans // Me
#define RTprev                         op_sv_ptr->RTprev
#define expire_time                    op_sv_ptr->expire_time
#define rebind_time                    op_sv_ptr->rebind_time
#define retrans_count                  op_sv_ptr->retrans_count
#define intrpt_type                    op_sv_ptr->intrpt_type
#define intrpt_code                    op_sv_ptr->intrpt_code
```

```
#define num_interfaces                  op_sv_ptr->num_interfaces
#define next_evh                        op_sv_ptr->next_evh
#define server_multicast_addr           op_sv_ptr->server_multicast_addr
//Me
#define node_ll_addr                    op_sv_ptr->node_ll_addr
#define global_stats                    op_sv_ptr->global_stats
#define local_stats                     op_sv_ptr->local_stats
#define transaction_time                op_sv_ptr->transaction_time
#define logging                         op_sv_ptr->logging
#define new_log_handle                  op_sv_ptr->new_log_handle
#define renew_log_handle                op_sv_ptr->renew_log_handle
#define error_log_handle                op_sv_ptr->error_log_handle
#define expire_log_handle               op_sv_ptr->expire_log_handle
#define server_inet_addr                op_sv_ptr->server_inet_addr



/* These macro definitions will define a local variable called    */
/* "op_sv_ptr" in each function containing a FIN statement. */
/* This variable points to the state variable data structure,     */
/* and can be used from a C debugger to display their values.     */
#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC      mobile_ip_mn_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE     \
            op_sv_ptr = ((mobile_ip_mn_state *)(OP_SIM_CONTEXT_PTR-
>_op_mod_state_ptr));

#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC      dhcp_client_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE     \
            op_sv_ptr = ((dhcp_client_state *)(OP_SIM_CONTEXT_PTR-
>_op_mod_state_ptr));

/* Function Block */

#if !defined (VOSD_NO_FIN)
enum { _op_block_origin = __LINE__ + 2};
#endif

/* Transitional Executives */
void SendMsg()
      {
      /* Build a DHCP message with
       * appropriate options as determined by the nodes
       * configuration, and send the packet to the well known
       * DHCP multicast address.
       */
      DhcpT_Opt*              tmp_dhcp_opt;
      DhcpT_Cli_Assignment*   iface_config;
      double                  retrans_time;
      int                     int_code;
      PrgT_List_Cell*         list_cell_ptr;
      int                     oro_length        = 0;
      Boolean                 retrans_msg       = OPC_FALSE;
      char                    log_str[1000];
```

```
      FIN(SendMsg());

      /* Double check if a result of receiving a message from potentially
      many servers, if we've already sent a request to a particular server,
      then ignore this recent message if it's not from the server we sent the
      request to. */
      if (intrpt_type == OPC_INTRPT_STRM)
            {
            tmp_dhcp_opt = dhcp_optlist_get_opt(DHCPC_OPT_SERVERID,
rcv_pkt_opts);
            if(tmp_dhcp_opt->simple_data != server_id)
                  {
                  Discard();
                  FOUT;
                  }
            }

      /* If retransmitting, determine if this is a normal retransmission,
       * or if we are switching from a Renew phase to a Rebind phase.  If
       * it is a normal retransmission, we only increment the retransmission
       * count.
       */
      if ((intrpt_type == OPC_INTRPT_SELF) && (intrpt_code ==
DHCPC_MSG_TIMEOUT))
            {
            if((rebind_time == 0) || ((rebind_time > 0) && (op_sim_time() <
rebind_time)))
                  {
                  retrans_count++;
                  retrans_msg = OPC_TRUE;

                  if(LTRACE_ACTIVE)
                        op_prg_odb_print_minor ("Client timed out waiting for
response. Retransmitting..." , OPC_NIL);
                  }
            else
                  {
                  snd_msg_type = DHCPC_MSG_REBIND;

                  sprintf(log_str, "Current server %s not responding to DHCP
messages. "
                        "Sending Rebind message instead of Renew message to
obtain service "
                        "from any available server.", srv_str);
                  LOG(renew_log_handle, log_str, PrgC_Log_Severity_Warning);
                  }
            }

      /* If we received an Advertise, increment the transaction ID and send a
Request: */
      if ((intrpt_type == OPC_INTRPT_STRM) && (rcv_msg_type ==
DHCPC_MSG_ADVERTISE))
            {
            snd_msg_type = DHCPC_MSG_REQUEST;
            last_trans_id++;
            }
```

```
      /* If our assignment is close to expiring, increment the transaction ID
and send a Renew: */
      if ((intrpt_type == OPC_INTRPT_SELF) && (intrpt_code ==
DHCPC_ASSIGN_TIMEOUT))
            {
            snd_msg_type = DHCPC_MSG_RENEW;
            last_trans_id++;

            sprintf(log_str, "Current assignment(s) getting stale. Initiating
Renew "
                  "message exchange with current server %s", srv_str);
            LOG(renew_log_handle, log_str, PrgC_Log_Severity_Information);
            }

      /* Now, determine the interval for the next retransmission attempt
        Set the time :: Me*/

      if((snd_msg_type == DHCPC_MSG_SOLICIT) || (snd_msg_type ==
DHCPC_MSG_SOLICIT_RAPID))
            {

            RTprev = dhcp_get_retrans_time(RTprev, DHCPC_TRP_SOL_TIMEOUT,
max_sol_tmout);
            retrans_time = op_sim_time() + RTprev;

            int_code = DHCPC_MSG_TIMEOUT;
            }

      if(snd_msg_type == DHCPC_MSG_REQUEST)
            {
            if (retrans_count < max_req_retries)
                  {
                  RTprev = dhcp_get_retrans_time(RTprev,
DHCPC_TRP_REQ_TIMEOUT, max_req_tmout);
                  retrans_time = op_sim_time() + RTprev;
                  int_code = DHCPC_MSG_TIMEOUT;
                  }
            else
                  {
                  /* Immediately fallback to Solicit message by going back to
"Begin" FSM state */
                  op_intrpt_schedule_self(op_sim_time(), DHCPC_MSG_FALLBACK);
                  FOUT;
                  }
            }

      if(snd_msg_type == DHCPC_MSG_RENEW)
            {
            RTprev = dhcp_get_retrans_time(RTprev, init_renew_tmout,
max_renew_tmout);
            if((op_sim_time() + RTprev) > rebind_time)
                  {
                  /* Another scheduled retransmission would exceeded the
Renew phase, so we must switch to the Rebind phase at the rebind_time */
                  retrans_time = rebind_time;
                  }
```

```
            else
                  retrans_time = op_sim_time() + RTprev;

            int_code = DHCPC_MSG_TIMEOUT;
            }

      if(snd_msg_type == DHCPC_MSG_REBIND)
            {
            RTprev = dhcp_get_retrans_time(RTprev, init_rebind_tmout,
max_rebind_tmout);
            if((op_sim_time() + RTprev) > expire_time)
                  {
                  /* Another scheduled retransmission would exceed the valid
lifetime of the assignment.  */
                  retrans_time = expire_time;
                  int_code = DHCPC_MSG_FALLBACK;
                  }
            else
                  {
                  retrans_time = op_sim_time() + RTprev;
                  int_code = DHCPC_MSG_TIMEOUT;
                  }
            }

      /* Schedule the next timer self-interrupt: */
      if (op_ev_pending(next_evh))
            op_ev_cancel(next_evh);
      next_evh = op_intrpt_schedule_self(retrans_time, int_code);

      /* Now that we know we'll be sending a message (and not falling back,
       * create the message):
       */
      snd_pkt_ptr = dhcp_msg_create(snd_msg_type, last_trans_id);
      snd_pkt_opts = dhcp_optlist_create();

      prg_list_insert(snd_pkt_opts,
                  dhcp_opt_create_id(DHCPC_OPT_CLIENTID, node_objid),
                        PRGC_LISTPOS_TAIL);

      /* If the message have reached a server, include its Server ID, as long
as this message is NOT a Rebind message: */
      if((server_id != 0) && (snd_msg_type != DHCPC_MSG_REBIND))
            prg_list_insert(snd_pkt_opts,
                                    dhcp_opt_create_id(DHCPC_OPT_SERVERID,
server_id),
                                    PRGC_LISTPOS_TAIL);
      /* If this message is a rapid commit Solicit, add the option: */
      if(snd_msg_type == DHCPC_MSG_SOLICIT_RAPID)
            prg_list_insert(snd_pkt_opts,
                                    dhcp_opt_create(DHCPC_OPT_RAPID_COMMIT),
                                    PRGC_LISTPOS_TAIL);

      /* Add an option for every interface we are requesting configuration
info for: */
      if((snd_msg_type == DHCPC_MSG_REQUEST) || (snd_msg_type ==
DHCPC_MSG_RENEW)
            || (snd_msg_type == DHCPC_MSG_REBIND)
```

```
            || (snd_msg_type == DHCPC_MSG_SOLICIT_RAPID))
            {
            for(list_cell_ptr = prg_list_head_cell_get(&interfaces_config);
                  list_cell_ptr != PRGC_NIL;
                  list_cell_ptr = prg_list_cell_next_get(list_cell_ptr))
                  {
                  iface_config = (DhcpT_Cli_Assignment
*)prg_list_cell_data_get(list_cell_ptr);

                  if(iface_config->assign_type == DHCPC_ASSIGN_ADDR)
                        /* We're adding an option to request an address: */
                        tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_IA_NA);
                  else
                        /* We're adding an option to request a prefix: */
                        tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_IA_PD);

                  tmp_dhcp_opt->simple_data = iface_config->iface_index;


                  if(iface_config->configured == OPC_FALSE)
                        tmp_dhcp_opt->length = 12;
                  else
                        {
                        if(iface_config->assign_type == DHCPC_ASSIGN_ADDR)
                              tmp_dhcp_opt->length =
DHCPC_OPT_IA_NA_DATA_LEN;
                        else
                              tmp_dhcp_opt->length =
DHCPC_OPT_IA_PD_DATA_LEN;
                        }
                  /* Add this option to the total length count of the Option
Request option: */
                  oro_length += 2;

                  prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
PRGC_LISTPOS_TAIL);
                  }

            tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_ORO);
            tmp_dhcp_opt->length = oro_length;
            prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
      PRGC_LISTPOS_TAIL);

            }

      /* Finalize the packet and send it to the UDP module for delivery.*/
      dhcp_msg_finalize(snd_pkt_ptr, snd_pkt_opts);
      op_ici_install(command_ici_ptr);
      op_pk_send(snd_pkt_ptr, output_strm);
      op_ici_install(OPC_NIL);

      /* Update the DHCP stats for the message type sent: */
      if(retrans_msg == OPC_TRUE)
            dhcp_update_cli_stat(DHCPC_MSG_RETRANSMIT, 1);
      else
            /* If this is not a retransmission, reset the transaction time:
*/
```

```
            transaction_time = op_sim_time();

      dhcp_update_cli_stat(snd_msg_type, 1);

      /* Free the received packet memory: */
      if(intrpt_type == OPC_INTRPT_STRM)
            {
            op_pk_destroy(rcv_pkt_ptr);
            dhcp_optlist_destroy(rcv_pkt_opts);
            }

      FOUT;
      }

void Config()
      {
      /* Me:Once a mobile node has received a Reply message from access point
        which includes address: CoA address, this function is called to
      configure the assignments on the nodes interfaces on mobile node.
       */
      DhcpT_Opt*                    dhcp_opt_ptr;
      InetT_Address_Range*    inet_range_ptr;
      DhcpT_Cli_Assignment*   assign_ptr;
      int                           timers_configured = OPC_FALSE;
      int                           new_addr          = 0;
      int                           new_prefix        = 0;
      int                           renew_addr        = 0;
      int                           renew_prefix      = 0;
      char        log_str[1000], tmp_str[100], val_str[100], val2_str[100];
      PrgT_List_Cell*               list_cell_opt_ptr;
      PrgT_List_Cell*               list_cell_assign_ptr;
      double                        trans_delay;

      PrgT_List                     *dyn_assignment_lptr = OPC_NIL;
      int                           dyn_assignment_count = 0;
      IpT_Dynamic_Assignment  *dyn_assignment_ptr;
      IpT_Dynamic_Assignment_Array  *dyn_array_ptr;
      IpT_Dynamic_Assignment_Type   dyn_assignment_type;
      int                                 default_route_action =
DHCPC_DEFAULT_ROUTE_UNCHANGED;

      Ici                           *rcvd_ici_ptr;
      InetT_Address                 *rcvd_inet_address;

      FIN(Config());

      /* Me: Calculate the transaction delay for this message exchange:- some
      factor of assigning CoA address delay */
      trans_delay = op_sim_time() - transaction_time;

      /* Me: Get the server address from access point:- the address of
      foreign agent */
      rcvd_ici_ptr = op_ev_ici (op_ev_current ());
      op_ici_attr_get (rcvd_ici_ptr, "rem_addr", &rcvd_inet_address);

      /* Cycle through the options to find the IA_NA and IA_PD options: */
      for(list_cell_opt_ptr = prg_list_head_cell_get(rcv_pkt_opts);
```

```
            list_cell_opt_ptr != PRGC_NIL;
            list_cell_opt_ptr = prg_list_cell_next_get(list_cell_opt_ptr))
            {
            dyn_assignment_type = IpC_Dynamic_Assignment_None;

            dhcp_opt_ptr = (DhcpT_Opt
*)prg_list_cell_data_get(list_cell_opt_ptr);

            if((dhcp_opt_ptr->code != DHCPC_OPT_IA_NA) && (dhcp_opt_ptr->code
!= DHCPC_OPT_IA_PD))
                  continue;

            if(dhcp_opt_ptr->simple_data2 == DHCPC_NO_ADDR_AVAIL)
                  {
                  /* If this option indicated no address available, skip it
to go to the next option: */
                  sprintf(log_str, "Option received from server %s indicated
no address available.", srv_str);
                  LOG(error_log_handle, log_str, PrgC_Log_Severity_Notice);
                  continue;
                  }
            else if(dhcp_opt_ptr->simple_data2 == DHCPC_NO_PREFIX_AVAIL)
                  {
                  /* If this option indicated no prefix available, skip it to
go to the next option: */
                  sprintf(log_str, "Option received from server %s indicated
no prefix available.", srv_str);
                  LOG(error_log_handle, log_str, PrgC_Log_Severity_Notice);
                  continue;
                  }

            /* Me: In this point, the mobile node received the valid address
in the complex_data  */
            inet_range_ptr = (InetT_Address_Range *)(dhcp_opt_ptr-
>complex_data);

            /* Me: Store all info for this particular assignment, have to
check interface by cycling through the array of interfaces and comparing the
interface index with the value we received as the IAID.*/
            for(list_cell_assign_ptr =
prg_list_head_cell_get(&interfaces_config);
                  list_cell_assign_ptr != PRGC_NIL;
                  list_cell_assign_ptr =
prg_list_cell_next_get(list_cell_assign_ptr))
                  {
                  assign_ptr = (DhcpT_Cli_Assignment
*)prg_list_cell_data_get(list_cell_assign_ptr);
                  if(assign_ptr->iface_index == dhcp_opt_ptr->simple_data)
                        break;
                  }
            if(list_cell_assign_ptr == PRGC_NIL)
                  op_sim_end("Error while attempting to configure client:",
                                    "Unable to determine interface for
assignment.", OPC_NIL, OPC_NIL);
```

```
            /* Me: If mobile node already have an assignment on this
interface, determine if it's a renewal,or a different assignment (potentially
from a different server or access point). */
            if(assign_ptr->configured == OPC_TRUE)
                  {
                  if(inet_address_range_equal(&(assign_ptr->assignment),
inet_range_ptr) != OPC_TRUE)
                        {
                        /* Me: Set the IP notification to update */
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Addr_Update;

                        /* Me: Just for logging: */
                        inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                        inet_address_range_print(val2_str, inet_range_ptr);
                        if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                              sprintf(tmp_str, "address");
                        else
                              sprintf(tmp_str, "prefix");
                        sprintf(log_str, "Received new %s assignment for %s:
%s\n Old assignment: %s\n "
                              "New server: %s\n Lifetime: %d\n Transaction
time: %f sec",
                              tmp_str, assign_ptr->iface_name, val2_str,
val_str, srv_str,
                              dhcp_opt_ptr->simple_data2,   trans_delay);
                        LOG(new_log_handle, log_str,
PrgC_Log_Severity_Notice);

                        /* This is a different assignment from what we
already have destroy the existing assignment before installing the new one:*/
                        inet_address_range_destroy(&(assign_ptr-
>assignment));
                        assign_ptr->assignment = inet_address_range_ptr_copy(

      (InetT_Address_Range *)dhcp_opt_ptr->complex_data);

                        /* Me: Increment the count of configured
addresses/prefixes CoAs: */
                        if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                              new_addr++;
                        else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                              {
                              new_prefix++;

                              /* Also inform IP that the default route will
need to be updated */
                              default_route_action =
DHCPC_DEFAULT_ROUTE_UPDATE;
                              }

                        }
                  else
                        {
                  /* This is a successful renew of an existing assignment. */
```

```
                        /* No IP notification needed */

                        /* For logging: */
                        inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                        if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                              sprintf(tmp_str, "address");
                        else
                              sprintf(tmp_str, "prefix");
                        sprintf(log_str, "Renewing existing %s assignment for
%s: %s\n "
                              "From server: %s\n Lifetime: %d\n Transaction
time: %f sec",
                              tmp_str, assign_ptr->iface_name, val_str,
srv_str, dhcp_opt_ptr->simple_data2, trans_delay);
                        LOG(renew_log_handle, log_str,
PrgC_Log_Severity_Information);

                        /* Write appropriate stats: */
                        if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                              renew_addr++;
                        else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                              renew_prefix++;

                        }
                  }

            /* If this interface has not yet been configured, then copy the
option data containing the assignment into our local assignment array.
             */
            if(assign_ptr->configured != OPC_TRUE)
                  {
                  /* Set IP notification to create a new assignment */
                  if (assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Addr_Create;
                  else
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Prefix_Create;

                  assign_ptr->assignment = inet_address_range_ptr_copy(

      (InetT_Address_Range *)dhcp_opt_ptr->complex_data);

                  /* For logging: */
                  inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                  if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                        sprintf(tmp_str, "address");
                  else
                        sprintf(tmp_str, "prefix");

            sprintf(log_str, "Obtained initial %s assignment for %s: %s\n "
            "From server: %s\n Lifetime: %d\n Transaction time: %f sec",
            tmp_str, assign_ptr->iface_name, val_str, srv_str, dhcp_opt_ptr-
>simple_data2, trans_delay);
            LOG(new_log_handle, log_str, PrgC_Log_Severity_Information);
```

```
                /* Write appropriate stats: */
                if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                      new_addr++;
                else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                      {
                      new_prefix++;

                      /* Inform IP to set up a new default route */
                      default_route_action = DHCPC_DEFAULT_ROUTE_ADD;
                      }
                }

          /* This interface is now configured: */
          assign_ptr->configured = OPC_TRUE;

          /* Me : Set DHCP timers, the timers are only set once per
message. The protocol allows for each individual assignment to have it's
           *  own timers/expiration times. */
          if(timers_configured == OPC_FALSE)
                {
                /* Cancel any pending events: */
                if(op_ev_pending(next_evh)) { op_ev_cancel(next_evh); }

                /* Store times needed to properly implement protocol
timers: */
                expire_time = (op_sim_time()) + dhcp_opt_ptr->simple_data2;
                rebind_time = (op_sim_time()) + (.8 * dhcp_opt_ptr-
>simple_data2);

                /* Schedule the interrupt to start sending renews: */
                next_evh = op_intrpt_schedule_self(
                      op_sim_time() + (.5 * dhcp_opt_ptr->simple_data2),
DHCPC_ASSIGN_TIMEOUT);

                /* Reset the retransmission timer: */
                RTprev = 0;

                timers_configured = OPC_TRUE;
                }

          /* Notify IP process of interface assignments: */
          if (dyn_assignment_type != IpC_Dynamic_Assignment_None)
                {
                /* - Build the IpT_Dynamic_Assignment structure */
                dyn_assignment_ptr = (IpT_Dynamic_Assignment *)
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
                dyn_assignment_ptr->assignment_type       = (assign_ptr-
>assign_type == DHCPC_ASSIGN_ADDR) ? IpC_Dynamic_Assignment_Addr_Create :
IpC_Dynamic_Assignment_Prefix_Create;
                dyn_assignment_ptr->intf_index                  =
assign_ptr->iface_index;
                dyn_assignment_ptr->dynamic_addr_range    =
                      inet_address_range_ptr_copy (inet_range_ptr);

                /* - Append to list of dynamic assignments */
```

```
			if (dyn_assignment_lptr == OPC_NIL)
				dyn_assignment_lptr = op_prg_list_create ();
			op_prg_list_insert (dyn_assignment_lptr, 
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
			}

		}

	/* If we were unable to configure ANY interfaces, fallback to
	 * the begin state, after a waiting period.  Otherwise, calculate the
	 * total time of this transaction and record it in the stats.
	 */
	if(new_addr + new_prefix + renew_addr + renew_prefix == 0)
		{
		next_evh = op_intrpt_schedule_self(op_sim_time() + max_sol_tmout, 
DHCPC_MSG_FALLBACK);

		sprintf(log_str, "The message from server %s did not contain any 
configuration information. "
			"Soliciting service from any available server", srv_str);
		LOG(error_log_handle, log_str, PrgC_Log_Severity_Error);
		}
	else
		{
		/* Record the stats for the number of addr/prefixes new or 
renewed,
		 * and the total transaction time:
		 */
		if(new_addr)
			dhcp_update_cli_stat(DHCPC_COUNT_NEW_ADDR, new_addr);
		if(new_prefix)
			dhcp_update_cli_stat(DHCPC_COUNT_NEW_PREFIX, new_prefix);
		if(renew_addr)
			dhcp_update_cli_stat(DHCPC_COUNT_RENEW_ADDR, renew_addr);
		if(renew_prefix)
			dhcp_update_cli_stat(DHCPC_COUNT_RENEW_PREFIX, 
renew_prefix);

		dhcp_update_cli_stat(DHCPC_TRANSACTION_DELAY, trans_delay);
		}

	/* Notify IP by sending a remote interrupt with all assignments: */
	if (dyn_assignment_lptr != OPC_NIL)
		{
		void *prev_state;
		int i;

		/* Remove an old default route */
		if (default_route_action == DHCPC_DEFAULT_ROUTE_UPDATE)
			{
			/* - Build the IpT_Dynamic_Assignment structure */
			dyn_assignment_ptr = (IpT_Dynamic_Assignment *) 
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
			dyn_assignment_ptr->assignment_type	= 
IpC_Dynamic_Assignment_Default_Route_Add;
			dyn_assignment_ptr->intf_index			= 
send_iface;
```

```
                  dyn_assignment_ptr->dynamic_addr_range    =
inet_address_range_create (server_inet_addr, 0);

                  /* - Append to list of dynamic assignments */
                  op_prg_list_insert (dyn_assignment_lptr,
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
                  }

            /* Add the default route if necessary */
            if (default_route_action >= DHCPC_DEFAULT_ROUTE_ADD)
                  {
                  /* Make the received address the new server address */
                  server_inet_addr = inet_address_copy (*rcvd_inet_address);

                  /* - Build the IpT_Dynamic_Assignment structure */
                  dyn_assignment_ptr = (IpT_Dynamic_Assignment *)
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
                  dyn_assignment_ptr->assignment_type       =
IpC_Dynamic_Assignment_Default_Route_Add;
                  dyn_assignment_ptr->intf_index                  =
send_iface;
                  dyn_assignment_ptr->dynamic_addr_range    =
inet_address_range_create (server_inet_addr, 0);

                  /* - Append to list of dynamic assignments */
                  op_prg_list_insert (dyn_assignment_lptr,
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
                  }

            /* - Convert list of dynamic assignments to
IpT_Dynamic_Assignment_Array */
            dyn_array_ptr = (IpT_Dynamic_Assignment_Array*) op_prg_mem_alloc
(sizeof (IpT_Dynamic_Assignment_Array));
            dyn_array_ptr->assignment_count = op_prg_list_size
(dyn_assignment_lptr);
            dyn_array_ptr->assignments = (IpT_Dynamic_Assignment **)
                  op_prg_mem_alloc (dyn_array_ptr->assignment_count * sizeof
(IpT_Dynamic_Assignment *));

            for (i = 0; i < dyn_array_ptr->assignment_count; i++)
                  {
                  dyn_array_ptr->assignments [i] = (IpT_Dynamic_Assignment *)
                        op_prg_list_remove (dyn_assignment_lptr,
OPC_LISTPOS_HEAD);
                  }

            /* - Set assignment array as event state */
            prev_state = op_ev_state_install (dyn_array_ptr, OPC_NIL);

            /* - Schedule a remote interrupt for ip_dispatch */
            op_intrpt_schedule_remote (op_sim_time (),
IPC_DYNAMIC_ASSIGNMENTS_INTRPT_CODE,
                  ip_support_module_ptr->module_id);

            op_ev_state_install (prev_state, OPC_NIL);

            /* Destroy the list structure */
```

```
			prg_list_destroy (dyn_assignment_lptr, OPC_FALSE);
			}

		/* Free all memory associated with this packet: */
		op_pk_destroy(rcv_pkt_ptr);
		dhcp_optlist_destroy(rcv_pkt_opts);

		FOUT;
		}

void
dhcp_get_packet(void) /* Me: When mobile node received DHCP message*/
		{

		FIN(dhcp_get_packet());
		/* Consume the packet from the input stream: */
		rcv_pkt_ptr   = op_pk_get(input_strm);

		/* Parse the message to obtain values for the state variables: */
		dhcp_msg_parse(rcv_pkt_ptr, &rcv_msg_type, &rcv_msg_trans);

		/* Get the options from the received packet: */
		op_pk_fd_get_ptr (rcv_pkt_ptr, DHCPC_PK_FIELD_OPTIONS,
(void**)&rcv_pkt_opts);

		/* If this is a Reply message with a Rapid Commit option
		 * change the received message type:*/
		if((rcv_msg_type == DHCPC_MSG_REPLY)
			&& (dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts))
!= OPC_NIL)
			rcv_msg_type = DHCPC_MSG_REPLY_RAPID;

		/* Update the local stats for the message type received: */
		dhcp_update_cli_stat(rcv_msg_type, 1);

		FOUT;
		}

/* End of Function Block */

void
dhcp_client (OP_SIM_CONTEXT_ARG_OPT)
		{
#if !defined (VOSD_NO_FIN)
		int _op_block_origin = 0;
#endif
		FIN_MT (dhcp_client ());

			{
			/* Temporary Variables */
			Objid cli_params_objid, timers_objid, timers_row_objid, objid1;
			char  tmp_str[TMP_STR_SIZE];
			char  val_str[TMP_STR_SIZE];
			int         tmp_int;
			double      initial_solicit_delay;

			/* IP address manipulations: */
```

```
            InetT_Addr_Family addr_fam;

            DhcpT_Opt*  dhcp_opt_ptr;

            /* End of Temporary Variables */

            FSM_ENTER ("dhcp_client")

            FSM_BLOCK_SWITCH
                  {
                  /** state (Begin) enter executives **/
                  FSM_STATE_ENTER_FORCED (0, "Begin", state0_enter_exec,
"dhcp_client [Begin enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Begin enter
execs]", state0_enter_exec)
                        {
                        /* Me: Initialize protocol related info that is reset
every time we restart a Solicit message exchange:*/
                        server_rapid_commit     = OPC_TRUE;
                        retrans_count           = 0;
                        server_id               = 0;
                        RTprev                        = 0;
                        rebind_time             = 0;
                        expire_time             = 0;
                        rcv_pkt_ptr             = OPC_NIL;
                        rcv_pkt_opts            = OPC_NIL;

                        /* Create the DHCP message to be sent: */
                        if(Rapid_Commit)
                              snd_msg_type = DHCPC_MSG_SOLICIT_RAPID;
                        else
                              snd_msg_type = DHCPC_MSG_SOLICIT;

            snd_pkt_ptr = dhcp_msg_create(snd_msg_type, last_trans_id++);

                        /* Initialize the DHCP options list to be sent: */
                        snd_pkt_opts = dhcp_optlist_create();

                        /* Always insert our Client ID in every message: */
                        prg_list_insert(snd_pkt_opts,

      dhcp_opt_create_id(DHCPC_OPT_CLIENTID, node_objid),
                                          PRGC_LISTPOS_TAIL);

                        if(Rapid_Commit)
                              {
                              DhcpT_Opt*                    tmp_dhcp_opt
      = OPC_NIL;
                              DhcpT_Cli_Assignment*   iface_config      =
OPC_NIL;
                              int                                 oro_length
      = 0;
                              int                                 rep;

                              /* Insert the Rapid Commit option: */
                              prg_list_insert(snd_pkt_opts,
```

```
      dhcp_opt_create(DHCPC_OPT_RAPID_COMMIT),
                                                          PRGC_LISTPOS_TAIL);

                              /* Insert an option for every interface we're
requesting configuration for: */
                              for(rep = 0; rep < num_interfaces; rep++)
                                    {
                                    iface_config = (DhcpT_Cli_Assignment
*)prg_list_access(&interfaces_config, rep);
                                    if(iface_config->assign_type ==
DHCPC_ASSIGN_ADDR)
                                          /* Requesting an address: */
                                          tmp_dhcp_opt =
dhcp_opt_create(DHCPC_OPT_IA_NA);
                                    else
                                          /* Requesting a prefix: */
                                          tmp_dhcp_opt =
dhcp_opt_create(DHCPC_OPT_IA_PD);


      tmp_dhcp_opt->simple_data = iface_config->iface_index;

                                    /* Set the length of this option: */
                                    tmp_dhcp_opt->length = 12;

                                    /* Add this option to the total length
count of the Option Request option: */
                                    oro_length += 2;

                                    prg_list_insert(snd_pkt_opts,
tmp_dhcp_opt, PRGC_LISTPOS_TAIL);
                                    }

                              tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_ORO);
                              tmp_dhcp_opt->length = oro_length;
                              prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
      PRGC_LISTPOS_TAIL);
                              }

                        dhcp_msg_finalize(snd_pkt_ptr, snd_pkt_opts);

      /* Schedule a timer self-interrupt in case we need to retransmit: */
                        RTprev = dhcp_get_retrans_time(RTprev,
DHCPC_TRP_SOL_TIMEOUT, max_sol_tmout);
                        next_evh = op_intrpt_schedule_self(op_sim_time() +
RTprev, DHCPC_MSG_TIMEOUT);

                        /* Me: Send the message to the multicast address,
with an initial delay for this first Solicit message:*/
                        initial_solicit_delay =
op_dist_uniform(DHCPC_TRP_SOL_MAX_DELAY);
                        op_ici_install(command_ici_ptr);
                        op_pk_send_delayed(snd_pkt_ptr, output_strm,
initial_solicit_delay);
                        op_ici_install(OPC_NIL);
```

```
                        /* Log this initial transmission: */
                        if(logging == OPC_TRUE)
                              {
                              op_prg_log_handle_severity_set(&new_log_handle,
PrgC_Log_Severity_Information);
                              op_prg_log_entry_write_t(new_log_handle,
op_sim_time() + initial_solicit_delay,
                                    "Soliciting any server for new
configuration information");
                              }

                        /* Note this delayed transmission in the stats: */
                        if(Rapid_Commit)
                              {
                              op_stat_write_t(global_stats-
>msg_count_solicit_rapid, 1.0, op_sim_time() + initial_solicit_delay);
                              op_stat_write_t(local_stats-
>msg_count_solicit_rapid, 1.0, op_sim_time() + initial_solicit_delay);
                              }
                        else
                              {
                              op_stat_write_t(global_stats-
>msg_count_solicit, 1.0, op_sim_time() + initial_solicit_delay);
                              op_stat_write_t(local_stats->msg_count_solicit,
1.0, op_sim_time() + initial_solicit_delay);
                              }

                        /* Record the start time of this transaction: */
                        transaction_time = op_sim_time();
                        }
                        FSM_PROFILE_SECTION_OUT (state0_enter_exec)

                  /** state (Begin) exit executives **/
                  FSM_STATE_EXIT_FORCED (0, "Begin", "dhcp_client [Begin exit
execs]")

                  /** state (Begin) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Begin trans
conditions]", state0_trans_conds)
                  FSM_INIT_COND (Rapid_Commit)
                  FSM_TEST_COND (!Rapid_Commit)
                  FSM_TEST_LOGIC ("Begin")
                  FSM_PROFILE_SECTION_OUT (state0_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"Rapid_Commit", "", "Begin", "Wait_Reply", "tr_12", "dhcp_client [Begin ->
Wait_Reply : Rapid_Commit / ]")
                        FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;,
"!Rapid_Commit", "", "Begin", "Wait_Advertise", "tr_13", "dhcp_client [Begin
-> Wait_Advertise : !Rapid_Commit / ]")
                        }
            /*---------------------------------------------------------*/
```

```
                  /** state (Wait_Reply) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (1, "Wait_Reply",
state1_enter_exec, "dhcp_client [Wait_Reply enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (3,"dhcp_client")

                  /** state (Wait_Reply) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (1, "Wait_Reply", "dhcp_client
[Wait_Reply exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply exit
execs]", state1_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              if(server_id == 0)
                                    {
                                    /* If Mobile nodes have not yet
discovered a server, remember this new server's identification:*/
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state1_exit_exec)

                  /** state (Wait_Reply) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply trans
conditions]", state1_trans_conds)
                  FSM_INIT_COND (Rcv_Reply)
                  FSM_TEST_COND (Rcv_NonReply & !Rcv_Advertise)
                  FSM_TEST_COND (Msg_Tmout)
                  FSM_TEST_COND (Msg_Fail)
                  FSM_TEST_COND (Rcv_Advertise)
                  FSM_TEST_LOGIC ("Wait_Reply")
                  FSM_PROFILE_SECTION_OUT (state1_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, Config();,
"Rcv_Reply", "Config()", "Wait_Reply", "Idle", "tr_21", "dhcp_client
[Wait_Reply -> Idle : Rcv_Reply / Config()]")
```

```
                        FSM_CASE_TRANSIT (1, 1, state1_enter_exec,
Discard();, "Rcv_NonReply & !Rcv_Advertise", "Discard()", "Wait_Reply",
"Wait_Reply", "tr_24", "dhcp_client [Wait_Reply -> Wait_Reply : Rcv_NonReply
& !Rcv_Advertise / Discard()]")
                        FSM_CASE_TRANSIT (2, 1, state1_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Reply", "Wait_Reply", "tr_26",
"dhcp_client [Wait_Reply -> Wait_Reply : Msg_Tmout / SendMsg()]")
                        FSM_CASE_TRANSIT (3, 0, state0_enter_exec, ;,
"Msg_Fail", "", "Wait_Reply", "Begin", "tr_46", "dhcp_client [Wait_Reply ->
Begin : Msg_Fail / ]")
                        FSM_CASE_TRANSIT (4, 1, state1_enter_exec,
SendMsg();, "Rcv_Advertise", "SendMsg()", "Wait_Reply", "Wait_Reply",
"tr_51", "dhcp_client [Wait_Reply -> Wait_Reply : Rcv_Advertise /
SendMsg()]")
                        }
            /*---------------------------------------------------------*/

                  /** state (Wait_Advertise) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (2, "Wait_Advertise",
state2_enter_exec, "dhcp_client [Wait_Advertise enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (5,"dhcp_client")

                  /** state (Wait_Advertise) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (2, "Wait_Advertise", "dhcp_client
[Wait_Advertise exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Advertise
exit execs]", state2_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              if(server_id == 0)
                                    {
      /* If the mobile nodes have not yet discovered a server, remember this
new server's identification:*/
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state2_exit_exec)
```

```
                  /** state (Wait_Advertise) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Advertise trans
conditions]", state2_trans_conds)
                  FSM_INIT_COND (Rcv_Advertise)
                  FSM_TEST_COND (Rcv_NonAdvertise)
                  FSM_TEST_COND (Msg_Tmout)
                  FSM_TEST_LOGIC ("Wait_Advertise")
                  FSM_PROFILE_SECTION_OUT (state2_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec,
SendMsg();, "Rcv_Advertise", "SendMsg()", "Wait_Advertise", "Wait_Reply",
"tr_16", "dhcp_client [Wait_Advertise -> Wait_Reply : Rcv_Advertise /
SendMsg()]")
                        FSM_CASE_TRANSIT (1, 2, state2_enter_exec,
Discard();, "Rcv_NonAdvertise", "Discard()", "Wait_Advertise",
"Wait_Advertise", "tr_18", "dhcp_client [Wait_Advertise -> Wait_Advertise :
Rcv_NonAdvertise / Discard()]")
                        FSM_CASE_TRANSIT (2, 2, state2_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Advertise", "Wait_Advertise",
"tr_19", "dhcp_client [Wait_Advertise -> Wait_Advertise : Msg_Tmout /
SendMsg()]")
                        }
      /*---------------------------------------------------------*/

                  /** state (Idle) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (3, "Idle", state3_enter_exec,
"dhcp_client [Idle enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (7,"dhcp_client")

                  /** state (Idle) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (3, "Idle", "dhcp_client [Idle exit
execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Idle exit
execs]", state3_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state3_exit_exec)

                  /** state (Idle) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Idle trans
conditions]", state3_trans_conds)
                  FSM_INIT_COND (Msg_Rcv)
                  FSM_TEST_COND (Assign_Tmout)
                  FSM_TEST_COND (Msg_Fail)
```

```
                  FSM_DFLT_COND
                  FSM_TEST_LOGIC ("Idle")
                  FSM_PROFILE_SECTION_OUT (state3_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec,
Discard();, "Msg_Rcv", "Discard()", "Idle", "Idle", "tr_27", "dhcp_client
[Idle -> Idle : Msg_Rcv / Discard()]")
                        FSM_CASE_TRANSIT (1, 4, state4_enter_exec,
SendMsg();, "Assign_Tmout", "SendMsg()", "Idle", "Wait_Reply_Conf", "tr_28",
"dhcp_client [Idle -> Wait_Reply_Conf : Assign_Tmout / SendMsg()]")
                        FSM_CASE_TRANSIT (2, 0, state0_enter_exec, ;,
"Msg_Fail", "", "Idle", "Begin", "tr_55", "dhcp_client [Idle -> Begin :
Msg_Fail / ]")
                        FSM_CASE_TRANSIT (3, 3, state3_enter_exec, ;,
"default", "", "Idle", "Idle", "tr_56", "dhcp_client [Idle -> Idle : default
/ ]")
                        }
            /*---------------------------------------------------------*/
                  /** state (Wait_Reply_Conf) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (4, "Wait_Reply_Conf",
state4_enter_exec, "dhcp_client [Wait_Reply_Conf enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (9,"dhcp_client")


                  /** state (Wait_Reply_Conf) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (4, "Wait_Reply_Conf", "dhcp_client
[Wait_Reply_Conf exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply_Conf
exit execs]", state4_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              /* If mobile nodes have not yet discovered a
server, or if we are in the Rebind phase,remember this new server's
identification:*/
                              if((server_id == 0) || (snd_msg_type ==
DHCPC_MSG_REBIND))
                                    {
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
```

```
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state4_exit_exec)

                  /** state (Wait_Reply_Conf) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply_Conf trans
conditions]", state4_trans_conds)
                  FSM_INIT_COND (Rcv_Reply)
                  FSM_TEST_COND (Rcv_NonReply)
                  FSM_TEST_COND (Msg_Tmout)
                  FSM_TEST_COND (Msg_Fail)
                  FSM_TEST_LOGIC ("Wait_Reply_Conf")
                  FSM_PROFILE_SECTION_OUT (state4_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, Config();,
"Rcv_Reply", "Config()", "Wait_Reply_Conf", "Idle", "tr_29", "dhcp_client
[Wait_Reply_Conf -> Idle : Rcv_Reply / Config()]")
                        FSM_CASE_TRANSIT (1, 4, state4_enter_exec,
Discard();, "Rcv_NonReply", "Discard()", "Wait_Reply_Conf",
"Wait_Reply_Conf", "tr_32", "dhcp_client [Wait_Reply_Conf -> Wait_Reply_Conf
: Rcv_NonReply / Discard()]")
                        FSM_CASE_TRANSIT (2, 4, state4_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Reply_Conf", "Wait_Reply_Conf",
"tr_34", "dhcp_client [Wait_Reply_Conf -> Wait_Reply_Conf : Msg_Tmout /
SendMsg()]")
                        FSM_CASE_TRANSIT (3, 0, state0_enter_exec,
Unconfig();, "Msg_Fail", "Unconfig()", "Wait_Reply_Conf", "Begin", "tr_50",
"dhcp_client [Wait_Reply_Conf -> Begin : Msg_Fail / Unconfig()]")
                        }
            /*---------------------------------------------------------*/

                  /** state (Init) enter executives **/
                  FSM_STATE_ENTER_UNFORCED_NOLABEL (5, "Init", "dhcp_client
[Init enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Init enter
execs]", state5_enter_exec)
                        {
                        if (LTRACE_ACTIVE)
                              op_prg_odb_print_major ("DHCP Client: Begin
simulation" , OPC_NIL);

                        /* Initialize some variables: */
                        module_objid                  = op_id_self();
                        node_objid                    =
op_topo_parent(module_objid);
                        ip_support_module_ptr   = ip_support_module_data_get
(node_objid);
                        gateway_node                  =
ip_rte_node_is_gateway(ip_support_module_ptr);
                        last_trans_id                 = 1;
                        server_inet_addr        = INETC_ADDRESS_INVALID;

                        prg_list_init(&interfaces_config);
```

```
                        /* Determine if the nodes are logging DHCP: */
            op_ima_sim_attr_get(OPC_IMA_TOGGLE, "DHCP Logging", &logging);
                        if(LOGGING_ACTIVE)
                              {
                              new_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "New
Configuration", 100);
                              renew_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Renewal", 100);
                              error_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Protocol Error",
100);
                              expire_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Expiration",
100);
                              }

                        /* Register process in the Process Registry: */
            op_ima_obj_attr_get(module_objid, "process model", tmp_str);

      my_proc_handle = oms_pr_process_register(node_objid, module_objid,
                              op_pro_self(), tmp_str);
oms_pr_attr_set(my_proc_handle,
                              "protocol", OMSC_PR_STRING, "dhcp",
                              OPC_NIL);

                  /* Schedule a self interrupt to wait for lower layers */
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state5_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (11,"dhcp_client")

                  /** state (Init) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (5, "Init", "dhcp_client [Init exit
execs]")

                  /** state (Init) transition processing **/
                  FSM_TRANSIT_FORCE (8, state8_enter_exec, ;, "default", "",
"Init", "Wait_0", "tr_19_0", "dhcp_client [Init -> Wait_0 : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Wait_1) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (6, "Wait_1", state6_enter_exec,
"dhcp_client [Wait_1 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_1 enter
execs]", state6_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state6_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (13,"dhcp_client")
```

```
                  /** state (Wait_1) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (6, "Wait_1", "dhcp_client [Wait_1
exit execs]")

                  /** state (Wait_1) transition processing **/
                  FSM_TRANSIT_FORCE (9, state9_enter_exec, ;, "default", "",
"Wait_1", "Wait_2", "tr_19_4", "dhcp_client [Wait_1 -> Wait_2 : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Wait_3) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (7, "Wait_3", state7_enter_exec,
"dhcp_client [Wait_3 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_3 enter
execs]", state7_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state7_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (15,"dhcp_client")

                  /** state (Wait_3) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (7, "Wait_3", "dhcp_client [Wait_3
exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_3 exit
execs]", state7_exit_exec)
                        {

                        /* Read in all of our attribute values: */
                        op_ima_obj_attr_get(module_objid, "DHCPv6 Client
Parameters", &cli_params_objid);
                        objid1 = op_topo_child(cli_params_objid,
OPC_OBJTYPE_GENERIC, 0);


                        op_ima_obj_attr_get_str(objid1, "Sending Interface",
TMP_STR_SIZE, val_str);
                        send_iface =
dhcp_get_ipv6_table_index_from_name(val_str, ip_support_module_ptr);
                        if(send_iface == -1)
                              {
                              sprintf(tmp_str, "Unable to get interface index
for interface '%s'", val_str);
                              op_sim_end("Error resolving interface index:",
                                    tmp_str, OPC_NIL, OPC_NIL);
                              }
                        else
                              {
                  /* Remember the link layer address of this interface: */
                              IpT_Interface_Info*     ip_iface_elem_ptr =
inet_rte_intf_tbl_access(ip_support_module_ptr, send_iface);
                              node_ll_addr =
ip_rte_intf_link_local_addr_get(ip_iface_elem_ptr);
                              }
```

```
                        op_ima_obj_attr_get(objid1, "Rapid Commit",
&rapid_commit);

                        op_ima_obj_attr_get(objid1, "Timers", &timers_objid);
                        timers_row_objid = op_topo_child(timers_objid,
OPC_OBJTYPE_GENERIC, 0);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Solicit Timeout", &max_sol_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Request Timeout", &max_req_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Request Retries", &max_req_retries);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Initial
Renew Timeout", &init_renew_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Renew Timeout", &max_renew_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Initial
Rebind Timeout", &init_rebind_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Rebind Timeout", &max_rebind_tmout);

                        /* Create the ICI to be used with UDP: */
                        command_ici_ptr = op_ici_create ("udp_command_inet");

                        /* Create a receive port for this application: */
                        tmp_int = dhcp_connect_to_udp(DHCPC_PORT_CLIENT,
module_objid,
                              &udp_objid, &input_strm, &output_strm,
command_ici_ptr);

                        if (tmp_int != UDPC_IND_SUCCESS)
                              {
                              sprintf (tmp_str, "%d in response to
CREATE_PORT command", tmp_int);
                              op_sim_end ("Error: process model dhcp received
error", tmp_str, "", "");
                              }

            /* Me: Since we are a client, mobile node will always send to the
well known multicast address for DHCP servers.  Set this address on the ICI
here: */
                        addr_fam = InetC_Addr_Family_v6;
                        server_multicast_addr =
inet_address_create(DHCPC_ADDR_ALL_AGENTS_AND_SERVERS, addr_fam);
                        op_ici_attr_set_ptr(command_ici_ptr, "rem_addr",
&server_multicast_addr);

            /* Me: For multicast to work, mobile node need a setting into the
"strm_index" field of the UDP ici - this setting will eventually be written
into the "multicast_major_port" of the IP layer ICI.*/
                        op_ici_attr_set (command_ici_ptr, "strm_index",
IPC_MCAST_ALL_MAJOR_PORTS);
                        server_multicast_addr =
inet_address_create(DHCPC_ADDR_ALL_AGENTS_AND_SERVERS, addr_fam);
```

```
                /* Mobile node always send to the DHCP server port: */
                      op_ici_attr_set (command_ici_ptr, "rem_port",
DHCPC_PORT_SERVER);

                      /* Also, since we are a client, mobile node will
always send using our link local address: */
                      op_ici_attr_set (command_ici_ptr, "src_addr",
&node_ll_addr);


/* Build the list to contain configuration information for all existing
interfaces. For both routers and hosts.*/
                      if(gateway_node)
                            {
                            int         rep;
                            int         num_total_interfaces = 0;
                            Objid ip_group_objid, iface_info_objid,
iface_row_objid;
                            Objid gaddr_info_objid, gaddr_row_objid;
                            DhcpT_Cli_Assignment * iface_config;

                            /* First get the total number of interfaces on
this node.  We're only dealing with physical interfaces for now.*/
                            op_ima_obj_attr_get(node_objid, "IPv6
Parameters", &ip_group_objid);
                            objid1 = op_topo_child(ip_group_objid,
OPC_OBJTYPE_GENERIC, 0);
                            op_ima_obj_attr_get(objid1, "Interface
Information", &iface_info_objid);
                            num_total_interfaces =
op_topo_child_count(iface_info_objid, OPC_OBJTYPE_GENERIC);

                            for (rep = 0; rep < num_total_interfaces;
rep++)
                                  {

                                  iface_row_objid =
op_topo_child(iface_info_objid, OPC_OBJTYPE_GENERIC, rep);



                                  op_ima_obj_attr_get(iface_row_objid,
"Global Address(es)", &gaddr_info_objid);
                                  gaddr_row_objid =
op_topo_child(gaddr_info_objid, OPC_OBJTYPE_GENERIC, 0);

                                  /* Get the "Address": */
                                  op_ima_obj_attr_get_str(gaddr_row_objid,
"Address", 100, val_str);
                                  if(strstr(val_str, "DHCP"))
                                        {
                                        int  intf_table_index;
                                        char iface_str [8];

      /* First confirm that the interface has been added into the interface
table */
                                        op_ima_obj_attr_get_str
(iface_row_objid, "Name", 7, iface_str);
```

```
                                        if ((intf_table_index =
dhcp_get_ipv6_table_index_from_name (iface_str, ip_support_module_ptr)) == -
1)
                                              {
                                              char log_str [500];

                 printf (log_str, "Interface %s has DHCP specified, but is
not being used in the simulation. Confirm that it is connected, has a link-
local address, and no other global addresses are specified.", iface_str);
                                              LOG(error_log_handle,
log_str, PrgC_Log_Severity_Error);

                                              continue;
                                              }
                                        else
                                              {
           /* This interface is using DHCP for either address or prefix
assignment. */
                                              iface_config =
(DhcpT_Cli_Assignment *)op_prg_mem_alloc(sizeof(DhcpT_Cli_Assignment));
                                              iface_config->configured =
OPC_FALSE;
                                              iface_config->iface_index =
rep;
                                              if(strstr(val_str, "Prefix
Delegation"))
                                                    iface_config-
>assign_type = DHCPC_ASSIGN_PREFIX;
                                              else
                                                    iface_config-
>assign_type = DHCPC_ASSIGN_ADDR;


      op_ima_obj_attr_get_str(iface_row_objid, "Name", TMP_STR_SIZE,
tmp_str);
                                              iface_config->iface_name =
prg_string_copy(tmp_str);


      prg_list_insert(&interfaces_config, iface_config, PRGC_LISTPOS_TAIL);
                                              }
                                        }
                                  }
                            }
                      else
                            {
      /* Initialize the configuration data for only a single interface: */
                            DhcpT_Cli_Assignment * iface_config =
                                  (DhcpT_Cli_Assignment
*)op_prg_mem_alloc(sizeof(DhcpT_Cli_Assignment));
                            iface_config->configured = OPC_FALSE;
                            iface_config->iface_index = send_iface;
                            iface_config->assign_type = DHCPC_ASSIGN_ADDR;

                            iface_config->iface_name = prg_string_copy("N/A
(single interface host)");
```

```
                              prg_list_insert(&interfaces_config,
iface_config, PRGC_LISTPOS_TAIL);
                              }
                        num_interfaces = prg_list_size(&interfaces_config);

                        if(num_interfaces < 1)
                              {
                              LOG(error_log_handle, "No interfaces are
requesting configuration information. "
                                    "Exiting DHCP client.",
PrgC_Log_Severity_Error);
                              op_pro_destroy(op_pro_self());
                              }
                        /* Get the handles to the DHCP statistics: */
                        global_stats = dhcp_get_global_stathandles();
                        local_stats  =
dhcp_get_local_stathandles(DHCPC_CLIENT);
                        }
                        FSM_PROFILE_SECTION_OUT (state7_exit_exec)

                  /* Me: Register this address for multicast: */

      Inet_Address_Multicast_Register(server_multicast_addr, tmp_int,

      DHCPC_PORT_SERVER, ip_support_module_ptr);

                  /** state (Wait_3) transition processing **/
                  FSM_TRANSIT_FORCE (0, state0_enter_exec, ;, "default", "",
"Wait_3", "Begin", "tr_19_2", "dhcp_client [Wait_3 -> Begin : default / ]")
            /*---------------------------------------------------------*/

/** state (Wait_0) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (8, "Wait_0", state8_enter_exec,
"dhcp_client [Wait_0 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_0 enter
execs]", state8_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state8_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (17,"dhcp_client")


                  /** state (Wait_0) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (8, "Wait_0", "dhcp_client [Wait_0
exit execs]")


                  /** state (Wait_0) transition processing **/
                  FSM_TRANSIT_FORCE (6, state6_enter_exec, ;, "default", "",
"Wait_0", "Wait_1", "tr_19_3", "dhcp_client [Wait_0 -> Wait_1 : default / ]")
      /*---------------------------------------------------------*/
```

```
                  /** state (Wait_2) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (9, "Wait_2", state9_enter_exec,
"dhcp_client [Wait_2 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_2 enter
execs]", state9_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state9_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (19,"dhcp_client")

                  /** state (Wait_2) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (9, "Wait_2", "dhcp_client [Wait_2
exit execs]")

                  /** state (Wait_2) transition processing **/
                  FSM_TRANSIT_FORCE (7, state7_enter_exec, ;, "default", "",
"Wait_2", "Wait_3", "tr_19_1", "dhcp_client [Wait_2 -> Wait_3 : default / ]")
            /*---------------------------------------------------------*/
                  }
            FSM_EXIT (5,"dhcp_client")
            }
      }

/* Undefine shortcuts to state variables to avoid */
#undef module_objid
#undef node_objid
#undef my_proc_handle
#undef udp_objid
#undef ip_support_module_ptr
#undef command_ici_ptr
#undef max_sol_tmout
#undef max_req_tmout
#undef max_req_retries
#undef init_renew_tmout
#undef max_renew_tmout
#undef init_rebind_tmout
#undef max_rebind_tmout
#undef send_iface
#undef last_trans_id
#undef rapid_commit
#undef server_rapid_commit
#undef server_id
#undef srv_str
#undef interfaces_config
#undef gateway_node
#undef input_strm
#undef output_strm
#undef snd_pkt_ptr
#undef snd_pkt_opts
#undef snd_msg_type
#undef rcv_pkt_ptr
#undef rcv_pkt_opts
#undef rcv_msg_type
```

```
#undef rcv_msg_trans
#undef RTprev
#undef expire_time
#undef rebind_time
#undef retrans_count
#undef intrpt_type
#undef intrpt_code
#undef num_interfaces
#undef next_evh
#undef server_multicast_addr
#undef node_ll_addr
#undef global_stats
#undef local_stats
#undef transaction_time
#undef logging
#undef new_log_handle
#undef renew_log_handle
#undef error_log_handle
#undef expire_log_handle
#undef server_inet_addr

#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE

#define FIN_PREAMBLE_DEC
#define FIN_PREAMBLE_CODE

VosT_Obtype
_op_dhcp_client_init (int * init_block_ptr)
	{
	VosT_Obtype obtype = OPC_NIL;
	FIN_MT (_op_dhcp_client_init (init_block_ptr))

	obtype = Vos_Define_Object_Prstate ("proc state vars (dhcp_client)",
		sizeof (dhcp_client_state));
	*init_block_ptr = 10;

	FRET (obtype)
	}

VosT_Address
_op_dhcp_client_alloc (VosT_Obtype obtype, int init_block)
	{
#if !defined (VOSD_NO_FIN)
	int _op_block_origin = 0;
#endif
	dhcp_client_state * ptr;
	FIN_MT (_op_dhcp_client_alloc (obtype))

	ptr = (dhcp_client_state *)Vos_Alloc_Object (obtype);
	if (ptr != OPC_NIL)
		{
		ptr->_op_current_block = init_block;
#if defined (OPD_ALLOW_ODB)
		ptr->_op_current_state = "dhcp_client [Init enter execs]";
#endif
		}
```

```
	FRET ((VosT_Address)ptr)
	}

void
_op_dhcp_client_svar (void * gen_ptr, const char * var_name, void **
var_p_ptr)
	{
	dhcp_client_state		*prs_ptr;

	FIN_MT (_op_dhcp_client_svar (gen_ptr, var_name, var_p_ptr))

	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (void *)OPC_NIL;
		FOUT
		}
	prs_ptr = (dhcp_client_state *)gen_ptr;

	if (strcmp ("module_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->module_objid);
		FOUT
		}
	if (strcmp ("node_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->node_objid);
		FOUT
		}
	if (strcmp ("my_proc_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->my_proc_handle);
		FOUT
		}
	if (strcmp ("udp_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->udp_objid);
		FOUT
		}
	if (strcmp ("ip_support_module_ptr" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->ip_support_module_ptr);
		FOUT
		}
	if (strcmp ("command_ici_ptr" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->command_ici_ptr);
		FOUT
		}
	if (strcmp ("max_sol_tmout" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->max_sol_tmout);
		FOUT
		}
	if (strcmp ("max_req_tmout" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->max_req_tmout);
		FOUT
```

```
        }
if (strcmp ("max_req_retries" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->max_req_retries);
        FOUT
        }
if (strcmp ("init_renew_tmout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->init_renew_tmout);
        FOUT
        }
if (strcmp ("max_renew_tmout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->max_renew_tmout);
        FOUT
        }
if (strcmp ("init_rebind_tmout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->init_rebind_tmout);
        FOUT
        }
if (strcmp ("max_rebind_tmout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->max_rebind_tmout);
        FOUT
        }
if (strcmp ("send_iface" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->send_iface);
        FOUT
        }
if (strcmp ("last_trans_id" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->last_trans_id);
        FOUT
        }
if (strcmp ("rapid_commit" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rapid_commit);
        FOUT
        }
if (strcmp ("server_rapid_commit" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->server_rapid_commit);
        FOUT
        }
if (strcmp ("server_id" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->server_id);
        FOUT
        }
if (strcmp ("srv_str" , var_name) == 0)
        {
        *var_p_ptr = (void *) (prs_ptr->srv_str);
        FOUT
        }
if (strcmp ("interfaces_config" , var_name) == 0)
```

```
        {
        *var_p_ptr = (void *) (&prs_ptr->interfaces_config);
        FOUT
        }
    if (strcmp ("gateway_node" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->gateway_node);
        FOUT
        }
    if (strcmp ("input_strm" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->input_strm);
        FOUT
        }
    if (strcmp ("output_strm" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->output_strm);
        FOUT
        }
    if (strcmp ("snd_pkt_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->snd_pkt_ptr);
        FOUT
        }
    if (strcmp ("snd_pkt_opts" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->snd_pkt_opts);
        FOUT
        }
    if (strcmp ("snd_msg_type" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->snd_msg_type);
        FOUT
        }
    if (strcmp ("rcv_pkt_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rcv_pkt_ptr);
        FOUT
        }
    if (strcmp ("rcv_pkt_opts" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rcv_pkt_opts);
        FOUT
        }
    if (strcmp ("rcv_msg_type" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rcv_msg_type);
        FOUT
        }
    if (strcmp ("rcv_msg_trans" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rcv_msg_trans);
        FOUT
        }
    if (strcmp ("RTprev" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->RTprev);
```

```
        FOUT
        }
if (strcmp ("expire_time" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->expire_time);
        FOUT
        }
if (strcmp ("rebind_time" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->rebind_time);
        FOUT
        }
if (strcmp ("retrans_count" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->retrans_count);
        FOUT
        }
if (strcmp ("intrpt_type" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->intrpt_type);
        FOUT
        }
if (strcmp ("intrpt_code" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->intrpt_code);
        FOUT
        }
if (strcmp ("num_interfaces" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->num_interfaces);
        FOUT
        }
if (strcmp ("next_evh" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->next_evh);
        FOUT
        }
if (strcmp ("server_multicast_addr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->server_multicast_addr);
        FOUT
        }
if (strcmp ("node_ll_addr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->node_ll_addr);
        FOUT
        }
if (strcmp ("global_stats" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->global_stats);
        FOUT
        }
if (strcmp ("local_stats" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->local_stats);
        FOUT
        }
```

```
	if (strcmp ("transaction_time" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->transaction_time);
		FOUT
		}
	if (strcmp ("logging" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->logging);
		FOUT
		}
	if (strcmp ("new_log_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->new_log_handle);
		FOUT
		}
	if (strcmp ("renew_log_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->renew_log_handle);
		FOUT
		}
	if (strcmp ("error_log_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->error_log_handle);
		FOUT
		}
	if (strcmp ("expire_log_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->expire_log_handle);
		FOUT
		}
	if (strcmp ("server_inet_addr" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->server_inet_addr);
		FOUT
		}
	*var_p_ptr = (void *)OPC_NIL;

	FOUT
	}

static void /* Me: Joinning multicast tree by using CoA address*/
ip_igmp_host_join_grp (IpT_Address ip_grp_addr, int interface)
	{
	IpT_Igmp_Host_Grp_Elem*		grp_elem_ptr;
	List*					ip_grp_intf_list_ptr;
	Packet*						igmp_msg_pkptr;
	Packet*						ip_dgram_pkptr;
	char					msg0 [256], msg1 [256], msg2 [256];
	char					ip_addr_str [IPC_ADDR_STR_LEN];

	FIN (ip_igmp_host_join_grp (ip_grp_addr, interface));

	/* Generate a trace message	*/
	if (LTRACE_IGMP)
		{
		ip_address_print (ip_addr_str, ip_grp_addr);
```

```
            sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
            sprintf (msg1, "IP Interface              :           %d",
interface);
            op_prg_odb_print_major ("Received a Join request from an
application for: ", msg0, msg1, OPC_NIL);
            }

      /* Check if the group membership information for the given IP group
address already exists */
      grp_elem_ptr = ip_igmp_host_get_grp_elem (ip_grp_addr, interface);
      if (grp_elem_ptr == OPC_NIL)
            {

            grp_elem_ptr = ip_igmp_host_grp_elem_alloc ();

            /* Set the fields of the data structure */
            grp_elem_ptr->ip_grp_addr = ip_address_copy (ip_grp_addr);
            grp_elem_ptr->interface   = interface;

            /* Until now only one application on this node has joined this
group */
            grp_elem_ptr->num_of_apps = 1;

            /* Assign a timer ID to this group's delay timer */
            grp_elem_ptr->delay_timer_id = timer_id++;

            /* Initialize this field to the value of Robustness Variable
attribute minus 1 */
            grp_elem_ptr->unsolicit_msg_count = robustness_variable - 1;

            /* Since we are going to send an Unsolicited Membership Report
message, set this field to OPC_TRUE*/
                        */
            grp_elem_ptr->report_sent_flag = OPC_TRUE;

            /* Start the delay timer to send the next Unsolicited Membership
Report message. */

            if (igmp_sim_efficiency == OPC_FALSE)
                  {
                  /* Set the delay timer to OPC_TRUE, as we are going to
start */
                  /* the timer for sending the next Unsolicited Report
message */
                  grp_elem_ptr->timer_on_flag = OPC_TRUE;

                  /* Start the delay timer      */
                  grp_elem_ptr->delay_timer_evh = op_intrpt_schedule_self
(op_sim_time () +
                              unsolicited_report_interval, grp_elem_ptr-
>delay_timer_id);
                  }
            else
                  {
                  /* Set the unsolicited messages count to zero   */
                  grp_elem_ptr->unsolicit_msg_count = 0;
```

```
                  }

            /** Add this group membership information to the corresponding
list      **/

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                  {
                  ip_address_print (ip_addr_str, ip_grp_addr);
                  sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
                  sprintf (msg1, "IP Interface              :           %d",
interface);
                  strcpy (msg2, "to the table, which is maintained by IGMP
Host process.");
                  op_prg_odb_print_major ("Adding group membership
information for: ", msg0, msg1, msg2, OPC_NIL);
                  }

            /* Get the list for the interface on which this node has joined
the group */
            ip_grp_intf_list_ptr = (List *) op_prg_list_access
(ip_grp_list_ptr, interface);

            /* Add the group membership information data structure to the
list        */
            op_prg_list_insert (ip_grp_intf_list_ptr, grp_elem_ptr,
OPC_LISTPOS_TAIL);

            /** Create and send an Unsolicited Membership Report message **/

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                  {
                  ip_address_print (ip_addr_str, ip_grp_addr);
                  sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
                  sprintf (msg1, "IP Interface              :           %d",
interface);
                  op_prg_odb_print_major ("Sending an IGMP Unsolicited
Membership Report message for: ", msg0, msg1, OPC_NIL);
                  op_prg_odb_print_major ("Starting Delay timer for the above
group.", OPC_NIL);
                  }

            /* Create an IGMP Report message  */
            igmp_msg_pkptr = ip_igmp_create_igmp_msg
(IpC_Igmp_Membership_Report_Msg, 0, ip_grp_addr);

            /* Update IGMP messages sent statistics   */
            ip_igmp_host_igmp_msgs_sent_stat_update (op_pk_total_size_get
(igmp_msg_pkptr));

            /* Create an IP datagram for transmitting the IGMP message */
            ip_dgram_pkptr = ip_igmp_create_ipdgram (igmp_msg_pkptr,
ip_grp_addr);
```

```
            /* Schedule a procedure interrupt for the current simulation time
to invoke the IP process */
            op_intrpt_schedule_call (op_sim_time (), interface,
ip_igmp_host_ip_process_invoke, ip_dgram_pkptr);
            }
      else
            {
            /* Group membership information already exists. Another
application on this node has joined */
      /* this group. Increment the num_of_apps field's value by one */
            grp_elem_ptr->num_of_apps++;
            }
      FOUT;
      }

static void /* Using IGMP protocol to leave multicast group*/
ip_igmp_host_leave_grp (IpT_Address ip_grp_addr, int interface)
      {
      IpT_Igmp_Host_Grp_Elem*       grp_elem_ptr;
    List*                           ip_grp_intf_list_ptr;
      int                           num_grp_elems, i;
      Packet*                       igmp_msg_pkptr;
      Packet*                       ip_dgram_pkptr;
      char                          ip_addr_str [IPC_ADDR_STR_LEN];
      char                          msg0 [256], msg1 [256], msg2 [256];

      FIN (ip_igmp_host_leave_grp (ip_grp_addr, interface));
      /* Generate a trace message   */
      if (LTRACE_IGMP)
            {
            ip_address_print (ip_addr_str, ip_grp_addr);
            sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
            sprintf (msg1, "IP Interface              :           %d",
interface);
            op_prg_odb_print_major ("Received a Leave request from an
application for: ", msg0, msg1, OPC_NIL);
            }

      /* Get the list corresponding to the given interface */
      ip_grp_intf_list_ptr = (List *) op_prg_list_access (ip_grp_list_ptr,
interface);

    /* Determine the number of elements in the list */
      num_grp_elems = op_prg_list_size (ip_grp_intf_list_ptr);

/* Access the group membership information data structure for the given */
      /* IP group address     from the list
                        */
      for (i=0; i<num_grp_elems; i++)
            {
            /* Access ith element from the list */
            grp_elem_ptr = (IpT_Igmp_Host_Grp_Elem *)
                  op_prg_list_access (ip_grp_intf_list_ptr, i);

/* If this is the list element we are looking for, break from the loop */
            if (ip_address_equal (grp_elem_ptr->ip_grp_addr, ip_grp_addr))
```

```
                    {
                    break;
                    }
            }

      /* Sanity check */
      if (i == num_grp_elems)
            {
            /** The group membership information for the given group address
and interface doesn't exist in the list **/

            /* Report a log message */
            ip_address_print (ip_addr_str, ip_grp_addr);
            ip_igmp_host_log_found_no_grp_info (ip_addr_str, interface);

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                    {
sprintf (msg0, "IP Group Address          :           %s", ip_addr_str);
sprintf (msg1, "IP Interface              :           %d", interface);
strcpy (msg2, "doesn't exist in the list, which is maintained by IGMP Host
process.");
op_prg_odb_print_major ("Group membership information for: ", msg0, msg1,
msg2, OPC_NIL);
                    }
            }
      else
            {
            /* An application on the node has left this group. Decrement
num_of_apps field */
            grp_elem_ptr->num_of_apps--;
            if (grp_elem_ptr->num_of_apps == 0)
                  {
                  /* Generate a trace message   */
                  if (LTRACE_IGMP)
                        {
                        ip_address_print (ip_addr_str, ip_grp_addr);
                        sprintf (msg0, "IP Group Address          :
      %s", ip_addr_str);
                        sprintf (msg1, "IP Interface              :
      %d", interface);
                        strcpy (msg2, "from the table, which is maintained by
IGMP Host process.");
                        op_prg_odb_print_major ("Removing the group
membership information for: ", msg0, msg1, msg2, OPC_NIL);
                        }

                  /* Remove the group membership information from the list */
                  op_prg_list_remove (ip_grp_intf_list_ptr, i);

                  /* Cancel the delay timer for this group membership, if its
running */
                  if (grp_elem_ptr->timer_on_flag == OPC_TRUE)
                        {
                        op_ev_cancel (grp_elem_ptr->delay_timer_evh);
                        }
```

```
                  /* If this is the last node to send a Report message, send
a Leave message */
                  if (grp_elem_ptr->report_sent_flag == OPC_TRUE)
                        {
                        /* Generate a trace message   */
                        if (LTRACE_IGMP)
                              {
                              ip_address_print (ip_addr_str, ip_grp_addr);
                              sprintf (msg0, "IP Group Address          :
      %s", ip_addr_str);
                              sprintf (msg1, "IP Interface              :
      %d", interface);
                              op_prg_odb_print_major ("Sending an IGMP Leave
Group message for: ", msg0, msg1, OPC_NIL);
                              }

                        /* Create an IGMP Leave Group message */
                        igmp_msg_pkptr = ip_igmp_create_igmp_msg
(IpC_Igmp_Leave_Group_Msg, 0, grp_elem_ptr->ip_grp_addr);

                        /* Update IGMP messages sent statistics   */
                        ip_igmp_host_igmp_msgs_sent_stat_update
(op_pk_total_size_get (igmp_msg_pkptr));

                        /* Create an IP datagram to transmit the Leave group
message */
                        ip_dgram_pkptr = ip_igmp_create_ipdgram
(igmp_msg_pkptr, ip_address_create (IPC_ALL_ROUTERS_MULTICAST_ADDR));

                        /* Schedule a procedure interrupt for the current
simulation time to invoke the IP process */
                        op_intrpt_schedule_call (op_sim_time (),
grp_elem_ptr->interface, &ip_igmp_host_ip_process_invoke, ip_dgram_pkptr);
                        }
                  ip_igmp_host_grp_elem_dealloc (grp_elem_ptr);
                  }
            }

      FOUT;
      }

static void
mip_mn_register (int lifetime, MipT_MN_Agent_Info agent_info, Boolean direct)
      {
      IpT_Port_Info     port_info;
      double                        retry_timer;
      Ici*              reg_ici_ptr;

      /** PURPOSE: Registers with home agent.**/
      /** REQUIRES: lifetime it is requesting and care of address to use.
      **/
      /** EFFECTS: An interrupt sent to the reg manager.**/
      FIN (mip_mn_register (lifetime, agent_info, direct));

      /* Create Ici for mobile IP module. */
      reg_ici_ptr = op_ici_create ("mobile_ip_reg_ici");
```

```
      /* Set the foreign agent address for later reference. */
      agent_address = agent_info.address;
      current_agent_pref_level = agent_info.pref_level;
      current_roaming_intf = agent_info.incoming_intf_ptr;

      /* Set the appropriate values on the ici. */
      op_ici_attr_set (reg_ici_ptr, "reg_type", MipC_Reg_Type_Req);
      op_ici_attr_set (reg_ici_ptr, "home_address", inet_ipv4_address_get
(home_address));
      op_ici_attr_set (reg_ici_ptr, "home_agent", inet_ipv4_address_get
(ha_address));
      op_ici_attr_set (reg_ici_ptr, "lifetime_req", lifetime);
      op_ici_attr_set (reg_ici_ptr, "dest_address", inet_ipv4_address_get
(agent_address));
      op_ici_attr_set (reg_ici_ptr, "identification", ++reg_id);
      op_ici_attr_set (reg_ici_ptr, "s", simultaneous_binding);

      /* What is the type of the current registration? */
      if (direct)
            {
            /* Diect to the home agent. */
            direct_reg = OPC_TRUE;
            op_ici_attr_set (reg_ici_ptr, "care_of_address", OPC_NIL);
            }
      else
            {
            direct_reg = OPC_FALSE;
            op_ici_attr_set (reg_ici_ptr, "care_of_address",
inet_ipv4_address_get (agent_address));
            }

      /* Send the registrtation packet out. */
      op_ici_install (reg_ici_ptr);
      op_intrpt_schedule_process (proc_info_struct_ptr->mip_reg_mgr_phndl,
op_sim_time (), 0);
      op_ici_install (OPC_NIL);

      /* need to reset the default gateway to the agent, that are registering
with. */
      port_info = ip_rte_port_info_from_tbl_index (module_data,
ip_rte_intf_index_get (agent_info.incoming_intf_ptr));

      if (!default_gateway)
            {
            Ip_Cmn_Rte_Table_Entry_Add (module_data->ip_route_table, OPC_NIL,
                  IpI_Default_Addr, IpI_Default_Addr,
agent_info.incoming_intf_ptr->addr_range_ptr->address, port_info,
                  0, IP_CMN_RTE_TABLE_UNIQUE_ROUTE_PROTO_ID
(IPC_DYN_RTE_MOBILE_IP, IPC_NO_MULTIPLE_PROC), 0, OPC_NIL);

            /* Cache info. */
            default_gateway = OPC_TRUE;
            }
      else
            {
            Ip_Cmn_Rte_Table_Entry_Update (module_data->ip_route_table,
IpI_Default_Addr, IpI_Default_Addr,
```

```
                  last_default_addr, IP_CMN_RTE_TABLE_UNIQUE_ROUTE_PROTO_ID
(IPC_DYN_RTE_MOBILE_IP, IPC_NO_MULTIPLE_PROC),
                  agent_info.incoming_intf_ptr->addr_range_ptr->address,
port_info, 0, OPC_NIL);
            }

      /* Cache info. */
      last_default_addr = ip_rte_intf_addr_get
(agent_info.incoming_intf_ptr);

      /*  need to schedule retry in case that do not get the answer back. */
      retry_timer = reg_info.interval * pow (2.0, (double) retry_counter);
      if (retry_timer > (double) (reg_info.req_lifetime))
            {
            retry_timer = (double) (reg_info.req_lifetime);
            }

      reg_retry_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
retry_timer, MipC_MN_Timer_Retry);
      retry_counter++;

      if (MIP_TRACE)
            {
            op_prg_odb_print_major ("Trying registering with HA.", OPC_NIL);
            }

      FOUT;
      }

static void
mip_mn_agent_timer_update (double new_lifetime)
      {
      /** PURPOSE: Update the timer for agent timeouts.**/
      /** REQUIRES: new lifetime value from the last agent ad.    **/
      /** EFFECTS: timer event handle gets updated.**/
      FIN (mip_mn_agent_timer_update (new_lifetime));

      /* Cancel the current handle. */
      op_ev_cancel_if_pending (agent_timer_ehndl);

      /* Schedule new one. */
      agent_timer_ehndl = op_intrpt_schedule_self (new_lifetime,
MipC_MN_Timer_Agent);

      FOUT;
      }

static Packet* /* Switch the Active/Standby mode. */
ip_icmp_echo_request_packet_create (int req_index)
      {
      Packet*                 req_pkptr;


      FIN (ip_icmp_echo_request_packet_create (req_index));

      /* Create an ICMP packet to switch Active/Standby mode.     */
      req_pkptr = op_pk_create_fmt ("ip_icmp_echo");
```

```
      /* Set the "type" field to indicate that this is a    */
      /* request packet.                                             */
      op_pk_nfd_set (req_pkptr, "type", IpC_Icmp_Echo_Request);

      op_pk_nfd_set (req_pkptr, "identifier", req_index);

      op_pk_nfd_set (req_pkptr, "sequence number", ping_specs_ptr
[req_index].seq_number);

      /* set the source node object id in the packet.                */
      op_pk_nfd_set (req_pkptr, "source module objid", (double) my_objid);

/* Increment the sequence number for the next message send operation.   */
      ping_specs_ptr [req_index].seq_number++;

      if (ping_specs_ptr [req_index].ping_pattern_ptr->pkt_size > 0)
            {

            op_pk_bulk_size_set (req_pkptr, ping_specs_ptr
[req_index].ping_pattern_ptr->pkt_size * 8);
            }

      FRET (req_pkptr);
      }

static void
ip_icmp_request_timeout (void *state_ptr, int index)/* Me:Check timeout for
the connection*/
      {
      IpT_Icmp_Ping_Specs*    ping_spec_ptr;
      char                          dest_host_name [OMSC_HNAME_MAX_LEN];

      FIN (ip_icmp_request_timeout (void *state_ptr, int index));

      /* Get the ping_spec_ptr from the state ptr.    */
      ping_spec_ptr = (IpT_Icmp_Ping_Specs *)state_ptr;

      oms_tan_hname_get (ping_spec_ptr->dest_objid, dest_host_name);

      op_prg_log_entry_write (ip_icmp_timeout_log_handle,
            "ERROR(S):\n"
            " The echo request for destination (%s) \n"
            " sent at (%.2f) seconds failed to \n"
            " receive a response before the timeout \n"
            " interval of (%.5f) seconds. \n"
            "\n",

            dest_host_name, ping_spec_ptr [index].start_time,
            ping_spec_ptr [index].ping_pattern_ptr->timeout, ping_spec_ptr
[index].ping_pattern_ptr->timeout);

      FOUT;

      }

static void
```

```
ip_pim_sm_join_prune_msg (void) /* Me:Function Join/Standby message for
keeping route */
      {
      IpT_Pim_Sm_Rte_Entry*                rpt_rte_entry_ptr;
      IpT_Pim_Sm_Rte_Entry*                rte_entry_ptr;
      IpT_Address                          rp_ip_addr;
      IpT_Pim_Sm_Msg*                      pim_sm_msg_ptr;
      IpT_Pim_Sm_Join_Prune_Msg*           join_prune_msg_ptr;
      IpT_Pim_Sm_Join_Prune_List_Elem*     join_prune_lelem_ptr;
      IpT_Pim_Sm_Join_Prune_Src*           join_prune_src_ptr;
      int                                  i, j;
      Boolean                              send_join_prune = OPC_FALSE;

      FIN (ip_pim_sm_join_prune_msg ());

      /* Fist check that PIM-SM is supported on the interface.    */
      if (intf_array [pkt_recvd_intf].pim_sm_status == OPC_FALSE)
            {
            /* The interface does not support PIM-SM. */

            op_pk_destroy (pimsm_pkptr);
            op_pk_destroy (ip_dgram_pkptr);

            FOUT;
            }

      /* Obtain the Join/Prune message from the PIM-SM packet     */
      op_pk_nfd_access (pimsm_pkptr, "message", &pim_sm_msg_ptr);
      join_prune_msg_ptr = (IpT_Pim_Sm_Join_Prune_Msg *) pim_sm_msg_ptr-
>msg_ds_ptr;

      if (ip_pim_sm_is_my_address (join_prune_msg_ptr-
>upstream_neighbor_addr) == OPC_TRUE)
            {
            /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
            if (LTRACE_PIM_SM_JOIN_PRUNE)
                  {
                  char                                           ip_addr_str
[IPC_ADDR_STR_LEN];
                  char                                           info0
[256], info1 [256];
                  char                                           msg0 [256];
                  ip_address_print (ip_addr_str, join_prune_msg_ptr-
>upstream_neighbor_addr);
                  sprintf (info0, "Upstream Neighbor Address      :
      %s", ip_addr_str);
                  sprintf (info1, "Number of Groups               :
      %d", join_prune_msg_ptr->num_grps);
                  sprintf (msg0, "Received a PIM-SM Join/Prune message on
interface, %d with: ", pkt_recvd_intf);
                  op_prg_odb_print_major (msg0, info0, info1, OPC_NIL);
                  }
#endif

      /* For each IP group in the list, process the Join and Prune list */
            for (i=0; i<join_prune_msg_ptr->num_grps; i++)
```

```
                  {
                  /* Obtain ith element from the Join/Prune list  */
                  join_prune_lelem_ptr = (IpT_Pim_Sm_Join_Prune_List_Elem *)
op_prg_list_access (join_prune_msg_ptr->join_prune_lptr, i);

                  /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                  if (LTRACE_PIM_SM_JOIN_PRUNE)
                        {
                        char        info0 [256];
                        char        grp_addr_str [IPC_ADDR_STR_LEN];
                        ip_address_print (grp_addr_str, join_prune_lelem_ptr-
>grp_addr);
                        sprintf (info0, "IP Group Address         :
      %s", grp_addr_str);
                        op_prg_odb_print_major ("Processing Join/Prune list
in the PIM-SM Join/Prune message for the group: ", info0, OPC_NIL);
                        }
#endif

                  /* Process each source in the Join list   */
                  for (j=0; j<join_prune_lelem_ptr->num_join_src; j++)
                        {
                        /* Obtain the jth element from the Join list    */
                        join_prune_src_ptr = (IpT_Pim_Sm_Join_Prune_Src *)
op_prg_list_access (join_prune_lelem_ptr->join_src_lptr, j);

                        /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                        if (LTRACE_PIM_SM_JOIN_PRUNE)
                              {
                              char
      ip_addr_str [IPC_ADDR_STR_LEN];
                              char
      info0 [256], info1 [256], info2 [256];
                              ip_address_print (ip_addr_str,
join_prune_src_ptr->src_addr);
                              sprintf (info0, "Source IP Address        :
      %s", ip_addr_str);
                              sprintf (info1, "WC-bit                   :
      %d", join_prune_src_ptr->wc_bit);
                              sprintf (info2, "RPT-bit                  :
      %d", join_prune_src_ptr->rpt_bit);
                              op_prg_odb_print_major ("Processing the
following Source element in the Join list: ", info0, info1, info2, OPC_NIL);
                              }
#endif

                        /* Get the route entry for this group     */
                        rte_entry_ptr = ip_pim_sm_mcast_rte_entry_get
(join_prune_lelem_ptr->grp_addr, join_prune_src_ptr->src_addr,

                  join_prune_src_ptr->wc_bit);

                        /* Print out information about the route entry that
was found.  */
#ifndef OPD_NO_DEBUG
```

```
                        ip_pim_sm_entry_exists_odb_print (rte_entry_ptr);
#endif

                        if ((join_prune_src_ptr->wc_bit == OPC_TRUE) &&
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                              {

                              if (rte_entry_ptr == OPC_NIL)
                                    {

                                    rte_entry_ptr =
ip_pim_sm_rte_entry_wc_create (join_prune_src_ptr->src_addr,

      join_prune_lelem_ptr->grp_addr);

                                    /* Add this route entry to the multicast
route table */
                  ip_pim_sm_mcast_rte_entry_add (rte_entry_ptr, OPC_TRUE);

                                    /* This case require a join/prune message
to be sent. */
                  send_join_prune = OPC_TRUE;
                                    }

                              if (rte_entry_ptr->in_intf_addr !=
inet_ipv4_address_get (ip_dgram_fdptr->src_addr))
                                    ip_pim_sm_oif_table_add (rte_entry_ptr,
pkt_recvd_intf, inet_ipv4_address_get (ip_dgram_fdptr->src_addr),
                                          join_prune_msg_ptr->hold_time,
OPC_FALSE);

      ip_pim_sm_all_spt_rte_entries_out_intf_add (rte_entry_ptr,

                  pkt_recvd_intf,

                  inet_ipv4_address_get (ip_dgram_fdptr->src_addr),

                  join_prune_msg_ptr->hold_time,

                  OPC_FALSE,

                  join_prune_lelem_ptr->prune_src_lptr);
                              }
                        else if ((join_prune_src_ptr->wc_bit == OPC_FALSE) &&
(join_prune_src_ptr->rpt_bit == OPC_FALSE))
                              {

                              if (rte_entry_ptr == OPC_NIL)
                                    {

                                    rp_ip_addr = ip_pim_sm_rp_addr_get
(rp_hash_table_ptr, rp_lptr, join_prune_lelem_ptr->grp_addr,
bsr_hash_mask_length);

                        /* If RP information is not found, its an error */
                                    if (rp_ip_addr == IPC_ADDR_INVALID)
                                          {
```

```
                                          /* Report a log message */

      ipnl_protwarn_mcast_rp_unknown_log_add (join_prune_lelem_ptr->grp_addr,
ip_module_data_ptr->node_id);
                                          }

                                    rpt_rte_entry_ptr =
ip_pim_sm_mcast_rte_entry_get (join_prune_lelem_ptr->grp_addr, rp_ip_addr,
OPC_TRUE);

      rte_entry_ptr = ip_pim_sm_rte_entry_spt_create (join_prune_src_ptr-
>src_addr, join_prune_lelem_ptr->grp_addr,
                                          rpt_rte_entry_ptr, OPC_FALSE);
                                    if (rpt_rte_entry_ptr != OPC_NIL)
                                          {
      op_prg_list_elems_copy (rpt_rte_entry_ptr->out_intf_table_lptr,
rte_entry_ptr->out_intf_table_lptr);
                                          }

                  /* Add this route entry to the multicast route table  */
                                    ip_pim_sm_mcast_rte_entry_add
(rte_entry_ptr, OPC_FALSE);

            /* This case requires sending a join/prune message.         */
                                    send_join_prune = OPC_TRUE;
                                    }
                              else
                                    {

                                    if (rte_entry_ptr->rpt_flag == OPC_TRUE)
                                          {
                                          /* Clear the RPT-bit    */
                                          ip_pim_sm_entry_clear_rpt_bit
(rte_entry_ptr, OPC_FALSE);

                  /* This case requires sending a join/prune message    */
                                          send_join_prune = OPC_TRUE;
                                          }
                                    }

                              if (inet_ipv4_address_get (ip_dgram_fdptr-
>src_addr) != rte_entry_ptr->in_intf_addr)
                                    ip_pim_sm_oif_table_add (rte_entry_ptr,
pkt_recvd_intf, inet_ipv4_address_get (ip_dgram_fdptr->src_addr),
                                          join_prune_msg_ptr->hold_time,
OPC_FALSE);
                              }
                        if (send_join_prune == OPC_TRUE)
                              ip_pim_sm_send_join_prune_msg (rte_entry_ptr,
OPC_TRUE);

                        /* Reset the flag to FALSE.               */
                        send_join_prune = OPC_FALSE;
                        }

                  /* Process each source in the Prune list  */
                  for (j=0; j<join_prune_lelem_ptr->num_prune_src; j++)
```

```
                        {
                        /* Obtain the jth element from the Prune list   */
                        join_prune_src_ptr = (IpT_Pim_Sm_Join_Prune_Src *)
op_prg_list_access (join_prune_lelem_ptr->prune_src_lptr, j);

                        /* Get the route entry for this group     */
                        rte_entry_ptr = ip_pim_sm_mcast_rte_entry_get
(join_prune_lelem_ptr->grp_addr, join_prune_src_ptr->src_addr,

                  join_prune_src_ptr->wc_bit);

                        /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                        if (LTRACE_PIM_SM_JOIN_PRUNE)
                              {
                              char
      ip_addr_str [IPC_ADDR_STR_LEN];
                              char
      info0 [256], info1 [256], info2 [256];
                              ip_address_print (ip_addr_str,
join_prune_src_ptr->src_addr);
                              sprintf (info0, "Source IP Address        :
      %s", ip_addr_str);
                              sprintf (info1, "WC-bit                   :
      %d", join_prune_src_ptr->wc_bit);
                              sprintf (info2, "RPT-bit                  :
      %d", join_prune_src_ptr->rpt_bit);
                              op_prg_odb_print_major ("Processing the
following Source element in the Prune list: ", info0, info1, info2, OPC_NIL);
                              }
#endif

#ifndef OPD_NO_DEBUG
                        ip_pim_sm_entry_exists_odb_print (rte_entry_ptr);
#endif

                        if ((join_prune_src_ptr->wc_bit == OPC_FALSE) &&
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                              {
      if (rte_entry_ptr == OPC_NIL)
                                    {
                                    /* Get the RP address for the group */
                                    rp_ip_addr = ip_pim_sm_rp_addr_get
(rp_hash_table_ptr, rp_lptr, join_prune_lelem_ptr->grp_addr,
bsr_hash_mask_length);

                        /* If RP information is not found, its an error */
                                    if (rp_ip_addr == IPC_ADDR_INVALID)
                                          {
      ipnl_protwarn_mcast_rp_unknown_log_add (join_prune_lelem_ptr->grp_addr,
ip_module_data_ptr->node_id);
                                          }
                        /* Get the (*, G) entry */
                                    rpt_rte_entry_ptr =
ip_pim_sm_mcast_rte_entry_get (join_prune_lelem_ptr->grp_addr, rp_ip_addr,
OPC_TRUE);
```

```
                                    if (rpt_rte_entry_ptr != OPC_NIL)
                                          {
                                          rte_entry_ptr = 
ip_pim_sm_rte_entry_spt_create (join_prune_src_ptr->src_addr, 

                                    join_prune_lelem_ptr->grp_addr, 

                                    rpt_rte_entry_ptr, 

                                    OPC_FALSE);

                                          ip_pim_sm_entry_set_rpt_bit 
(rte_entry_ptr, OPC_FALSE);

                                          rte_entry_ptr->in_intf = 
rpt_rte_entry_ptr->in_intf;
                                          rte_entry_ptr->in_intf_addr = 
ip_address_copy(rpt_rte_entry_ptr->in_intf_addr);
                                          op_prg_list_elems_copy 
(rpt_rte_entry_ptr->out_intf_table_lptr, rte_entry_ptr->out_intf_table_lptr);

                  /* Add this route entry to the multicast route table  */
                                          ip_pim_sm_mcast_rte_entry_add 
(rte_entry_ptr, OPC_FALSE);
                                          }
                                    }
                              }

                        if (rte_entry_ptr != OPC_NIL)
                              {

                              if (op_prg_list_size (rte_entry_ptr-
>out_intf_table_lptr) == 0)
                                    {
                        /* Check if the data timer should be reset.     */
                                    if ((join_prune_src_ptr->wc_bit == 
OPC_FALSE) && (join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                          {

            ip_pim_sm_reset_data_timer (rte_entry_ptr, 
LTRACE_PIM_SM_JOIN_PRUNE);
                                          }

            /* Nothing more needs to be done.  Just continue.           */
                                    continue;
                                    }

ip_pim_sm_oif_table_remove (rte_entry_ptr->out_intf_table_lptr, 
pkt_recvd_intf, OPC_FALSE);

                              if ((join_prune_src_ptr->wc_bit == OPC_TRUE) && 
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                    {

      ip_pim_sm_all_spt_rte_entries_out_intf_remove (join_prune_lelem_ptr-
>grp_addr, pkt_recvd_intf, OPC_FALSE);
                                    }
```

```
                              if (op_prg_list_size (rte_entry_ptr-
>out_intf_table_lptr) == 0)
                                    {

                                    ip_pim_sm_send_join_prune_msg
(rte_entry_ptr, OPC_FALSE);

                                    if (((join_prune_src_ptr->wc_bit ==
OPC_FALSE) && (join_prune_src_ptr->rpt_bit == OPC_TRUE)) &&
                                          ((rte_entry_ptr->wc_flag ==
OPC_FALSE) && (rte_entry_ptr->rpt_flag == OPC_FALSE)))
                                          {
                  /* Set the RPT bit to TRUE for this route entry.      */
                                          ip_pim_sm_entry_set_rpt_bit
(rte_entry_ptr, OPC_FALSE);

                  /* Set the SPT bit to FALSE for this route entry.     */
                                          ip_pim_sm_entry_clear_spt_bit
(rte_entry_ptr, OPC_FALSE);
                                          }

                  /* Remove the route entry from the multicast route
table,only if its is not a (S, G)RPT-bit entry  */
                                    if (!((rte_entry_ptr->wc_flag ==
OPC_FALSE) && (rte_entry_ptr->rpt_flag == OPC_TRUE)))
                                          {
#ifndef OPD_NO_DEBUG
                                          if (LTRACE_PIM_SM_JOIN_PRUNE)
                                                {
                                                op_prg_odb_print_major ("The
Out Interface table size of the route entry became zero.", OPC_NIL);
                                                }
#endif
ip_pim_sm_mcast_rte_entry_remove (rte_entry_ptr->grp_addr, rte_entry_ptr-
>src_addr, rte_entry_ptr->wc_flag);

                                          continue;
                                          }
                                    }

                              if ((join_prune_src_ptr->wc_bit == OPC_FALSE)
&& (join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                    {
                                    ip_pim_sm_reset_data_timer
(rte_entry_ptr, LTRACE_PIM_SM_JOIN_PRUNE);
                                    }
                              }
                        }
                  }
            }

      /* We no longer need the PIM-SM packet and the  */
      /* IP datagram. Destroy them  */
      op_pk_destroy (pimsm_pkptr);
      op_pk_destroy (ip_dgram_pkptr);
```

```
	FOUT;
	}

static void
mip_mn_tunneled_pk_stat_write (Packet*    pk_ptr)
	{
	/** PURPOSE: Write statistic for tunneld packets received.**/
	FIN (mip_mn_tunneled_pk_stat_write (pk_ptr));

	/* Write the stats. */
	op_stat_write (tunneled_pk_rcvd_sec_sh, 1.0);
	op_stat_write (tunneled_bit_rcvd_sec_sh, op_pk_total_size_get
(pk_ptr));
	op_stat_write (tunneled_pk_rcvd_sec_sh, 0.0);
	op_stat_write (tunneled_bit_rcvd_sec_sh, 0.0);

	FOUT;
	}

static void
mip_mn_agent_solicit_pk_send (void)
	{
	Packet*           solicit_pkptr;
	double            solicit_interval;

	/** PURPOSE: Send the ICMP agent solicitation packet.**/
	/** REQUIRES: none.     **/
	/** EFFECTS: Packet will be given to IP to handle.**/
	FIN (mip_mn_agent_solicit_pk_send (void));

	/* Time to send out the solicitation. */
	solicit_pkptr = op_pk_create_fmt ("mobile_ip_irdp_solicit");

	/* Send the packet out. */
	module_data->ip_ptc_mem.child_pkptr = mip_sup_irdp_pkt_encapsulate
		(solicit_pkptr, home_address, subnet_bcast_addr,
IcmpC_Type_IRDP_Sol);

	/* Record some stats. */
	op_stat_write (irdp_sent_pkts_sh, 1.0);
	op_stat_write (irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, 0.0);

	/* Invoke IP to handle the packet. */
	op_pro_invoke (proc_info_struct_ptr->ip_phndl, OPC_NIL);

	/* Schedule the next transmission. */
	if (++solicit_count > 3)
		{
		solicit_interval = MipC_MN_Solicit_Min_Interval * pow (2.0,
(double) (solicit_count - 3));

		if (solicit_interval > MipC_MN_Solicit_Max_Interval)
```

```
                {
                solicit_interval = MipC_MN_Solicit_Max_Interval;
                }
            }
    else
            {
            solicit_interval = MipC_MN_Solicit_Min_Interval;
            }

    solicit_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
solicit_interval, MipC_MN_Timer_Solicit);

    FOUT;
    }

static void
mip_mn_agent_cache_update (MipT_MN_Agent_Info* agent_info, InetT_Address
    new_agent_address,
    double life_time, int pref_level, MipT_Invocation_Info*
invoke_info_ptr)
    {
    /** PURPOSE: Update Agent cache based on rule. (higher pref level or
same or lower if the current one expired) **/
    /** REQUIRES: new FA address and its lifetime and pref level.     **/
    /** EFFECTS: the cache will be updated if it matches the criteria.**/
    FIN (mip_mn_agent_cache_update (agent_info, new_agent_address,
life_time, pref_level, invoke_info_ptr));

    if ((pref_level >= agent_info->pref_level) ||
            (agent_info->lifetime < op_sim_time ()) ||
            inet_address_equal (agent_info->address, new_agent_address))
            {
            /* Update the FA cache information. */
            agent_info->address = new_agent_address;
            agent_info->lifetime = op_sim_time () + life_time;
            agent_info->pref_level = pref_level;
            agent_info->incoming_intf_ptr = ip_rte_intf_tbl_access
                (module_data, invoke_info_ptr->rte_info_ici_ptr-
>intf_recvd_index);
            }

    FOUT;
    }

static void
mip_mn_ip_pk_handle (MipT_Invocation_Info* invoke_info_ptr)
    {
    Prohandle   tmp_phndl;
    FIN (mip_mn_ip_pk_handle (invoke_info_ptr));

    /* See if there are any visiting MN with the address. */
    if (mip_sup_visitor_search_by_addr (proc_info_struct_ptr-
>node_visitor_list_lptr,
            invoke_info_ptr->rte_info_ici_ptr->dest_addr, &tmp_phndl) ==
OPC_COMPCODE_SUCCESS)
            {
            /*  will let the FA agent to handle this packet. */
```

```
			op_pro_invoke (tmp_phndl, invoke_info_ptr);
			}
	else
			{
			if ((inet_address_range_check (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr,
				&proc_info_struct_ptr->intf_info_ptr->inet_addr_range)) &&
				!inet_address_equal (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr, home_address) &&
				!inet_address_equal (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr, ha_address))
				{
				/* This address falls in the same range but not for me.
Destroy packet. */
				mip_sup_pk_cleanup (invoke_info_ptr);
				}
			else
				{
				if (current_roaming_intf)
					{
					if (current_roaming_intf == invoke_info_ptr-
>interface_ptr)
						{
						/* This is going out on the roaming interface.
Send to the current agent. */
						if (inet_address_valid (agent_address))
							invoke_info_ptr->rte_info_ici_ptr-
>next_addr = agent_address;
						}
					else
						{
						/* Unknown address. Let IP handle it
generically. */
						}
					}
				}
			}

	FOUT;
	}

static void
mip_mn_ad_packet_parse (Packet* pkptr, int* h, int* f, InetT_Address*
agent_addr,
	int* lifetime, int* pref)
	{
	IpT_Address		tmp_addr;

	/* Helper macro to parse information from the advertisement packet
received. */
	FIN (mip_mn_ad_packet_parse (...));

	/* Access the information from the packet received. */
	op_pk_nfd_access (pkptr, "H", h);
	op_pk_nfd_access (pkptr, "F", f);
	op_pk_nfd_access (pkptr, "Agent Address", &tmp_addr);
	op_pk_nfd_access (pkptr, "lifetime", lifetime);
```

```
	op_pk_nfd_access (pkptr, "Preference Level", pref);

	/* Convert the V4 address to v6 for internal reference. */
	*agent_addr = inet_address_from_ipv4_address_create (tmp_addr);

	FOUT;
	}

static void
mip_mn_agent_solicit_pk_send_adv (void) /*Me: Sending Mobile IP to join in
advance */
	{
	Packet*			solicit_pkptr_adv;
	double			solicit_interval;

	/** PURPOSE: Send the ICMP agent solicitation packet.**/
	/** REQUIRES: none.	**/
	/** EFFECTS: Packet will be given to IP to handle.**/
	FIN (mip_mn_agent_solicit_pk_send (void));

	/* Time to send out the solicitation. */

	usleep(10000000); // 10 secs
	solicit_pkptr_adv = op_pk_create_fmt ("mobile_ip_irdp_solicit");

	/* Send the packet out. */
	module_data->ip_ptc_mem.child_pkptr = mip_sup_irdp_pkt_encapsulate
		(solicit_pkptr_adv, home_address, subnet_bcast_addr,
IcmpC_Type_IRDP_Sol);

	/* Record some stats. */
	op_stat_write (irdp_sent_pkts_sh, 1.0);
	op_stat_write (irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, 0.0);

	/* Invoke IP to handle the packet. */
	op_pro_invoke (proc_info_struct_ptr->ip_phndl, OPC_NIL);

	/* Schedule the next transmission. */
	if (++solicit_count > 3)
		{
		solicit_interval = MipC_MN_Solicit_Min_Interval * pow (2.0,
(double) (solicit_count - 3));

		if (solicit_interval > MipC_MN_Solicit_Max_Interval)
			{
			solicit_interval = MipC_MN_Solicit_Max_Interval;
			}
		}
	else
		{
		solicit_interval = MipC_MN_Solicit_Min_Interval;
		}
```

```
	solicit_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
solicit_interval, MipC_MN_Timer_Solicit);

	FOUT;
	}

//me

/* End of Function Block */

/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.         */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
	void mobile_ip_mn (OP_SIM_CONTEXT_ARG_OPT);
	VosT_Obtype _op_mobile_ip_mn_init (int * init_block_ptr);
	void _op_mobile_ip_mn_diag (OP_SIM_CONTEXT_ARG_OPT);
	void _op_mobile_ip_mn_terminate (OP_SIM_CONTEXT_ARG_OPT);
	VosT_Address _op_mobile_ip_mn_alloc (VosT_Obtype, int);
	void _op_mobile_ip_mn_svar (void *, const char *, void **);


#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Process model interrupt handling procedure */

void
mobile_ip_mn (OP_SIM_CONTEXT_ARG_OPT)
	{
#if !defined (VOSD_NO_FIN)
	int _op_block_origin = 0;
#endif
	FIN_MT (mobile_ip_mn ());

		{
		/* Temporary Variables */
		Boolean		is_ip_pk = OPC_FALSE;
		Boolean		is_ad_reception = OPC_FALSE;
		Boolean		is_ha_ad_reception = OPC_FALSE;
		Boolean		is_fa_ad_reception = OPC_FALSE;
		Boolean		is_solicitation_time = OPC_FALSE;
		Boolean		is_valid_fa_candidate = OPC_FALSE;
		Boolean		is_ha_timeout = OPC_FALSE;
		Boolean		is_fa_timeout = OPC_FALSE;
		Boolean		is_timeout = OPC_FALSE;
		Boolean		is_retry = OPC_FALSE;
		Boolean		is_ha_reg_success = OPC_FALSE;
```

```
            Boolean     is_fa_reg_success = OPC_FALSE;
            Boolean     is_reg_pk = OPC_FALSE;
            Boolean     is_out_of_retries = OPC_FALSE;
            Boolean     is_invalid_reply = OPC_FALSE;
            Boolean     is_reregister = OPC_FALSE;
            Boolean     is_switch_fa = OPC_FALSE;

            Objid mip_reg_cfg_objid;
            char  ha_address_str[64];
            int         h_bit,f_bit, reply_code, lifetime_grant, tmp_reg_id,
                              irdp_lifetime, inv_mode, intrpt_code,
pref_level;
            Packet                  *irdp_pkptr, *encap_pk_ptr;
            InetT_Address           tmp_agent_address;
            MipT_Invocation_Info*   invoke_info_ptr;
            Ici*  reg_ici_ptr;
            /* End of Temporary Variables */


            FSM_ENTER ("mobile_ip_mn")

            FSM_BLOCK_SWITCH
                  {
            /*---------------------------------------------------------*/
                  /** state (Init) enter executives **/
                  FSM_STATE_ENTER_FORCED_NOLABEL (0, "Init", "mobile_ip_mn
[Init enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Init enter
execs]", state0_enter_exec)
                        {
                        /* Access the parent memory. */
                        proc_info_struct_ptr = (MipT_Proc_Info*)
op_pro_parmem_access ();

                        /* Get the type of agent I am. (MN or MR)*/
                        mip_node_type = proc_info_struct_ptr->node_type;

                        /* Access the module wide memory. */
                        module_data = (IpT_Rte_Module_Data*)
op_pro_modmem_access ();

                        /* Keep the subnet bcast address for later use. */
                        subnet_bcast_addr = inet_rte_intf_broadcast_addr_get
                              (proc_info_struct_ptr->intf_info_ptr,
InetC_Addr_Family_v4);

                        /* Parse the home agent interface address. */
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Home Agent IP Address",
                              &ha_address_str);
                        ha_address = inet_address_create (ha_address_str,
InetC_Addr_Family_v4);
                        if (inet_address_equal (ha_address,
InetI_Invalid_v4_Addr))
                              {
                              op_sim_end ("An invalid address was configured
as Home Agent.","","","");
```

```
                        }

                    /* local interface address. */
                    home_address = inet_rte_intf_addr_get
(proc_info_struct_ptr->intf_info_ptr, InetC_Addr_Family_v4);
                    agent_address = InetI_Invalid_v4_Addr;

                    /* Parse the registration related information. */
                    op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Registration Parameters",
                        &mip_reg_cfg_objid);
                    mip_reg_cfg_objid = op_topo_child (mip_reg_cfg_objid,
OPC_OBJTYPE_GENERIC, 0);

                    op_ima_obj_attr_get (mip_reg_cfg_objid, "Interval",
&(reg_info.interval));
                    op_ima_obj_attr_get (mip_reg_cfg_objid, "Retry",
&(reg_info.retry));
                    op_ima_obj_attr_get (mip_reg_cfg_objid, "Lifetime
Request", &(reg_info.req_lifetime));

                    /* Initialize state vars. */
                    reg_id = 0;
                    retry_counter = 0;
                    latest_fa_info.address = InetI_Invalid_v4_Addr;
                    latest_fa_info.lifetime = 0.0;
                    latest_fa_info.pref_level = 0;
                    current_roaming_intf = OPC_NIL;

                    /* Register some stats. */
                    tunneled_bit_rcvd_sec_sh = op_stat_reg ("Mobile
IP.Tunneled Traffic Received (bits/sec)", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    tunneled_pk_rcvd_sec_sh = op_stat_reg ("Mobile
IP.Tunneled Traffic Received (packets/sec)", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);

                    /* Find out if  need to send out solicitation when
lost. */
                    op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Agent Solicitation",
                        &solicitation);

                    /* See if I am configured on a loopback interface. */
                    loopback_intf = ip_rte_intf_is_loopback
(proc_info_struct_ptr->intf_info_ptr);

                    /* Schedule interrupt to send solicitation if
enabled. */
                    if (solicitation)
                        {
                        if (loopback_intf)
                            {
                            /*  will not solicitate if on a loopback
interface. */
                            op_sim_end ("Mobile IP currently cannot
support solicitation when configured on loopback interface.",
```

```
                                        OPC_NIL, OPC_NIL, OPC_NIL);
                                   }

                              /* Schedule an interrupt for the first
solicitation. */
                              solicit_count = 0;
                              solicit_timer_ehndl = op_intrpt_schedule_self
(mip_sup_activation_time_calculate (), MipC_MN_Timer_Solicit);

                              /* Register some stats. */
                              irdp_sent_pkts_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (packets)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                              irdp_sent_bits_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (bits)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                              g_irdp_sent_bits_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (bits/sec)", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
                              }

                        /* Find out if need to ask for simultaneous binding.
*/
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Simultaneous Binding Support", &simultaneous_binding);

                        /* Cache the node objid info. */
                        node_objid = op_topo_parent (op_id_self ());

                        /* To handle default gateway. */
                        default_gateway = OPC_FALSE;

                        /* Animation. */
                        if (op_sim_anim ())
                              {
                              /* Initialize the view. */
                              mip_sup_prepare_animation ();
                              }

                        }
                        FSM_PROFILE_SECTION_OUT (state0_enter_exec)

                  /** state (Init) exit executives **/
                  FSM_STATE_EXIT_FORCED (0, "Init", "mobile_ip_mn [Init exit
execs]")


                  /** state (Init) transition processing **/
                  FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "",
"Init", "Lost", "tr_-1", "mobile_ip_mn [Init -> Lost : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Lost) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (1, "Lost", state1_enter_exec,
"mobile_ip_mn [Lost enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost enter
execs]", state1_enter_exec)
                        {
                        /* Update the status for debugging. */
```

```
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Lost,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state1_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (3,"mobile_ip_mn")

                  /** state (Lost) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (1, "Lost", "mobile_ip_mn [Lost
exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost exit
execs]", state1_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_solicitation_time = OPC_TRUE;

                              if (op_intrpt_code () == MipC_MN_Timer_Solicit)
                                    {
                                    mip_mn_agent_solicit_pk_send ();
                                    }
                              }
                        else
                              {
                              /* See if  are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*  have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*  have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                        /*   do not want to deal with this packet. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);

                                                      is_ip_pk = OPC_TRUE;
                                                      break;
                                                      }

                        /* This must be an advertisement from an agent. */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
```

```
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                      {
                                                      if (loopback_intf)
                                                            {
                                    /* This is not supported. */
                                                            op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                OPC_NIL, OPC_NIL, OPC_NIL);
                                                            }

                                          /* Ad from my home agent. */
                                                is_ad_reception = OPC_TRUE;

                        /* Initialize counter before start reg process. */
                                                      retry_counter = 0;

                  /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                            (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Deregister with HA. */
                                                      mip_mn_register (0,
latest_ha_info, OPC_TRUE);

                                          /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                      }
                                                else
                                                      {
                                                      if (f_bit)
                                                            {
                                                /* A foreign agent ad. */
                                                is_ad_reception = OPC_TRUE;

                        /* Initialize counter before start reg process. */
                                                retry_counter = 0;

                        /* Update the latest fa info structure for later. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                        (double) irdp_lifetime, pref_level, invoke_info_ptr);

                              /* Register with HA using the FA address. */
      mip_mn_register (reg_info.req_lifetime, latest_fa_info, OPC_FALSE);

                                    /* Update timer for HA timeout. */
```

```
      mip_mn_agent_timer_update (latest_fa_info.lifetime);
                                                               }
                                                         else
                                                               {
/* cannot do anything with this agent who is only HA for other group. */
                                                   is_ip_pk = OPC_TRUE;
                                                               }
                                                         }
                                       /* Clean up solicitation if any. */
      if (solicitation && is_ad_reception)
                                                         {
      op_ev_cancel_if_pending (solicit_timer_ehndl);
                                                         }

                                                   /* Clean up. */
      mip_sup_pk_cleanup (invoke_info_ptr);
      op_pk_destroy (irdp_pkptr);

      break;
                                                   }

                                             case MipC_Invoke_Type_Tunnel_Check:
                                                   {
                                                   is_ip_pk = OPC_TRUE;

                  /*   should try to decapsulte packet if in MR mode. */
                              if (mip_node_type == MipC_Node_Type_MR)
                                                         {
      /*   will check first if this packet is tunneling other IP packet */
                                                         if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                               ==
OPC_COMPCODE_SUCCESS)
                                                               {
                                             /* Sanity check on the packet */

                              /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                        /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                  /* Let IP caller know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                               }
                                                         }

                                                   break;
                                                   }

                                             case MipC_Invoke_Type_IP_Datagram:
                                                   {
                                                   is_ip_pk = OPC_TRUE;
```

```
/* The destination address that is going out on this interface when in MR
mode should be forwarded to either HA or FA.  Whoever already registered or
trying to. */
                                    if (mip_node_type == MipC_Node_Type_MR)
                                                      {
                              /* Handle packet if I am a MR. */
                                    mip_mn_ip_pk_handle (invoke_info_ptr);
                                                      }
                                                else
                                                      {
                        /* Unknown address.  Send to the current agent. */
                                    if (inet_address_valid (agent_address))
                                                            invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                      }

                                                break;
                                                }
                                          } /* switch (invoke_info_ptr-
>invocation_type) */
                                    }
                              else
                                    {
                                    /* Registration arrival. */
                                    op_ici_destroy (op_intrpt_ici ());
                                    is_reg_pk = OPC_TRUE;
                                    }
                              }
                        }
                        FSM_PROFILE_SECTION_OUT (state1_exit_exec)

                  /** state (Lost) transition processing **/
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost trans
conditions]", state1_trans_conds)
                  FSM_INIT_COND (IP_PK)
                  FSM_TEST_COND (SOLICITATION_TIME)
                  FSM_TEST_COND (REG_PK)
                  FSM_TEST_COND (AD_RECEPTION)
                  FSM_TEST_LOGIC ("Lost")
                  FSM_PROFILE_SECTION_OUT (state1_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"IP_PK", "", "Lost", "Lost", "tr_8", "mobile_ip_mn [Lost -> Lost : IP_PK /
]")
                        FSM_CASE_TRANSIT (1, 1, state1_enter_exec, ;,
"SOLICITATION_TIME", "", "Lost", "Lost", "tr_32", "mobile_ip_mn [Lost -> Lost
: SOLICITATION_TIME / ]")
                        FSM_CASE_TRANSIT (2, 1, state1_enter_exec, ;,
"REG_PK", "", "Lost", "Lost", "tr_36", "mobile_ip_mn [Lost -> Lost : REG_PK /
]")
                        FSM_CASE_TRANSIT (3, 4, state4_enter_exec, ;,
"AD_RECEPTION", "", "Lost", "Pending registration", "tr_1", "mobile_ip_mn
[Lost -> Pending registration : AD_RECEPTION / ]")
                        }
            /*---------------------------------------------------------*/
```

```
                  /** state (At home) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (2, "At home", state2_enter_exec,
"mobile_ip_mn [At home enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home enter
execs]", state2_enter_exec)
                        {
                        /* Update the status for debugging. */
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Home,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state2_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (5,"mobile_ip_mn")


                  /** state (At home) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (2, "At home", "mobile_ip_mn [At
home exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home exit
execs]", state2_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_ha_timeout = OPC_TRUE;

                              if (MIP_TRACE)
                                    {
                                    op_prg_odb_print_major ("Trying
reregister for timeout occurred.", OPC_NIL);
                                    }
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*   have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*   have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                        /*   do not want to deal with this packet. */
```

```
                                                        mip_sup_pk_cleanup
(invoke_info_ptr);

                                                        is_ip_pk = OPC_TRUE;
                                                        break;
                                                        }

                        /* This must be an advertisement from an agent. */
                                                  is_ad_reception = OPC_TRUE;

                  /*   have an agent advertisement.  But which kind thou? */
                                                  op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                                  mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                  if
(inet_address_equal(tmp_agent_address, ha_address))
                                                        {
                                                        if (loopback_intf)
                                                              {
                                    /* This is not supported. */
                                                              op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                                    OPC_NIL,
OPC_NIL, OPC_NIL);
                                                              }

                                          /* Ad from my home agent. */

                        /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                              (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                        }
                                                  else
                                                        {
                                                        if (f_bit)
                                                              {
                                                  /* A foreign agent ad. */

                        /* Update the latest fa info structure for later. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                                    (double)
irdp_lifetime, pref_level, invoke_info_ptr);
                                                              }
                                                        else
                                                              {
```

```
/* cannot do anything with this agent who is only HA for other group. */
                                                            }
                                                      }

                                                /* Clean up. */
                                                mip_sup_pk_cleanup
(invoke_info_ptr);
                                                op_pk_destroy (irdp_pkptr);

                                                break;
                                                }

                                           case
MipC_Invoke_Type_Tunnel_Check:
                                                {
                                                is_ip_pk = OPC_TRUE;

                  /*   should try to decapsulte packet if in MR mode. */
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                      {
      /*   will check first if this packet is tunneling other IP packet */
                                                      if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                            ==
OPC_COMPCODE_SUCCESS)
                                                            {
                                    /* Sanity check on the packet */

                              /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

      /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

            /* Let IP caller know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                            }
                                                      }

                                                break;
                                                }

                                          case MipC_Invoke_Type_IP_Datagram:
                                                {
                                                is_ip_pk = OPC_TRUE;

                                                /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                    if (mip_node_type == MipC_Node_Type_MR)
                                                      {
                                          /* Handle packet if I am a MR. */
                                    mip_mn_ip_pk_handle (invoke_info_ptr);
```

```
                                                  }
                                          else
                                                  {
                        /* Unknown address.  Send to the current agent. */
            invoke_info_ptr->rte_info_ici_ptr->next_addr = agent_address;
                                                  }

                                          break;
                                          }
                                    }/* switch (invoke_info_ptr-
>invocation_type) */
                              }
                        else
                              {
                              /* Registration arrival. */
                              op_ici_destroy (op_intrpt_ici ());
                              is_reg_pk = OPC_TRUE;
                              }
                        }
                  }
                  FSM_PROFILE_SECTION_OUT (state2_exit_exec)

            /** state (At home) transition processing **/
            FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home trans
conditions]", state2_trans_conds)
            FSM_INIT_COND (HA_TIMEOUT)
            FSM_TEST_COND (IP_PK)
            FSM_TEST_COND (AD_RECEPTION)
            FSM_TEST_COND (REG_PK)
            FSM_TEST_LOGIC ("At home")
            FSM_PROFILE_SECTION_OUT (state2_trans_conds)

            FSM_TRANSIT_SWITCH
                  {
                  FSM_CASE_TRANSIT (0, 5, state5_enter_exec, ;,
"HA_TIMEOUT", "", "At home", "Check FA cache", "tr_5", "mobile_ip_mn [At home
-> Check FA cache : HA_TIMEOUT / ]")
                  FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;,
"IP_PK", "", "At home", "At home", "tr_9", "mobile_ip_mn [At home -> At home
: IP_PK / ]")
                  FSM_CASE_TRANSIT (2, 2, state2_enter_exec, ;,
"AD_RECEPTION", "", "At home", "At home", "tr_26", "mobile_ip_mn [At home ->
At home : AD_RECEPTION / ]")
                  FSM_CASE_TRANSIT (3, 2, state2_enter_exec, ;,
"REG_PK", "", "At home", "At home", "tr_35", "mobile_ip_mn [At home -> At
home : REG_PK / ]")
                  }
      /*---------------------------------------------------------*/

            /** state (Away) enter executives **/
            FSM_STATE_ENTER_UNFORCED (3, "Away", state3_enter_exec,
"mobile_ip_mn [Away enter execs]")
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away enter
execs]", state3_enter_exec)
                  {
                  /* Update the status for debugging. */
```

```
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Foreign,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state3_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (7,"mobile_ip_mn")

                  /** state (Away) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (3, "Away", "mobile_ip_mn [Away
exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away exit
execs]", state3_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              intrpt_code = op_intrpt_code ();
                              switch (intrpt_code)
                                    {
                                    case MipC_MN_Timer_Agent:
                                          {
                                          is_fa_timeout = OPC_TRUE;
                                          break;
                                          }

                                  case MipC_MN_Timer_Rereg:
                                          {
                                          is_reregister = OPC_TRUE;
                                          break;
                                          }
                                    }

                              if (MIP_TRACE)
                                    {
                                    op_prg_odb_print_major ("Trying
reregister for timeout occurred.", OPC_NIL);
                                    }
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                                    /*   have an invocation from IP.  What
kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
```

```
                  /*   have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                        /*   do not want to deal with this packet. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);

                                                      is_ip_pk = OPC_TRUE;
                                                      break;
                                                      }

                        /* This must be an advertisement from an agent. */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffices. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                      {
                                                      if (loopback_intf)
                                                            {
                                          /* This is not supported. */
                                                            op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                                  OPC_NIL,
OPC_NIL, OPC_NIL);
                                                            }

                                    /* Ad from my home agent. */
                                                      is_ha_ad_reception =
OPC_TRUE;

                        /* Initialize counter before start reg process. */
                                                      retry_counter = 0;

                        /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                            (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Deregister with HA. */
                                                      mip_mn_register (0,
latest_ha_info, OPC_TRUE);

                                          /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                      }
                                                else
                                                      {
                                                      if (f_bit)
```

```
                                                        {
                                        /* A foreign agent ad. */
                                                        if
(current_agent_pref_level < pref_level)
                                                              {
            /* New FA agent advertising has higher preference.  Switch. */

      is_switch_fa = OPC_TRUE;
                                                              }
                                                        else
                                                              {

      is_fa_ad_reception = OPC_TRUE;
                                                              }

                                  /* Update the FA cache information. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                              (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                              /* Check to see if it is the same agent. */
                                                        if
(inet_address_equal (agent_address, tmp_agent_address))
                                                              {
                                    /* Update timer for FA timeout. */

      mip_mn_agent_timer_update (op_sim_time () + (double) irdp_lifetime);
                                                              }
                                                        }
                                                  else
                                                        {
/* cannot do anything with this agent who is only HA for other group. */
                                                        is_ip_pk =
OPC_TRUE;

                                                        }
                                                  }

                                        /* Clean up. */
                                        mip_sup_pk_cleanup
(invoke_info_ptr);
                                        op_pk_destroy (irdp_pkptr);

                                        break;
                                        }

                                  case MipC_Invoke_Type_Tunnel_Check:
                                        {
                                        is_ip_pk = OPC_TRUE;

                  /*   should try to decapsulte packet if in MR mode. */
                                        if (mip_node_type ==
MipC_Node_Type_MR)
                                                  {
      /*   will check first if this packet is tunneling other IP packet */
                                                  if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
```

```
                                                            ==
OPC_COMPCODE_SUCCESS)
                                                            {
                                          /* Sanity check on the packet */

                              /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                        /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                  /* Let IP caller know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                            }
                                                      }

                                                break;
                                                }

                                          case MipC_Invoke_Type_IP_Datagram:
                                                {
                                                is_ip_pk = OPC_TRUE;

                                                /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                    if (mip_node_type == MipC_Node_Type_MR)
                                                      {
                                          /* Handle packet if I am a MR. */
                                    mip_mn_ip_pk_handle (invoke_info_ptr);
                                                      }
                                                else
                                                      {
                        /* Unknown address.  Send to the current agent. */
                                                      invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                      }

                                                break;
                                                }
                                          }/* switch (invoke_info_ptr-
>invocation_type) */
                                    }
                              else
                                    {
                                    /* Reg packet arrival. */
                                    op_ici_destroy (op_intrpt_ici ());
                                    is_reg_pk = OPC_TRUE;
                                    }
                              }
                        }
                        FSM_PROFILE_SECTION_OUT (state3_exit_exec)
```

```
                  /** state (Away) transition processing **/
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away trans
conditions]", state3_trans_conds)
                  FSM_INIT_COND (FA_TIMEOUT || REREGISTER || SWITCH_FA)
                  FSM_TEST_COND (IP_PK)
                  FSM_TEST_COND (FA_AD_RECEPTION)
                  FSM_TEST_COND (REG_PK)
                  FSM_TEST_COND (HA_AD_RECEPTION)
                  FSM_TEST_LOGIC ("Away")
                  FSM_PROFILE_SECTION_OUT (state3_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 5, state5_enter_exec, ;,
"FA_TIMEOUT || REREGISTER || SWITCH_FA", "", "Away", "Check FA cache",
"tr_4", "mobile_ip_mn [Away -> Check FA cache : FA_TIMEOUT || REREGISTER ||
SWITCH_FA / ]")
                        FSM_CASE_TRANSIT (1, 3, state3_enter_exec, ;,
"IP_PK", "", "Away", "Away", "tr_10", "mobile_ip_mn [Away -> Away : IP_PK /
]")
                        FSM_CASE_TRANSIT (2, 3, state3_enter_exec, ;,
"FA_AD_RECEPTION", "", "Away", "Away", "tr_27", "mobile_ip_mn [Away -> Away :
FA_AD_RECEPTION / ]")
                        FSM_CASE_TRANSIT (3, 3, state3_enter_exec, ;,
"REG_PK", "", "Away", "Away", "tr_37", "mobile_ip_mn [Away -> Away : REG_PK /
]")
                        FSM_CASE_TRANSIT (4, 4, state4_enter_exec, ;,
"HA_AD_RECEPTION", "", "Away", "Pending registration", "tr_45", "mobile_ip_mn
[Away -> Pending registration : HA_AD_RECEPTION / ]")
                        }
            /*---------------------------------------------------------*/

                  /** state (Pending registration) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (4, "Pending registration",
state4_enter_exec, "mobile_ip_mn [Pending registration enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending
registration enter execs]", state4_enter_exec)
                        {
                        /* Update the status for debugging. */
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Pending,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state4_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (9,"mobile_ip_mn")


                  /** state (Pending registration) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (4, "Pending registration",
"mobile_ip_mn [Pending registration exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending
registration exit execs]", state4_exit_exec)
                        {
                        /* Who invoked me? */
```

```
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_timeout = OPC_TRUE;
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*   have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*   have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                              /*   do not want to deal with this packet. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);

                                                      is_ip_pk = OPC_TRUE;
                                                      break;
                                                      }

                        /* This must be an advertisement from an agent. */
                                                is_ad_reception = OPC_TRUE;

                  /*   have an agent advertisement.  But which kind thou? */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                      {
                                                      if (loopback_intf)
                                                            {
                                          /* This is not supported. */
                                                            op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                                  OPC_NIL,
OPC_NIL, OPC_NIL);
                                                            }

                        /* Update the latest ha info structure for later. */
```

```
      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                                  (double)
irdp_lifetime, pref_level, invoke_info_ptr);

      /* Check if   are currently trying to register directly with HA. */
                                                            if (!direct_reg)
                                                                  {

      op_ev_cancel_if_pending (reg_retry_timer_ehndl);
                                                      /* Deregister with HA. */
                                                                  mip_mn_register
(0, latest_ha_info, OPC_TRUE);
                                                                  }

                                                /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                            }
                                                      else
                                                            {
                                                            if (f_bit)
                                                                  {
                                          /* Update the FA cache information. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                                        (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Check to see if it is the same agent. */
                                                                  if
(inet_address_equal (agent_address, tmp_agent_address))
                                                                        {
                                                /* Update timer for FA timeout. */

      mip_mn_agent_timer_update (op_sim_time () + (double) irdp_lifetime);
                                                                        }
                                                                  }
                                                            else
                                                                  {
/* cannot do anything with this agent who is only HA for other group. */
                                                                  }
                                                            }

                                                      /* Clean up. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);
                                                      op_pk_destroy (irdp_pkptr);

                                                      break;
                                                      }

                                                case MipC_Invoke_Type_Tunnel_Check:
                                                      {
                                                      is_ip_pk = OPC_TRUE;

                        /*   should try to decapsulte packet if in MR mode. */
```

```
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                      {
      /*   will check first if this packet is tunneling other IP packet */
                                                      if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                            ==
OPC_COMPCODE_SUCCESS)
                                                            {
                                          /* Sanity check on the packet */

                              /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                  /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                                                            /* Let IP caller
know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                            }
                                                      }

                                                break;
                                                }

                                          case MipC_Invoke_Type_IP_Datagram:
                                                {
                                                is_ip_pk = OPC_TRUE;

                                                /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                      {
                  /* Handle packet if I am a MR. */
                                                      mip_mn_ip_pk_handle
(invoke_info_ptr);
                                                      }
                                                else
                                                      {
                        /* Unknown address.  Send to the current agent. */
                                                      invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                      }

                                                break;
                                                }
                                          } /* switch (invoke_info_ptr-
>invocation_type) */
                                    }
                              else
                                    {
```

```
                                /* Registration packet arrival. */
                                is_reg_pk = OPC_TRUE;
                                }
                          }
                    }
                    FSM_PROFILE_SECTION_OUT (state4_exit_exec)

              /** state (Pending registration) transition processing **/
              FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending registration
trans conditions]", state4_trans_conds)
              FSM_INIT_COND ( REG_PK)
              FSM_TEST_COND (TIMEOUT)
              FSM_TEST_COND (AD_RECEPTION)
              FSM_TEST_COND (IP_PK)
              FSM_TEST_LOGIC ("Pending registration")
              FSM_PROFILE_SECTION_OUT (state4_trans_conds)

              FSM_TRANSIT_SWITCH
                    {
                    FSM_CASE_TRANSIT (0, 6, state6_enter_exec, ;, "
REG_PK", "", "Pending registration", "Handle registration", "tr_15",
"mobile_ip_mn [Pending registration -> Handle registration :  REG_PK / ]")
                    FSM_CASE_TRANSIT (1, 7, state7_enter_exec, ;,
"TIMEOUT", "", "Pending registration", "Handle timeout", "tr_19",
"mobile_ip_mn [Pending registration -> Handle timeout : TIMEOUT / ]")
                    FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;,
"AD_RECEPTION", "", "Pending registration", "Pending registration", "tr_28",
"mobile_ip_mn [Pending registration -> Pending registration : AD_RECEPTION /
]")
                    FSM_CASE_TRANSIT (3, 4, state4_enter_exec, ;,
"IP_PK", "", "Pending registration", "Pending registration", "tr_11",
"mobile_ip_mn [Pending registration -> Pending registration : IP_PK / ]")
                    }
        /*---------------------------------------------------------*/

              /** state (Check FA cache) enter executives **/
              FSM_STATE_ENTER_FORCED (5, "Check FA cache",
state5_enter_exec, "mobile_ip_mn [Check FA cache enter execs]")
                    FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Check FA cache
enter execs]", state5_enter_exec)
                    {
                    /* See if   can use the cached FA information. */
                    if (inet_address_valid (latest_fa_info.address))
                          {
                          if (latest_fa_info.lifetime > op_sim_time () )
                                {
                    /* Initialize counter before start reg process. */
                                retry_counter = 0;

                    /* Register with the latest advertised FA. */
                                mip_mn_register (reg_info.req_lifetime,
                                      latest_fa_info, OPC_FALSE);

                                /* Update the timer timeout. */
                                if (!HA_TIMEOUT && !FA_TIMEOUT)
                                      {
                    mip_mn_agent_timer_update (latest_fa_info.lifetime);
```

```
                                        }
                                else
                                        {
                                /* cannot cancel the current event. */
        agent_timer_ehndl = op_intrpt_schedule_self (latest_fa_info.lifetime,
                                                MipC_MN_Timer_Agent);
                                        }

                                is_valid_fa_candidate = OPC_TRUE;
                                }
                        }

                if (!is_valid_fa_candidate)
                        {
                        if (solicitation)
                                {
                /* Schedule an interrupt to send solicitation packet. */
            op_intrpt_schedule_self (op_sim_time (), MipC_MN_Timer_Solicit);
                                }

                /* Cancel reregister timer first. */
                        op_ev_cancel_if_pending (reregister_timer_ehndl);

                        if (op_sim_anim ())
                                {
                                /* Erase the existing tunnel. */
                                mip_sup_draw_tunnel (node_objid,
ha_address, ha_address, ((mip_node_type == MipC_Node_Type_MR) ? OPC_TRUE :
OPC_FALSE));
                                }
                        }

                }
                FSM_PROFILE_SECTION_OUT (state5_enter_exec)

            /** state (Check FA cache) exit executives **/
            FSM_STATE_EXIT_FORCED (5, "Check FA cache", "mobile_ip_mn
[Check FA cache exit execs]")

            /** state (Check FA cache) transition processing **/
            FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Check FA cache trans
conditions]", state5_trans_conds)
            FSM_INIT_COND (!VALID_FA_CANDIDATE)
            FSM_TEST_COND (VALID_FA_CANDIDATE)
            FSM_TEST_LOGIC ("Check FA cache")
            FSM_PROFILE_SECTION_OUT (state5_trans_conds)

            FSM_TRANSIT_SWITCH
                {
                FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"!VALID_FA_CANDIDATE", "", "Check FA cache", "Lost", "tr_6", "mobile_ip_mn
[Check FA cache -> Lost : !VALID_FA_CANDIDATE / ]")
                FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;,
"VALID_FA_CANDIDATE", "", "Check FA cache", "Pending registration", "tr_7",
"mobile_ip_mn [Check FA cache -> Pending registration : VALID_FA_CANDIDATE /
]")
                }
```

```
		/*-----------------------------------------------------------*/

			/** state (Handle registration) enter executives **/
			FSM_STATE_ENTER_FORCED (6, "Handle registration",
state6_enter_exec, "mobile_ip_mn [Handle registration enter execs]")
				FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Handle
registration enter execs]", state6_enter_exec)
				{
				/* Access ICI from mobile ip module. */
				reg_ici_ptr = op_intrpt_ici ();

				/* Get the ICI values. */
				op_ici_attr_get (reg_ici_ptr, "reply_code",
&reply_code);
				op_ici_attr_get (reg_ici_ptr, "lifetime_grant",
&lifetime_grant);
				op_ici_attr_get (reg_ici_ptr, "identification",
&tmp_reg_id);

				if (tmp_reg_id == reg_id)
					{
					/* Did it go through? */
					if ((reply_code == MipC_Reg_Reply_Code_Accept)
||
						(reply_code ==
MipC_Reg_Reply_Code_Accept_No_Simultaneous_Binding))
						{
						/* Cancel the retry timer first. */
						op_ev_cancel_if_pending
(reg_retry_timer_ehndl);

						if (direct_reg)
							{
							is_ha_reg_success = OPC_TRUE;

							if (MIP_TRACE)
								{
								op_prg_odb_print_major
("Registering directly with HA successful.", OPC_NIL);
								}

							if (op_sim_anim ())
								{
								/* Draw tunnel to the HA. */
								mip_sup_draw_tunnel
(node_objid, ha_address, agent_address, ((mip_node_type == MipC_Node_Type_MR)
? OPC_TRUE : OPC_FALSE));
								}
							}
						else
							{
							is_fa_reg_success = OPC_TRUE;

						/* Cancel reregister timer first. */
							op_ev_cancel_if_pending
(reregister_timer_ehndl);
```

```
                                    /* Update the lifetime. */
                                    time_to_reregister = op_sim_time ()
+ (double) lifetime_grant - MipC_MN_Rereg_Buffer;
                                    reregister_timer_ehndl =
op_intrpt_schedule_self (time_to_reregister,
                                          MipC_MN_Timer_Rereg);

                                    if (MIP_TRACE)
                                          {
                                          op_prg_odb_print_major
("Registering via a FA successful.", OPC_NIL);
                                          }

                                    if (op_sim_anim ())
                                          {
                              /* Draw tunnel to the HA through FA. */
                                          mip_sup_draw_tunnel
(node_objid, ha_address, agent_address, ((mip_node_type == MipC_Node_Type_MR)
? OPC_TRUE : OPC_FALSE));
                                          }
                                    }
                              }
                        else
                              {
                              /* have to retry. */
                              if (retry_counter <=  reg_info.retry)
                                    {
                                    is_retry = OPC_TRUE;
                                    }
                              else
                                    {
                                    op_ev_cancel_if_pending
(reg_retry_timer_ehndl);
                                    is_out_of_retries = OPC_TRUE;
                                    }
                              }
                        }
                  else
                        {
                        /* Identification mismatch. */
                        is_invalid_reply = OPC_TRUE;
                        }

                  /* Clean up. */
                  op_ici_destroy (reg_ici_ptr);
                  }
                  FSM_PROFILE_SECTION_OUT (state6_enter_exec)

            /** state (Handle registration) exit executives **/
            FSM_STATE_EXIT_FORCED (6, "Handle registration",
"mobile_ip_mn [Handle registration exit execs]")

            /** state (Handle registration) transition processing **/
            FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Handle registration
trans conditions]", state6_trans_conds)
            FSM_INIT_COND (FA_REG_SUCCESS)
            FSM_TEST_COND (OUT_OF_RETRIES)
```

```
                  FSM_TEST_COND (RETRY || INVALID_REPLY)
                  FSM_TEST_COND (HA_REG_SUCCESS)
                  FSM_TEST_LOGIC ("Handle registration")
                  FSM_PROFILE_SECTION_OUT (state6_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;,
"FA_REG_SUCCESS", "", "Handle registration", "Away", "tr_2", "mobile_ip_mn
[Handle registration -> Away : FA_REG_SUCCESS / ]")
                        FSM_CASE_TRANSIT (1, 5, state5_enter_exec, ;,
"OUT_OF_RETRIES", "", "Handle registration", "Check FA cache", "tr_12",
"mobile_ip_mn [Handle registration -> Check FA cache : OUT_OF_RETRIES / ]")
                        FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;, "RETRY
|| INVALID_REPLY", "", "Handle registration", "Pending registration",
"tr_18", "mobile_ip_mn [Handle registration -> Pending registration : RETRY
|| INVALID_REPLY / ]")
                        FSM_CASE_TRANSIT (3, 2, state2_enter_exec, ;,
"HA_REG_SUCCESS", "", "Handle registration", "At home", "tr_44",
"mobile_ip_mn [Handle registration -> At home : HA_REG_SUCCESS / ]")
                        }
            /*---------------------------------------------------------*/
```

REFERENCES

[1] Media Post, http://www.mediapost.com/publications/?fa=Articles.showArticle&art_aid=122460, 2011.

[2] Cisco (2013). IP Multicast Technology Overview., *IP Multicast: PIM Configuration Guide, Cisco IOS XE Release 3S* (pp.1-23). Retrieved from http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipmulti_pim/configuration/xe-3s/imc-pim-xe-3s-book/imc_tech_oview.html

[3] S. Deering, "Host extensions for IP multicasting," RFC1112, Internet Engineering Task Force, August 1989.

[4] Host extensions for IP multicasting, RFC 988, available from: http://www.rfc-editor.org/rfc/rfc988.txt, cited on 20 October 2013

[5] Internet Group Management Protocol, Version3, RFC 2236, available from: http://www.rfc-editor.org/rfc/rfc3376.txt, cited on 20 October 2013

[6] Using Internet Group Management Protocol Version3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast, RFC 4604, available from: http://www.rfc-editor.org/rfc/rfc4604.txt, cited on 20 October 2013

[7] H3C - Technical Support & Documents - 09 - IP Multicast Volume, http://www.h3c.com/portal/Technical_Support___Documents/Technical_Documents/Routers/H3C_SR6600_Series_Routers/Configuration/Operation_Manual/H3C_SR6600_OM-Release_2315(V1.09)/09/201011/701616_1285_0.htm

[8] C. Wen, C. Wu, and W. Lee, "A context-aware handover scheme and all-ip mobile multicast service for heterogeneous wireless networks," IEEE Proc. International Conference Ultra Modern Telecommunications & Workshops (ICUMT'09), pp. 1-7, Oct. 2009.

[9] D. Minoli, IP Multicast with Applications to IPTV and Mobile DVB-H. John Wiley and Sons, Canada: 2008.

[10] MBone: Multicasting Tomorrow's Internet, http://www.savetz.com/mbone/ch3_6.html

[11] CRU, Global FTTX developments fibre reaching closer to the home, News: ICF, issue 59, September 2007.

[12] A. Benslimane, Multimedia Multicast on the Internet, ISTE Ltd, UK: 2007.

[13] Wes, S. (2006). Video over IP: A Practical Guide to Technology and Applications. Focal Press.

[14] Chalmers, R. C., & Almeroth, K. C. (2000). *Developing a Multicast Metric*. In Proceedings of IEEE Globecom 2000, San Francisco, California, USA.

[15] Chuang, J., & Sirbu, M. (1998). *Pricing Multicast Commuincation: A Cost Based Approach*. In Proceeding of INET'98, Geneva, Switzerland.

[16] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, Internet Engineering Task Force, April 1998.

[17] A. Mihailovic, M. Shabeer, and A.H. Aghvami, "Multicast for mobility protocol (mmp) for emerging internet networks," Proc. 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication (PIMRC2000), pp. 327-333, Sep. 2000.

[18] T. Schmidt, M. Waehlisch and G. Fairhurst, "Multicast Mobility in Mobile IP version 6 (MIPv6)," RFC5757, February 2010.

[19] J. Guan, Y. Qin, S. Gao, and H. Zhang, "The performance analysis of multicast in proxy mobile ipv6," Proc. ICCTA2009, 2009.

[20] R. Rummler, A. Gluhak, and A.H. Aghvami, Multicast in Third-Generation Mobile Networks: Services, Mechanisms and Performance. John Wiley and Sons, UK: 2009.

[21] H. Gossain, C. de Morais Cordeiro and P. Agrawal, "Multicast: Wired to Wireless," IEEE Communications Magazine, June 2002.

[22] D. Waitzman, C. Partridge and S Deering, “Distance Vector Multicast Routing Protocol,” RFC 1075, Internet Engineering Task Force, November 1988.

[23] A. Ballaedie, “Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification,” RFC 2189, Internet Engineering Task Force, September 1997.

[24] K.Chi, C. Tseng and T. Huang, “IP Multicast Support in Mobile Interworks,” Journal of Computers: 1-21, 2006.

[25] P. Karn, C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig, J. Mahdavi, G. Montenegro, J. Touch and L. Wood, “Advice for Internet SubnetworkDesigners,” RFC 3819, Internet Engineering Task Force, July 2004.

[26] T. C. Schmidt, M. Wahlisch, “Roaming Real-Time Applications – Mobility Services in IPv6 Networks,” Proceeding TERENA Networking Conference, 2003.

[27] A. Adams, J. Nicholas and W. Siadak, “Protocol Independent Multicast – Dense Mode (PIM-DM): Protocol Specification (revised),” RFC 3973, Internet Engineering Task Force, January 2005.

[28] F. Siddiqui and S. Zeadally, “Mobility management across hybrid wireless networks: trends and challenges,” Computer Communication., vol. 29, pp. 1363-1385, 2006.

[29] J. Korhonen, U. N. Teliasonera, and V. D. Azaire, “Service Selection for Mobile IPv6,” RFC5149, February 2008.

[30] D. Johnson, C. Perkins and J. Arkko, “Mobility support in IPv6,” RFC 3775, June 2004.

[31] T. C. Schmidt, M. Wahlisch, “Performance Analysis of Multicast Mobility in a Hierarchical Mobile IP Proxy Environment,” the TERENA Networking Conference, 2004.

[32] H. Soliman, Mobile IPv6: Mobility in a Wireless Internet, Addison-Wesley, USA: 2004.

[33] V. Chikarmane, C. L. Williamson, R. B. Bunt and W. L. Mackrell, "Multicast support for mobile hosts using Mobile IP: Design issues and proposed architecture," Mobile Networks and Applications Baltzer Science Publishers BV 3: 365-379, 1998.

[34] B. Balavenkatesh, K. A. B. Krishnan, S. Ramkumar, V. B. Hency, and D. Sridharan, "Enhancement of qos of voip over heterogeneous networks by improving handover speed and throughput," Proc. IEEE International Conference on Advances in Computing, Control, and Telecommincation Technologies, pp. 840-844, 2009.

[35] A. Conta and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol version 6 (IPv6) Specification," RFC2463, Dec 1998.

[36] S. Jeon, N. Kang, and Y. Kim, "Mobility management based on proxy mobile ipv6 for multicasting services in home networks," IEEE Trans. On Consumer Electronics, pp. 1227-1232, Jun. 2009.

[37] W. Run-liu, and Y. Yun-Hui, "Mobile IP Multicast Routing Algorithm by Using Super Node Set," 2012 Fourth International Conference on Computational and Information Sciences (ICCIS), August 2012.

[38] H. Holbrook, B. Cain and B. Haberman, "Using IGMPv3 and MLDv2 for Source-Specific Multicast," Inter Draft, October 2003.

[39] S. Figueiredo, S. Jeon and R.L. Aguiar, "Empowering IP Multicast for Multimedia Delivery over Heterogeneous Mobile Wireless Networks," 2014 IEEE Conference on Computer Communication Workshops (INFOCOM WKSHPS), May 2014.

[40] I. Romdhani, M. Kellil, and H. Lach, "IP mobile multicast," IEEE communications surveys, volume6, No.1, 2004.

[41] P. Savola, "IPv6 multicast deployment issues," Internet draft, February 2004.

[42] Y. LI, W. Chen, L. Su, D. Jin, and L. Zeng, "Proxy mobile ipv6 based multicast listener mobility architecture," Proc. IEEE Wireless Communications and Networking Conference (WCNC2009), IEEE press, Apr. 2009.

[43] S. Yang and W. Chen, “Sip multicast-based mobile quality-of-service support over heterogeneous ip multimedia subsystem,” IEEE trans. Mobile Computing., vol. 7, pp. 1297-1310, November 2008.

[44] S. Mohanty, “A new architecture for 3g and wlan integration and inter-system handover management,” Wireless Netw., vol. 12, pp. 733-745, 2006.

[45] T. Melia, Ed, "Mobility Services Transport: Problem Statement", RFC 5164, Internet Engineering Task Force, March 2008.

[46] Y. Kim, and S. Han, “Proxy Mobile IP Extension for Mobile Multimedia Multicast Services,” 6th IEEE Consumer Communications and Networking Conference (CCNC), January 2009.

[47] E. Stevens-Navarro, V. W.S. Wong, and Y. Lin, “A vertical handover decision algorithm for heterogeneous wireless networks,” Proc. IEEE Wireless Communication and Networking Conference (WCNC’07), IEEE Press, Mar. 2007.

[48] S. Park, Y. Won, J. Kim, I. Jung, S. Jo, W. Ryu and J. Chae, “A Network-Based Mobile Multicast Framework for Heterogeneous IP-Based Network,” 2013 International Conference on Information Science and Applications (ICISA), June 2013.

[49] C. Chen, S. Wang, Y. Tsai, and H. Chen, “A framework of multicast key agreement for distributed user on 3g-wlan,” Proc. IEEE 5th International Joint Conference on INC, IMS and IDC, pp. 2062-2068, 2009.

[50] J. Lee and T. Ernst, “Fast PMIPv6 Multicast Handover Procedure for Mobility-Unaware Mobile Nodes,” 2011 IEEE 73rd Vehicular Technology Conference (VTC Spring), 2011.

[51] S. Mohanty, “A new architecture for 3g and wlan integration and inter-system handover management,” Wireless Netw., vol. 12, pp. 733-745, 2006.

[52] S. Yang and W. Chen, “Sip multicast-based mobile quality-of-service support over heterogeneous ip multimedia subsystem,” IEEE trans. Mobile Computing., vol. 7, pp. 1297-1310, November 2008.

[53] Y.Y. An, B.H. Yae, K.W. Lee, Y.Z. Cho, and W. Y. Jung, “Reduction of handover latency using MIH services in MIPv6,” Proc. 20th International Conference on Advanced Information Networking and Applications (AINA’06), 2006.

[54] T. Nguyen, “On the Efficiency of Dynamic Multicast Mobility Anchor Selection in DMM: Use Cases and Analysis,” 2014 IEEE International Conference Communications (ICC), 2014.

[55] T. Nguyen and C. Bonnet, “Load Balancing Mechanism for Proxy Mobile IPv6 networks: An IP Multicast perspective,” 2014 International Conference on Computing, Networking and Communications (ICNC), February 2014.

[56] Amitabh, K. (2010). Implementing Mobile TV; 2nd Edition. Focal Press.

[57] 3GPP2, “IP Network Architecture Model for CDMA2000 Spread Spectrum Systems,” Technical Report S.R0037, 3rd Generation Parnership Project 2 (3GPP2), 2002.

[58] A. Alexiou, C. Bouras and A. Papazois, “An Efficient Mechanism for UMTS Multicast Routing,” Mobile Network Appl, Springer 15:802-815, 2010.

[59] L. Wang, S. Gao and J. Guan, “Multicast Source Mobility Support Schemes in PMIPv6 Networks,” 2013 IEEE 78th Vehicular Technology Conference (VTC Fall), 2013.

[60] Y. Baddi and E. Kettani, “MC-PIM-SM: Multicast routing protocol PIM-SM with Multiple Cores Shared tree for Mobile IPv6 Environment,” 2012 2nd International Conference on Innovative Computing Technology (INTECH), September 2012.

[61] "Draft IEEE Standard for Local and Metropolitan Area Networks: Media Independent Handover Services", IEEE LAN/MAN Draft IEEE P802.21, July 2007.

[62] T. Melia, Ed., Bajko, G., Das, S., Golmie, N., and JC. Zuniga, "IEEE 802.21 Mobility Services Framework Design (MSFD)", RFC 5677, Internet Engineering Task Force, December 2009.

[63] J. Lee, T. Ernst, D. Deng and H. Chao, “Improved PMIPv6 Handover Procedure for Consumer Multicast Traffic,” IET Communications, volume:5, Issue: 15, page: 2149 – 2156, 2011.

[64] M. Waehlisch and T.C. Schmidt, "Between underlay and overlay: on deployable, efficient, mobility-agnostic group communication services," Internet Research,17(5), pp. 519-534. November 2007.

[65] H. Omar, T. Saadawi and M. Lee, "Multicast support for Mobile-IP with Hierarchical Local Registration Approach," 3rd ACM Wireless mobile multimedia, Boston, 2000.

[66] K. Namee and N. Linge, "A Framework of Multicast Mobility in Heterogeneous Networks", Proceeding of the 11th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet2010), Liverpool UK, pp.32-36, 2010.

[67] K. Namee, and N. Linge, "Designing a Protocol to Support Multicast Mobility in IPv6 Network", Proceeding of the 12th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PgNet2011), Liverpool UK, June 2011.

[68] A. Helmy, "A multicast-based protocol for ip mobility support," Proc. Networked Group Communication (NGC2000), pp. 49-58, 2000.

[69] ITU-T Recommendation, "G.114 - one-way transmission time," Telecommunication union standardization sector of ITU, May 2003.

[70] Zheng, L., &Hongji, Y. (2012). Unlocking the Power of OPNET Modeler. Cambridge University Press.