PERFORMANCE ANALYSIS OF MOBILE IPv4 WITH AND WITHOUT ROUTE OPTIMIZATION

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF
NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSTY

BY

SERAP ALTAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

MAY 2003

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Yurdahan GÜLER
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Turhan ALPER
Chairman Of The Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Gülsün TÖRECİ
Supervisor

Examining Committee Members

Prof. Dr. Ziya B.GÜVENÇ

Asst. Prof. Gülsün TÖRECİ

Dr. Mehmet ALTUNER

ABSTRACT

Performance Analysis of Mobile IPv4 With and Without Route Optimization

Altay, Serap

Ms, Department of Computer Engineering

Supervisor: Asst. Prof. Gülsün Töreci

May 2003, 92 pages

The support of mobility in the modern communications network is becoming essential and important with the development of mobile devices. Mobile Internet Protocol is built on IPv4. Mobile IP has been proposed by IETF (Internet Engineering Task Force) to support portable IP addresses on Internet. In the basic Mobile IP protocol, datagrams destined for the mobile node are sent from wired or wireless hosts. These datagrams have to travel through the home agent when mobile node is away from home. On the other hand, the datagrams sent from mobile hosts to wired host can be sent directly. This asymmetric routing, called "Triangle Routing". On the other hand, when the destination node is very close to mobile host, this creates a problem. Solving of the problem "Triangle Routing" is one appealing topic in mobile IP. IETF proposed extension part of the basic mobile IP, called "Route Optimization" to address this problem. IPv4 has already been widely deployed. Moreover, it will most probably dominate the Internet for a long time. Therefore, in this thesis implemented mobile IPv4 with and without route optimization in OMNeT++. Simulations have already been done to justify the modification.

Keywords: Mobile IP, mobile communication, route optimization, triangle routing.

ÖZ

Mobil Internet Protokolü Versiyon 4'de, Yönlendirme Optimizasyonlu ve Optimizasyonsuz Performans Analizi

Altay, Serap

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Y. Doç. Dr. Gülsün Töreci

Mayıs 2003, 92 sayfa

Mobil cihazların gelişmesiyle birlikte internete telsiz ve mobil erişimin sağlanması önemli ve temel bir ihtiyaç olmuştur. Mobil Internet Protokolü bu ihtiyaçtan dolayı IETF tarafından internette hareketliliği desteklemek için IPv4 temel alınarak geliştirildi. Temel Mobil IP'de, eğer mobil cihaz bağlı olduğu networkten başka bir networke gittiyse, mobil cihaz gönderilen data paketlerini kablolu ve kablosuz olarak kendi networkündeki yönlendiricisi yoluyla alır. Diğer taraftan mobil cihaz data paketlerini kablolu cihazlara direk olarak gönderir. Bu asimetrik yönlendirme "üçgen yönlendirme" olarak adlandırılır. Bu durum paketlerin taşımasında problemler oluşturmaktadır. IETF bu problemleri çözmek için "yönlendirme protokolünü" geliştirdi. IPV6 da bu sorunun çözümü için geliştirilmiştir, fakat IPv4 kullanımı daha baskın oldugu için, bu tezde IPv4 üzerinde çalışılmış ve optimizasyonlu ve optimizasyonsuz performance analizi için OMNeT++ proğramı kullanılmıştır. Simulasyon sonucunda yapılan çalışmanın doğruluğu gözlemlenmiştir.

Anahtar Kelimeler: Mobil IP, mobil iletişim, yönlendirme optimizasyonu, üçgen yönlendirme.

To my parents; Mağfire and Cevdet

ACKNOWLEDGMENTS

I express sincere appreciation to Asst.Prof. Gülsün Töreci and Orhan Gazi for his guidance and insight throughout the research. Then I must thank Asst. Prof. Halil Eyyuboğlu and Assoc. Prof. Dr. Yahya Baykal for their encouragements. My friends Çağlar Arpalı, Muammer Bal, and Barbaros Preveze is gratefully acknowledged. To my parents, I offer sincere thanks for their unshakable faith in me and their willingness to endure with me the vicissitudes of my endeavors.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Due to the increasing use of Personal Digital Assistants (PDAs), portable computers and cellular phones, there has been an increasing demand for wireless Internet access. However, some problems need to be solved before mobile access to the Internet becomes widespread. A first problem is caused by the way Internet Protocol (IP) routes packets to their destination according to IP addresses. This addresses are associated with a fixed network location and would not work in a wireless environment since when the mobile node moves, it will eventually reach a network, with a new network number and a new IP address [1].

Mobile IP was designed to solve this problem and other problems associated with mobile networking by allowing the mobile node to use two IP addresses, which are a fixed home IP address and a temporary care-of IP address that changes at each new point of attachment. Mobile IP protocol was built on the top of the Internet Protocol to support the mobility. Mobile IP protocol for IPv4 provides continuous Internet connectivity to mobile host. There are different mobility protocols. However these connection technologies protocols are not used for all communication technologies because of limited application areas.

There are several additional problems that need to be solved to make Mobile IP an efficient protocol. One of the major problems is the "Triangle Routing". In the basic mobile IP protocols datagrams are sent to the mobile host by the home agent when

the mobile node is away from home. Nevertheless datagrams are sent from mobile host to other wired node directly. This routing called "Triangle Routing", is generally far from optimal, especially when the destination node is close to the mobile node. Internet Engineering Task Force (IETF) to developed route optimization protocol over come of "Triangle Routing" problem.

Other approaches, like reverse routing are also proposed for this purpose. Actually, in the next generation of the Internet protocol IPv6, "Routing optimization" is integrated as a fundamental part of the mobility support [2]. However, IPv4 has already been widely deployed and will continuously dominate the Internet for a long time. In this thesis work mobile IP with and without route optimization is implement and using program OMNeT++. Result of the simulations have already been provable the modification.

In chapter 2 of the thesis a brief overview to mobility support in IPv4 is given. The chapters 3,4,5 include details to the introduction of mobile IPv4. The following chapter is survey on the routing schema in mobile IP protocol, including the " Triangle Routing" problem and the route optimization extension in mobile IP. The chapter 7 is brief introduction to OMNeT++. The chapter 8 focuses on the current implementation architecture of mobile IP in OMNeT++. It is the highlight, elaborating this implementation of mobile IP with Route Optimization in OMNeT++ from the system architecture modification to the implementation details. After extending Route Optimization on mobile IP, simulations are carried out for validation. The simulation scenarios, scripts and results will be presented in the section 9. Then, the report is finalized with the summary and discussion in chapter 10.

CHAPTER 2

MOBILE IPv4 PROTOCOL OVERVIEW

Mobile IP supports mobility when a mobile node is attached to a different network from its home network, as shown in Figure 2.1. Mobile IP address of the mobile node remains the same of the network Mobile IP has two IP adresses, a constant home address, and a care-of address. Care-of address changes when mobile node moves to a new foreign network.

Home agent sends all packets to the mobile node's care-of address via the foreign agent when the mobile node is away from its home network. When sending to the mobile node

Home agent constructs a new IP packet, which contains the mobile nodes care-of address as the destination IP address. The created IP packet, which is routed to the destination address, encapsulates the original IP packet. This procedure is called tunneling. As encapsulated the packet reaches to the care-of address, the original IP packet is decapsulated and delivered to the mobile node. There are two types of care-of addresses, foreign agent care-of address and collocated care-of address [1].

a. Foreign agent care-of address:

A foreign agent provides a care-of address through its agent advertisement message. Foreign agent's IP address is the care-of address and the foreign agent is the endpoint of the tunnel. Foreign agent receives tunneled packets and decapsulates them. The decapsulated packets are send to the mobile node. Since the foreign agents are

limited in the number of care-of addresses, this method useful for IPv4 which has limited addresses space

b. Co-located care-of address:

A co-located care of address is an IP address acquired by the mobile node as either statically or through the use of Dynamic Host Control Protocol (DHCP). The mobile node uses the co-located care-of address as end point of the tunnel with the home agent in the other end. In this case the mobile node decapsulates the tunneled datagrams itself. While the mobile node sends a packet tunneling is not require. The mobile node sends an IP packet directly to its destination address without home agent.

There are same protocols developed by Internet Engineering Task Force (IETF) about mobile IP these standard protocols are used when a mobile node moves to a new foreign agent. The protocols contains following areas [1]:

-Advertisement

-Registering

-Tunneling

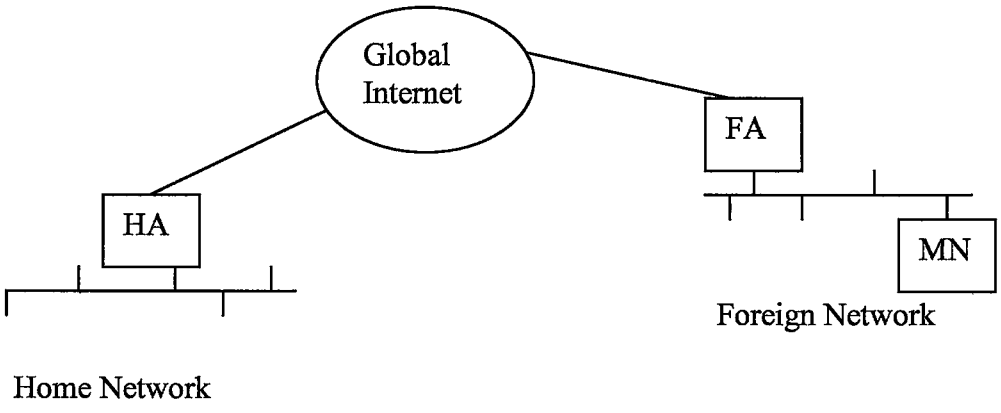


Figure 2.1:Basic principal of MIPv4

CHAPTER 3

ADVERTISEMENT

In mobile IP, getting care-of address is based on Router Advertisement protocol of the standard ICMP (Internet Control Message Protocol), specified in RFC 1256 [Request For Comments 1256]. In mobile IPv4, these router advertisements are called as agent advertisement. home agents and foreign agents broadcast these advertisements in a random mode at regularly intervals.

The agent advertisements includes the following functions:

- Detection of agent, type,i.e. home agent or foreign agent or both.
- Keeping available care-of addresses.
- Allows the mobile node to determine the network number and know their status to link to the Internet.
- Foreign Agent provides information about special features, such that alternative encapsulation techniques, for the mobile node.
- Mobile node listens these advertisements and decides whether it is in its home network or in a foreign network.

The home agents broadcast agent advertisements periodically. If a mobile node needs a care-of address and does not want to wait for the periodic advertisement, it can send out agent solicitation message that is to be responded by a foreign agent or home agent.

Mobile nodes detect any change in the set of mobility agents, which are available at the current point of attachment by the help of Internet Service Message Provider

(ISMP) router solicitations (agent solicitation). If a mobile node does not receive agent solicitation from a foreign agent anymore, it understands that it is not within the range of the foreign agent. In this case, the mobile node begins to search for a new care of address, or possibly use a care-of address known from advertisements it is still receiving. If the mobile node does not receive any recently advertised care of addresses, it may send an agent solicitation or it may choose to wait for another advertisement.

3.1 Agent Advertisement Message:

An agent advertisement is an ICMP router advertisement that has been extended to carry a mobility agent advertisement extension. A mobility agent transmits agent advertisements to advertise its services on a link. The agent advertisement can also carry other extensions as specified by the ICMP protocol. Agent advertisements include the following link-layer, IP, and ICMP header fields same as ICMP router advertisements [2].

3.1.1 Link-layer fields

Destination address: of a unicast agent advertisement that considered being the same as the source link layer address is must be the source link-layer address of the solicitation that prompted the advertisement [1].

3.1.2 IP fields

TTL: TTL (Time To Live) of the IP header must be set to 1 for all agent advertisements. This is to prevent solicitations and advertisements from reaching mobile nodes or mobility agents not on the same link.

Destination address: ICMP routers discover the IP destination address, which is required to be either the all systems multicast address (224.0.0.1) or the limited broadcast address (255.255.255.255).

3.1.3 ICMP fields

- Type: Type must be 9.

- Code: If the advertising agent also acts as a router for IP traffic, the code field must be 0, when the advertising agent forwards datagrams from the MN (Mobile Node) to an appropriate router then the code field must be 16.

- Lifetime: The Lifetime field must contain the period of time which advertisement is to be considered valid in the absence of further advertisements.

- Router address: The router address of any advertisement

- Num address: The number of router addresses advertised in this message.

3.1.4 Mobility Agent Advertisement Extension

The mobility agent advertisement extension format is shown in Figure 3.1. It follows the ICMP router advertisement fields. It means that an ICMP router advertisement being send by a mobility agent [1].

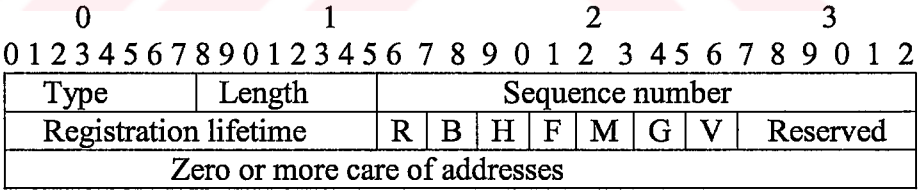


Figure 3.1: Mobility agent advertisement extension

Type: This field must be set to 16.

Length: (6+4*N), N is number of care-off address.

Sequence Number: The count of agent advertisement messages after initialization of the agent.

Registration Lifetime: Longest lifetime in any registration request; A value of 65,535 indicates infinity. Maximum registration lifetime= 18 hour (65535/3600).

R: Registration request. This means that a MN must register with the foreign agent (or another foreign agent on this link) and cannot use a co-located IP address.

B: Busy. The foreign agent will not admit registrations from additional mobile node when this bit is set. The capacity is full.

H: Home Agent. When this bit is set. The agent offers service as a home agent on the link on which the agent advertisement is sent.

F: Foreign Agent. This agent offers service as a foreign agent.

M: Minimal Encapsulation. The agent uses minimal encapsulation tunneling while implement receiving tunneled datagrams.

G: Generic Routing Encapsulation. The agent supports tunneling via minimal encapsulation technique. While Tunnels messages, it uses GRE methods.

V: Van Jacobsen header compression. This agent supports header compression over this link with any registered mobile node.

Reserved: The rest of the bits are reserved for future use. Send as 0;ignored on reception.

Care-of address: Foreign agent provides care-of address for the advertisement. An agent advertisement is required to include at least one care-of address when the F bit is set The number of care-of addresses present is determined by the length of the extension.

A home agent must always be prepared to serve its mobile nodes It never refuses the mobile nodes in its home network. A foreign agent may at times be too busy to serve additional mobile nodes; even so, it must continue to send agent advertisements, so

that any mobile nodes already registered with it will know that they have not moved out of range of the foreign agent and that the foreign agent has not failed [3]. A foreign agent may indicate that it is "too busy" to allow new mobile nodes to register with it, by setting the 'B' bit in its agent advertisements. An agent advertisement message must not have the 'B' bit set if the 'F' bit is not also set. Either the 'F' bit or the 'H' bit must be set in any agent advertisement message sent.

When a foreign agent wishes to require registration even from those mobile nodes, which have acquired a co-located care-of address, it sets the 'R' bit to one. Because this bit applies only to foreign agents, an agent must not set the 'R' bit to one unless the 'F' bit is also set to one.

There are two more extensions in the Mobile IP standard. These extensions are possible to use in connection with agent advertisement.

3.1.5 The Prefix-Length Extension

The prefix-length extension may follow the mobility agent advertisement extension [1]. It is used to specify the network prefix that applies to the routers listed in the ICMP router advertisement portion of the agent advertisement. . Note that the prefix lengths given don't apply to care-of addresses listed in the mobility agent advertisement extension. The prefix lengths extension field structure is shown in Figure 3.2.

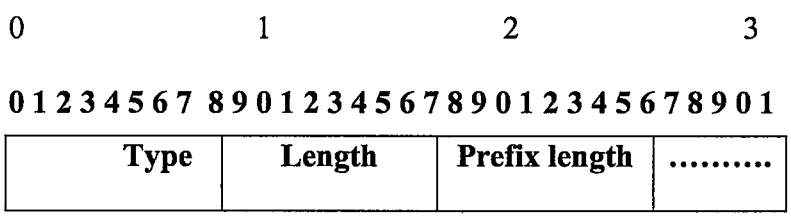


Figure 3.2: Prefix length extension format

Type: 19

Length: N, where N is the value (possibly zero) of the number address field in the ICMP router advertisement portion of the agent advertisement.

Prefix length: Network number of the corresponding router address, which is number of leading bits listed in the ICMP router advertisement portion of the message.

The prefix length for each router address is encoded as a separate byte, in the order that the router addresses are listed in the ICMP router advertisement portion of the message.

3.1.6 The One-byte Padding Extension

Some TCP/IP implementations insist on having only an even number of bytes in an ICMP message. One-byte padding extension since its necessary to have. There should be used only one one-byte padding extension for obvious reasons and this extension should be the last the last extension in the agent advertisement.

The one byte padding extension is encoded as a single byte in Mobile IP. The one byte padding extension is shown in Figure 3.3, where type is set to 0 to denote one-byte padding extension.

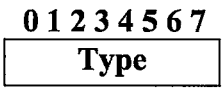


Figure 3.3: Pad extension format.

3.2 Agent Solicitation Message

The format of the agent solicitation is the same as the ICMP router solicitation. Agent solicitations do, however, always set the TTL (Time To Live) to one (When an IP host needs a local default router, router solicitation processing starts. It multicasts or broadcasts a router solicitation message. Any router in the vicinity will

respond with a unicast router advertisement sent directly to the soliciting host. After receiving the advertisement the host then responds just as if the advertisement were unsolicited.). However the way in which it is used is slightly different. There are some operational differences. The format of agent solicitation is shown in Figure 3.4.

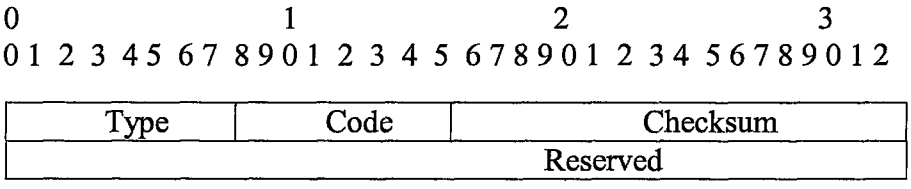


Figure 3.4: Agent solicitation message

Type: 10 refer to agent solicitation message.

Code:0

Checksum: The 16 bit one's complement of the one's complement sum of the ICMP message, starting with the ICMP type. To compute the checksum field is set to 0 [1].

Reserved: Send as 0, ignored on reception. Advertisements only need to be sent when the site policy requires registration with the agent (R- bit is set) or as a response to a specific agent solicitation [1].

Mobile IP agent advertisements uses the same procedures, defaults and constants as specified for ICMP router discovery (advertisement and solicitation) except for the following [4]:

- A mobility agent is required to limit the rate at which it sends agent advertisements (broadcast or multicast advertisements). A recommended maximal rate of one per second.
- A foreign agent must accept and respond to solicitations even if the source IP address in the solicitation does not appear to come from the foreign agent's own

subnet.

- A mobility agent may be configured to only send advertisements as a respond to solicitations.

3.3 Agent Discovery by Mobile Nodes

The mobile node should only send solicitations in the absence of agent advertisements absence of a care-of address determined by other means. The care-of address can be obtained through some link-layer protocol or a co-located care-of address can be acquired through DHCP. During solicitation the mobile node has to obey the same rules as the node sending normal ICMP router discovery solicitations. The only difference is that the mobile node may solicit more often than once every three seconds and a mobile node that is not connected to a foreign agent may solicit more than max_solicitations.

A mobile node limits the rate at which it sends solicitations. After sending three initial solicitations once a second, must use an exponential back off algorithm to reduce the rate until a maximum delay has been reached. The mobile node uses the mobility agent advertisement extension to distinguish between mobility agent advertisements and normal uses of the ICMP protocol.

While searching for an agent, the mobile node doesn't increase the rate at which it sends solicitations unless it has received a positive indication that it has moved to a new link. After the mobile node should increase the rate at which it sends solicitations if it begins searching for a new agent to register. The increased solicitation rate may revert to the maximum rate, but then must be limited in the manner described above. In all cases, the recommended solicitation intervals are nominal values. Mobile nodes must randomize their solicitation times around these nominal values as specified for ICMP router discovery [3].

Mobile nodes must process received agent advertisements. A mobile node can distinguish an agent advertisement message from other uses of the ICMP router advertisement message by examining the number of advertised addresses and the IP total length field. When the IP total length indicates that the ICMP message is longer than needed for the number of advertised addresses, the remaining data is interpreted as one or more Extensions. The presence of a mobility agent advertisement extension identifies the advertisement as an agent advertisement.

If there is more than one advertised address, the mobile node picks the first address for its initial registration attempt. If their registration attempt fails with a status code indicating rejection by the foreign agent, the mobile node may retry the attempt with each subsequent advertised address in turn.

When multiple methods of agent discovery are in use, the mobile node should first attempt registration with agents including mobility agent advertisement extensions in their advertisements, in preference to those discovered by other means. This preference maximizes the likelihood that the registration will be recognized, thereby minimizing the number of registration attempts. A mobile node must ignore reserved bits in agent advertisements, as opposed to discarding such advertisements. In this way, new bits can be defined later, without affecting the ability for mobile nodes to use the advertisements even when the newly defined bits are not understood.

CHAPTER 4

REGISTRATION

Registration operation of mobile IP provides for the mobile nodes to communicate and to send their information to their home agents. Mobile nodes use registration to request forwarding services when visiting a foreign network, inform their home agent of their present care-of address. When mobile nodes return to their home network. Their mobility bindings expire and they deregister.

The purpose of registration is to let the home agent keep track of the mobile nodes current care-of address to tunnel incoming datagrams, destined for the mobile node. This kind of registration that redirect traffic bear substantial security risks and therefore some means of authentication are necessary. When the mobile nodes register the home agent creates a mobility binding. A mobility binding is an association of the mobile node's home address with its current care-of address and the remaining lifetime of that association.

Several other optional capabilities are available through the registration procedure, which enables a mobile node to do the following [1]:

- Discover the address of a home agent if the mobile node is not configured with this information.
- Choose between alternative tunneling protocols (IP-in-IP, minimal encapsulation and GRE).
- Use Van Jacobson header compression.

- Maintain multiple simultaneously mobility bindings so that a copy of each datagrams is tunneled to each active care-of address.
- Deregister certain care-of addresses while retaining others.

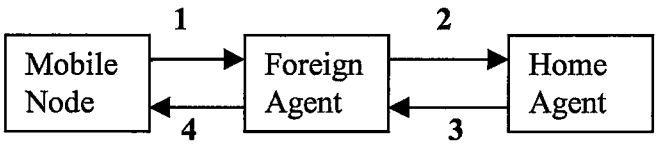
There are two variations of registration procedure depending on whether the mobile node tries to register a colocated care-of address or the care-of address of a foreign agent. In the latter case the mobile node is required to register via that foreign agent. The mobile node is required to register via a foreign agent if it has received an agent advertisement with the R bit set. If the mobile node has returned to its home network it first deregister with its home agent, then sends the registration request addressed directly to its home agent. Likewise the mobile node naturally sends the registration request addressed directly to its home agent if a colocated care-of address is used. While registering via a foreign agent the registration procedure uses the following four message types, illustrated in Figure 4.1 [4].

The registration steps:

- The mobile node sends a registration request to the prospective foreign agent to begin the registration process.
- The foreign agent processes the registration request and relays it to the home agent.

The home agent sends a registration reply to the foreign agent to approve or refuse the request. The foreign agent processes the registration reply and relays it to the mobile node to inform it of the disposition of its requests.

If the mobile node tries to register using with its home agent, the registration procedure consists of the mobile node sending the registration request to the home agent. The home agent then processes the request and sends back a reply mobile node.



1: Mobile node sends a registration request.

2: Foreign agent relays request to home agent.

3: Home agent sends registration reply (accepts or denies).

4: Foreign agent relays status to home agent.

Figure 4.1: Mobile IP registration overview.

Mobile IP registration messages use the User Datagram Protocol (UDP). The registration message is shown in Figure 4.2. A nonzero UDP checksum should be included in the header, and must be checked by the recipient. The recipient accepts a zero UDP checksum. The behavior of the mobile node and the home agent with respect to their mutual acceptance of packets with zero UDP checksums should be defined as part of the mobility security association, which exists between them [1].

IP header fields	UDP header	Mobile IP message header	Extension …

Figure 4.2: General Mobile IP registration message format

4.1 Registration Request:

A mobile node uses a registration request message to registers with its home agent once register is finished. Its home agent can create or modify a mobility binding for that mobile node (e.g., with a new lifetime). The request may be relayed to the home agent by the foreign agent through which the mobile node is registering, or it may be sent directly to the home agent for the case in which the mobile node is registering a

co-located care-of address. The registration request message consists of the following fields.

IP fields:

Source Address: Typically the interface address from which the message is sent.

Destination Address: Typically that of the foreign agent or the home agent.

UDP fields:

Source Port variable

Destination Port 434

Mobile IP fields:

The UDP header followed by the mobile IP fields shown in Figure 4.3.

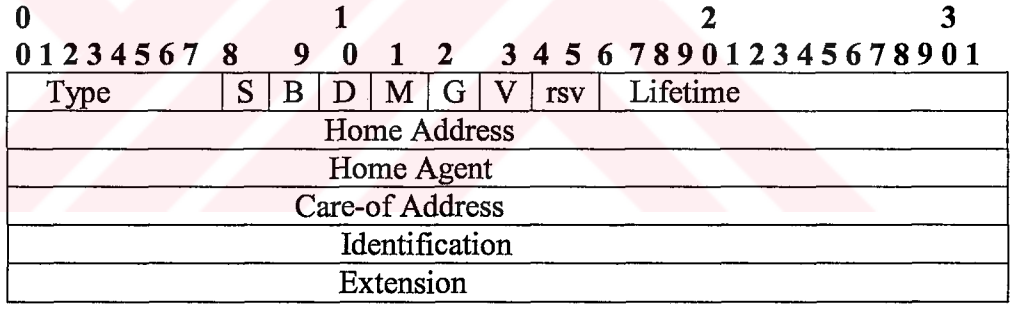


Figure 4.3: Registration request packet format.

Type - 1 (registration request).

S (Simultaneous bindings.): By setting this bit, the mobile node requests that the home agent retains its prior mobility bindings.

B (Broadcast datagrams.): By setting this bit, the mobile node request that broadcast datagrams will be tunneled to it.

D (Decapsulation.): By setting this, the mobile node indicates the use of a colocated care-of address.

M (Minimal encapsulation.): The mobile node requests the use of minimal encapsulation by setting this bit.

G: GRE encapsulation.

V: Van Jacobson header compression. By setting this bit the mobile node. request the use of VJ header compression

rsv :The rest of the bits in this byte is reserved for future use.

Lifetime : The number of seconds before the registration is considered expired. A value of zero indicates a request for deregistration. A value of 0xffff indicates infinity.

Home Address: The IP address of the mobile node.

Home Agent: The IP address of the mobile node's home agent.

Care-of address: The IP address of the tunnel endpoint.

Identification: A 64-bit number generated by the mobile node used to match registration request with registration replies and for protecting against replay attacks.

Extensions: Everything that follows the fixed part.

4.2 Registration Reply

The registration request is answered by a registration reply by the home agent. The registration reply packet format is shown in Figure 4.4. The mobile node is requesting service from a foreign agent, that foreign agent will receive the registration reply from the home agent and subsequently relay it to the mobile node. The registration reply message contains the necessary codes to inform the mobile node about the status of its registration request, along with the lifetime granted by the home agent, which may be smaller than the original request. Registration lifetime allowed by the foreign agent. If the lifetime received in the registration reply is greater than that in the registration request, the lifetime in the request must be used.

When the lifetime received in the registration reply is less than that in the registration request, the lifetime in the reply message must be used. The field in the IP header, UDP header, and in the registration request message.

IP fields:

Source Address: Typically copied from the destination address of the registration request to which the agent is replying.

Destination Address: Copied from the source address of the registration request to which the agent is replying

UDP fields:

Source Port: <variable>

Destination Port: Copied from the source port of the corresponding registration request.

The Mobile IP fields shown below follow the UDP header:

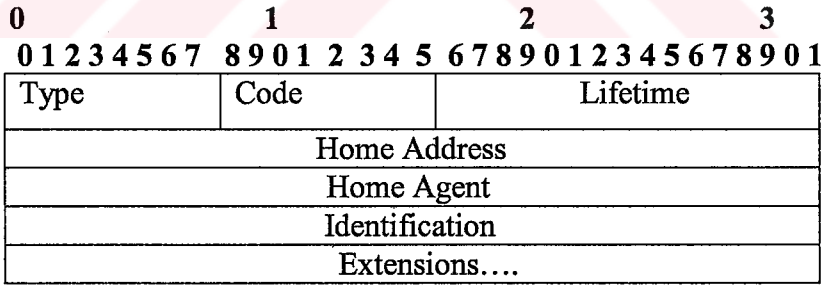


Figure 4.4: Registration replay packet format.

This reply either grants or denies service to the mobile node by the home agent. The fields in the reply has the following meaning:

Type: 3 (Registration reply).

Code: A value indicating the result of the registration request.

Lifetime: The duration in seconds for which the registration is valid. Note that this value may be smaller than the requested lifetime.

Home address: The IP address of the mobile node.

Home agent : The IP address of the home agent.

Identification: A 64 bit number generated by the mobile node used to match registration request with registration replies and for protecting against replay attacks.

Extensions: Everything that follows the fixed part.

4.3 Registration Extension:

Each mobile node, foreign agent, and home agent must be able to support a mobility security association for mobile entities, indexed by their SPI (Security Parameter Index) and IP address. In the case of the mobile node, this must be its home address. Registration messages between a mobile node and its home agent must be authenticated with an authorization-enabling extension, e.g. the Mobile-home authentication extension. This extension must be the first authentication extension; other foreign agent-specific extensions may be added to the message after the mobile node computes the authentication.

Authentication is provided as an extension to the Mobile IP protocol. Three such registration extensions are defined all of which allows additional security measures in the registration procedures. The three extensions are:

Mobile-home authentication extension.

Mobile-foreign authentication extension.

Foreign-home authentication extension.

Exactly one Mobile-home authentication extension is required to be present in all registration requests and replies. The location of the extension marks the end of the

data authenticated by the extension. The other two extensions can be used if the parties share a mobility security association, but are not required. The security association to be used is pointed to by the Security Parameter Index (SPI).

The default authentication algorithm used is keyed Message Digest 5 (MD5) (Rivest 1992) which is a cryptographic hash function. The algorithm is used in prefix + suffix mode [3]. This means that the data to be authenticated is preceded and followed by the shared secret. The MD5 checksum is computed over the following stream of bytes:

- The shared secret defined by the security association between the nodes.
- The registration request or registration reply header fields.
- All prior extensions in their entirety.
- The type, length, and SPI included within the extension itself.
- The shared secret again.

The authentication extension follows the TLV (Type - Length - Value) format see Figure 4.5. The type field can take three different values (32, 33 or 34) depending on whether it is a Mobile-home, Mobile-foreign or a Foreign-home authentication extension. The length field is 4 plus the number of bytes in the authenticator, the SPI identifies the security association and the authenticator is a variable length field whose the length depends on the authentication algorithm.

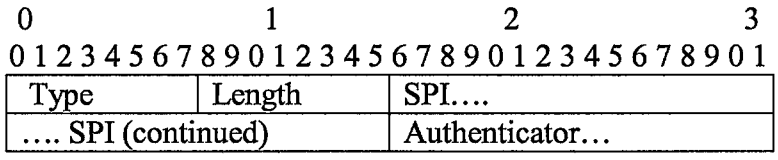


Figure 4.5: Authentication extension packet format.

Type: 32 Mobile-home authentication extension.

33 Mobile-foreign authentication extension.

34 Foreign-home authentication extension.

Length: 4 plus the number of bytes in the authenticator.

SPI: Four bytes; an opaque identifier.

Authenticator: Variable length, depending on the SPI.

This authentication scheme makes it impossible for someone to impersonate another without knowledge of the shared secret and because the identification field in the registration request and reply cannot be reused it is not possible to use a replay attack to redirect the traffic.

CHAPTER 5

TUNNELING

While mobile node is away from home network mobile IP requires the use of tunneling to deliver datagrams from the home network to the mobile node. During the tunneling operation; datagrams are encapsulated in the home agent (entry point of the tunnel) and sent to the foreign agent. Foreign agent decapsulates packets and sends them to the mobile node as soon as it receives encapsulated packets. The most general tunneling case illustrated in Figure 5.1, where the source, encapsulator, decapsulator, and destination are separate nodes.

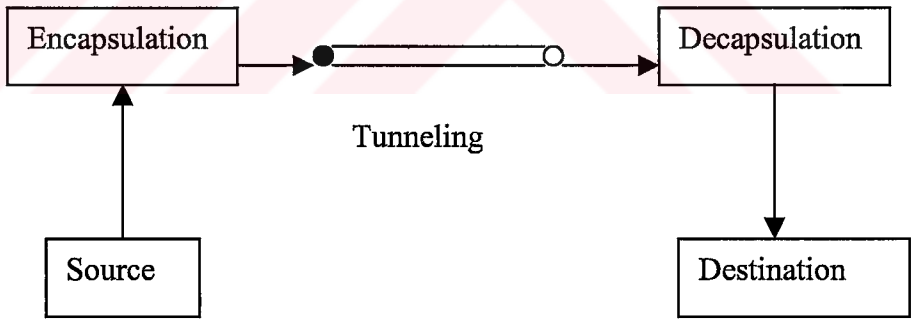


Figure5.1: General tunneling.

There are several methods of encapsulation. Mobile IP requires each home agent and foreign agent to support tunneling datagrams using IP-in-IP encapsulation [1]. Minimal encapsulation and GRE encapsulation are alternate encapsulation methods.

5.1 IP-in-IP Encapsulation

The most commonly used method of tunneling is IP-in-IP encapsulation, where an IP packet destined for a mobile node is intercepted by the home agent and encapsulated within the payload of another IP packet, as shown in Figure 5.2. The encapsulated packet is then tunneled to the care-of address registered with the mobile node. This method requires only the foreign agent and home agent to implement IP-in-IP Encapsulation so that home agent can send packets to care-of address via tunneling. All routers within the tunnel's path are obvious to the fact that they are routing encapsulated packets.

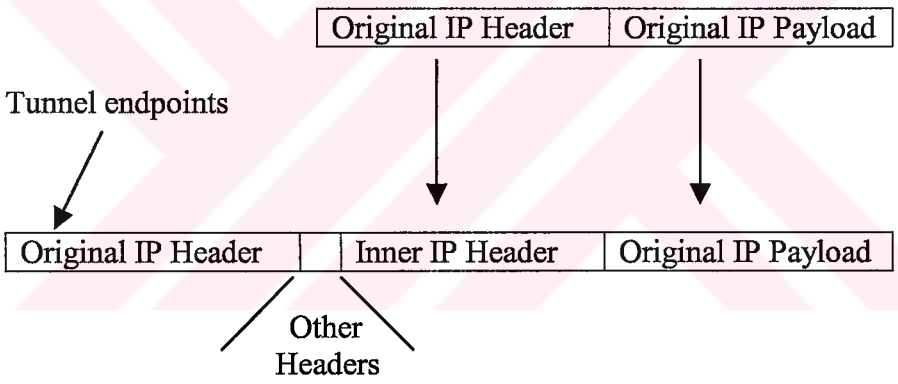


Figure 5.2: IP-in-IP encapsulation.

The figure demonstrates that there may be other header included between the inner header and the outer header of the encapsulated datagram. The outer header source address and destination address identify the end points of the tunnel. The inner IP header source address and destination address identify the original sender and recipient of the datagram respectively.

The inner IP header is not changed by the encapsulator and remains unchanged during its delivery to the tunnel exit point. If necessary other protocol headers such

as the IP authentication header may be inserted between the outer IP header and the inner IP header.

5.2 Minimal Encapsulation

It is defined within the basic Mobile IP protocol, and While IP-in-IP encapsulation protocol adds 20 bytes (the size of an IP header) to each packet tunneled to a mobile node, minimal encapsulation protocol adds only 8 or 12 bytes to each packet [5]. Instead of totally encapsulating each packet to be tunneled, minimal encapsulation inserts a minimal forwarding header between the original IP header and payload of the packet. The original header is modified and the minimal forwarding header is also filled before the packet is tunneled, as shown in Figure 5.3. Result of original IP header is modified in encapsulating the datagram as in [6].

- Protocol number 55 for the minimal encapsulation protocol replaces the Protocol field in the IP header.
- The Destination Address field in the IP header is replaced by the IP address of the exit point of the tunnel.
- If the encapsulator is not the original source of the datagram, the source address field in the IP header is replaced by the IP address of the encapsulator.
- The total length field in the IP header is incremented by the size of the minimal forwarding header added to the datagram. This incremental size is either 12 or 8 octets, depending on whether or not the Original Source Address Present (S) bit is set in the forwarding header.
- The header checksum field in the IP header is recomputed or updated to account for the changes in the IP header described here for encapsulation.

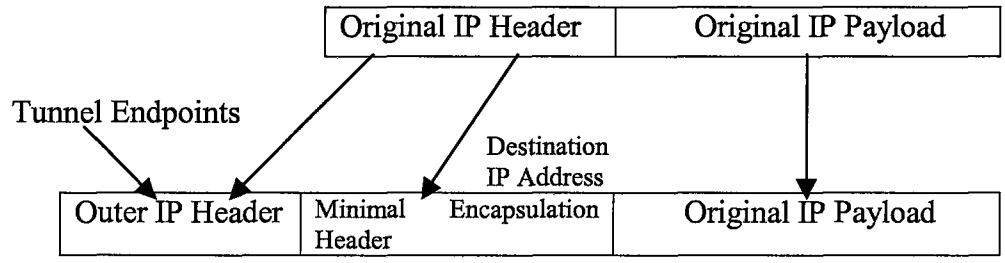


Figure 5.3: Minimal encapsulation.

5.3 Generic Routing Encapsulation

Generic routing encapsulation allows multiple protocols to be used during encapsulation procedure, whereas IP in IP encapsulation and minimal encapsulation only support IP [7]. The entire encapsulated packet has the form presented in Figure5.4. If a situation required 'P' number of protocols for Payload packets, and 'D' number of protocols for delivery packets, a total number of D x P documents would be needed to describe how every payload packet with its own protocol would be encapsulated within every delivery packet, also with its own protocol. However, with generic routing encapsulation, only D + P documents would be needed to describe all possible combinations of encapsulation. 'P' number of documents would be needed to describe how to encapsulate each Payload protocol within generic routing encapsulation, and 'D' number of documents would be needed to describe how to encapsulate generic routing encapsulation within various delivery packets generic routing encapsulation, at this point, becomes a very attractive solution for organizations with many protocols running on their networks.

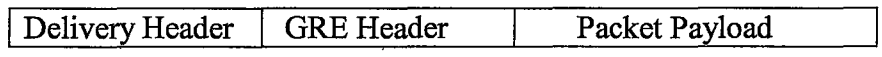


Figure 5.4: Generic Routing Encapsulation packet structure

CHAPTER 6

ROUTE OPTIMIZATION

In the mobile IP protocol, IP packets destined to a mobile node that is outside its home network are routed through the home agent. However packets from the mobile node to the correspondent node are routed directly. This is known as triangle routing [3]. Figure 6.1 illustrates triangle routing.

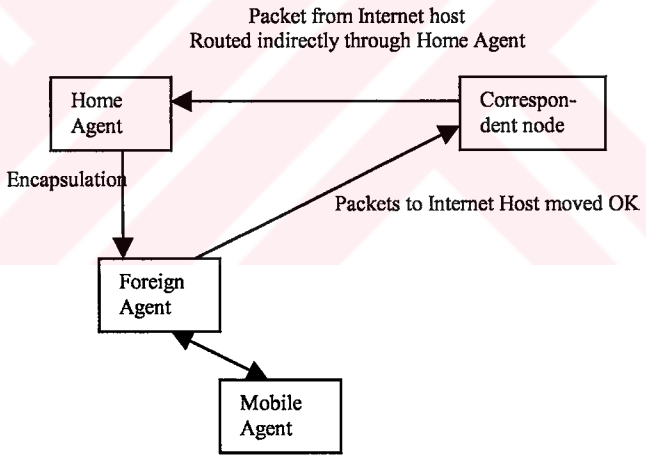


Figure 6.1: Triangle routing.

Triangle routing may be inefficient in many cases. It increases the burden on the Internet especially in heavy traffic situation or when the number of mobile nodes roaming in the Internet increases. The second problem is that packets in flight are lost when a mobile node moves from one foreign network to another foreign

network. Therefore, the IETF mobile IP working group proposes the Mobile IP with Route Optimization protocol. Route Optimization is an extension proposed to the basic Mobile IP protocol [8]. Here messages from the correspondent node are routed directly to the mobile node's care-of address without having to go through the home agent.

Route Optimization provides four main operations [1]:

1. Updating binding caches,
2. Managing smooth handoffs between foreign agents,
3. Acquiring registration keys for smooth handoffs,
4. Using special tunnels.

6.1 Binding Caches

Route optimization gives a way for any host to maintain a binding cache containing the care-of address of one or more mobile nodes. When sending an IP datagram to a mobile node, if the sender has a binding cache entry for the destination mobile node, it can tunnel the datagram directly to the care of address indicated in the cached mobility binding. When there is no binding cache entry, sender tunnel the datagram to the care of address through mobile node's home agent. This is the only routing mechanism supported by the base mobile IP protocol. With the route optimization mobile node's home agent sends the current mobility binding information of the mobile node to the sender. After that the sender would be able to send datagrams to mobile node without the service of the home agent. Each node may maintain a binding cache to optimize its own communication with mobile nodes. Correspondent node can create a binding entry for a mobile node only when it has received the mobile node's mobility binding.

There are four kinds of binding cache maintenance message in route optimization:

* Binding Update message.

* Binding Acknowledgement message.

* Binding Request message.

* Binding Warning message.

When a mobile mode's home agent intercepts a datagram from correspondent node and tunnels it to the mobile node, the home agent decides that the correspondent has no binding cache entry for the destination mobile node. The home agent then will send a Binding Update message to the correspondent node, informing it of the mobile node's current mobility binding, and in particular its care-of address. After receiving a binding update message from the home agent of the mobile node, it will tunnel the datagram to the mobile node's care-of address [8].

The following Figure 6.2 is intended to help to understand how the binding updates in route optimization. The binding update message contains the care-of address and the home address of the mobile node and also the lifetime of the binding. It also must contain a mobile IP authentication extension. An identification number may also be present to provide a way of matching updates with acknowledgement and to protect against replay attacks.

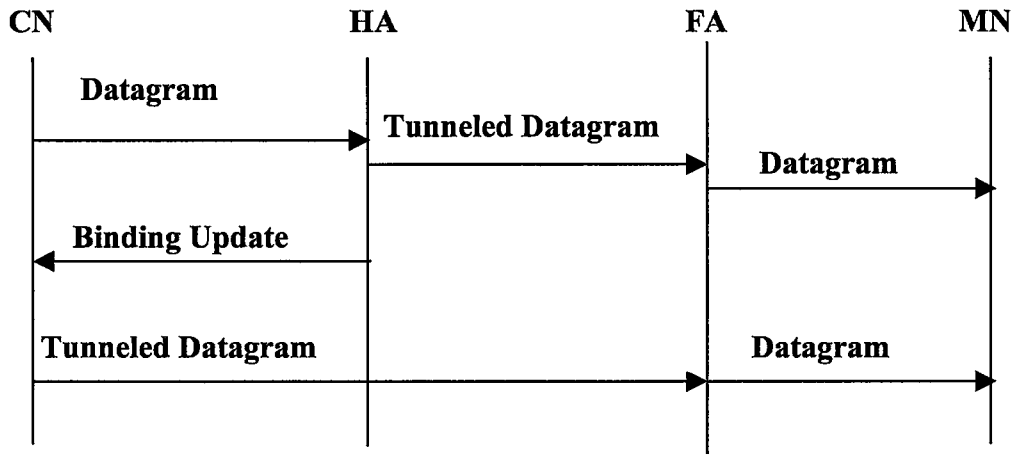


Figure 6.2: Binding update to correspondent node.

The format of the binding update message is illustrated in Figure 6.3 [1].

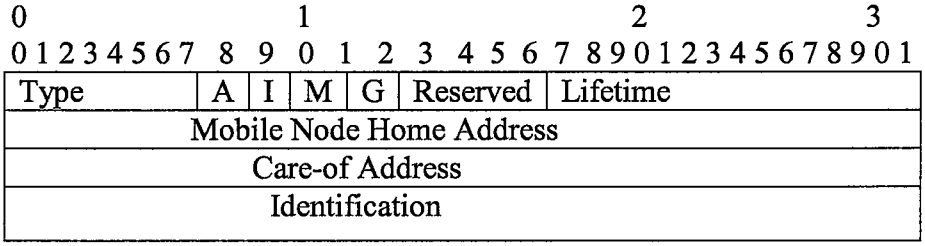


Figure 6.3: Binding update message format.

The type is 18, the A (acknowledgement) bit is set by the node sending the binding update message to request that a binding acknowledgement message be returned. The I (identification present) bit is set by the node sending the binding update message if the identification field is present in the message. If the M (minimal encapsulation) bit is set, datagram may be tunneled to the mobile node using the minimal encapsulation protocol. If the G (Generic Record Encapsulation) bit is set, datagrams may be tunneled to the mobile node using generic record encapsulation. The reserved sent as 0 (ignored on reception). The number of seconds remaining before the binding cache entry must be considered expired. A value of all ones indicates infinity. A value of 0 indicates that no binding cache entry for the mobile node should be created and that any existing binding cache entry for the mobile node should be deleted. The lifetime is typically equal to the remaining lifetime of the mobile node's registration. The mobile node home address is the home address of the mobile node to which the binding update refers. The care of address is current care-of address of the mobile node. The binding update message indicates that no binding cache entry if set equal to the home address of the mobile node. If present, the identification field contains a 64-bit number assigned by the node sending the binding request message, used to assist in matching requests with replies and to protect against replay attacks [3].

The correspondent node or the foreign agent in response to the binding update sends binding acknowledgement. It contains the mobile node's home address and a status code. It also contains an identification number, if there was one in the corresponding binding update. The format of binding acknowledgement message is illustrated in Figure 6.4.

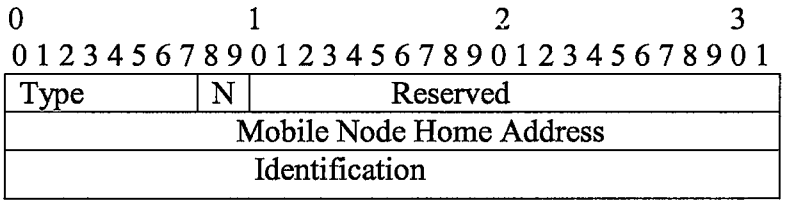


Figure 6.4: Binding acknowledgement message format.

The type is 19. If the N (negative acknowledge) bit is set, the acknowledgement is negative. If the binding update was not accepted but the incoming datagram has the acknowledge (A) flag set, then the N bit should be set in this binding acknowledge message. The reserved sent as 0. The mobile node home address copied from the binding update message being acknowledged. The identification copied from the binding update message being acknowledged, if present.

Binding request is sent by the correspondent node to the home agent to request a binding update. It contains the home address of the queried mobile node and possibly an identification number. The format of the binding request message is illustrated in Figure 6.5.

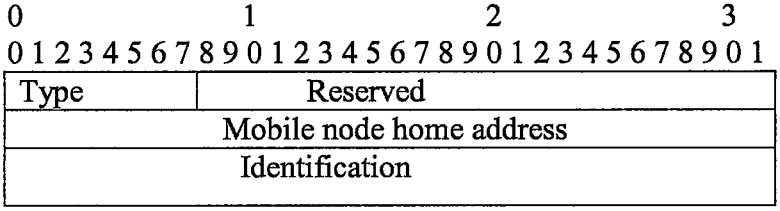


Figure 6.5: Binding request message format.

The type is 17. The reserved sent as 0. The mobile node home address is the home address of the mobile node to which the binding request refers. The identification field contains a 64-bit sequence number assigned by the node sending the binding request message, used to assist in matching requests with replies and to protect against replay attacks [3].

Binding warning message is used to transmit warning that one or more correspondent nodes need a binding update message. The mobile node can initiate a binding warning message to its home agent, requesting the home agent to send a binding update message to its correspondent host whenever it obtains a new care-of address from a new foreign agent. The binding warning is send to the home agent. The format of the binding warning is illustrated in Figure 6.6.

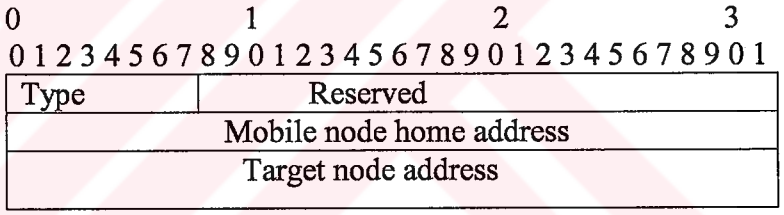


Figure 6.6: Binding warning message.

The type is 16. The reserved sent as 0. The mobile node home address is the home address of the mobile node to which the binding warning message refers. The target node address is the address of the node tunneling the datagram that caused the binding warning message; the target of the binding update message sent by the home agent.

6.2 Managing Smooth Handoffs Between Foreign Agents

When mobile node registers with a new foreign agent, the basic mobile IP does not specify a method to inform the previous foreign agent. Thus the datagram in flight, which had already tunneled to the old care-of address of the mobile node, are lost.

This problem is solved in route optimization by introducing smooth handoffs. Smooth handoff provides a way to notify the previous foreign agent of the mobile node's new mobility binding [9].

If foreign agent supports handoffs, it indicates this in its agent advertisement message. When the mobile node moves to a new location, it requests the new foreign agent to inform its previous foreign agent about the new location as part of the registration procedure. The new foreign agent then constructs a binding update message and sends it to the previous foreign agent of the mobile node. Thus if the previous foreign agent receives packets from a correspondent node having an out-of-date binding, it forwards the packet to the mobile node's care-of address. It then sends a binding warning message to the mobile node's home agent. The home agent in turn sends a binding update message to the correspondent node. This notification also allows datagrams sent by correspondent nodes having out-of-date binding cache entries to be forwarded to the current care-of address.

6.3 Acquiring Registration Keys For Smooth Handoffs

For managing smooth handoffs, mobile nodes need to communicate with the previous foreign agent. This communication needs to be done securely as any careful foreign agent should require assurance that it is getting authentic handoff information and not arranging to forward in-flight datagrams to a bogus destination. For this purpose a registration key have been proposed in the order of declining preference [10]:

- If the home agent and the foreign agent share a security association, the home agent can choose the registration key.
- If the foreign agent has a public key, it can again use the home agent to supply the registration key.

- If the mobile node includes its public key in its registration request, the foreign agent can choose the new registration key.
- The mobile node and its foreign agent can execute the Diffie-Hellman key exchange protocol as part of the registration protocol.

This registration key is used to form a security association between the mobile node and the foreign agent.

6.4 Using Special Tunnels

When a foreign agent receives a tunneled datagram for which it has no visitor list entry, it concludes that the node sending the tunneled datagram has an out-of-date binding cache entry for the mobile node. If the foreign agent has a binding cache entry for the mobile node, it should re-tunnel the datagram to the care-of address indicated in its binding cache entry. On the other hand, when a foreign agent receives a datagram for a mobile node for which it has no visitor list or binding cache entry, it constructs a special tunnel datagram [10]. Encapsulating the datagram and making the outer destination address equal to the inner destination address construct the special datagram. This allows the home agent to see the address of the node that tunneled the datagram and prevent sending it to the same node. This avoids a possible routing loop that might have occurred if the foreign agent crashed and lost its state information.

CHAPTER 7

INTRODUCTIONS TO OMNeT++

OMNeT++ is an object-oriented modular discrete event simulator. The name itself stands for Objective Modular Network Tested in C++. OMNeT++ has its distant roots in OMNeT, a simulator written in Object Pascal by dr. György Pongor [11].

The modeling of communication protocols, this simulator can be used for computer networks and traffic modeling multi-processor and distributed systems, administrative systems, ... and any other system, can. An OMNeT++ model consists of hierarchically nested modules. Modules communicate with message passing. Messages can include arbitrarily complex data structures. Modules can send messages either directly to their destination or along a predefined path, through gates and connections [11].

Modules can have parameters which are used for to customize module behavior, to create flexible model topologies (where parameters can specify the number of modules, connection structure e.t.c), and for module communication, as shared variables. The user does not need to learn a new programming language, but must to have some knowledge of C++ programming to write simple modules. OMNeT++ simulations can feature different user interfaces for different purposes as debugging, demonstration and batch execution. The models visibility has advantages for user. For example user can intervene by changing variables/objects in the model. User interfaces also facilitate demonstration of how a model works. Since it was written in

C++, the simulator is basically portable; it should run on most platforms with a C++ compiler. OMNeT++'s advanced user interfaces support X-window, DOS and are portable to Win3.1/Win95/WinNT. OMNeT++ has been extended to execute the simulation in parallel. Any kind of synchronization mechanism can be used. One suitable synchronization mechanism is the statistical synchronization, for which OMNeT++ provides explicit support. OMNeT++ is targeted at roughly the same segment of network simulation as OPNET.

7.1 Overview to OMNeT++

OMNeT++ provides efficient tools for the user to describe the structure of the actual system. The OMNeT++ model consists of hierarchically nested modules, which communicate with messages. OMNeT++ models are often referred to as networks. The top-level module is the system module. The system module includes submodules, which can also contain submodules themselves. The model is showed Figure 7.1 [11].

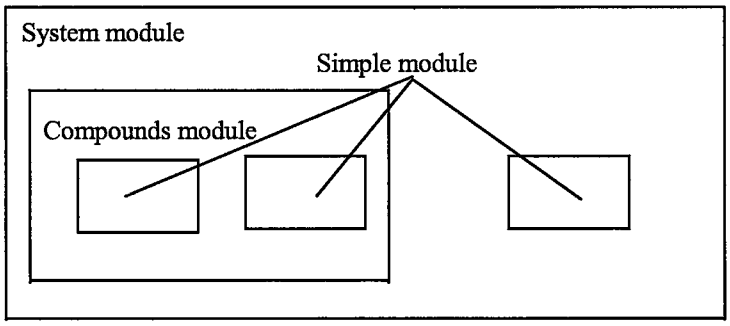


Figure 7.1: Simple and compound modules.

Modules that contain submodules are termed compound modules, as opposed simple modules, which are at the lowest level of the module hierarchy. Simple modules

contain the algorithms in the model. The user implements the simple modules in C++, using the OMNeT++ simulation class library [11].

Both simple and compound modules are instance of module types. The user defines module types; instances of these module types serve as components for more complex module types while describing the model. Finally, the user creates the system module as an instance of a previously defined module type; all modules of the network are instantiated as submodules and sub-submodules of the system module. Both module types can be connected via links. Links originate from an output gate on a module and connect to an input gate on either the same or another module. Gates are the input and output interfaces of modules; messages are sent out through output gates and arrive through input gates. The sending and arrivals of messages are considered as discrete events. The receiver module becomes the new owner of the message. It also possible to send self message which are often useful for modeling timers.

The structure of a scenario (modules, submodules and links) can be specified in a description language called NED. However, it is possible to create all of these components during runtime of a simulation using the OMNeT++ specific methods.

7.2 Building Simulation Programs

As it was already mentioned, an OMNeT++ model physically consists of the NED language topology description(s), which are files with the .Ned suffix and Simple modules. These are C++ files, with .cc suffix [11].

Model files are usually placed in the projects/modelname subdirectory of the main OMNeT++ directory. The NED files are compiled into C++ using the NEDC compiler, which is part of OMNeT++. The NEDC compiler (source and executable) is normally located in the nedc subdirectory of the main OMNeT++ directory.

Simulation programs are built from the above components. First, the NED files are compiled into C++ source code using the NEDC compiler. Then all C++ sources are compiled and linked with the simulation kernel and a user interface to form a simulation executable. The following Figure 7.2 gives an overview of the process of building and running simulation programs [11].

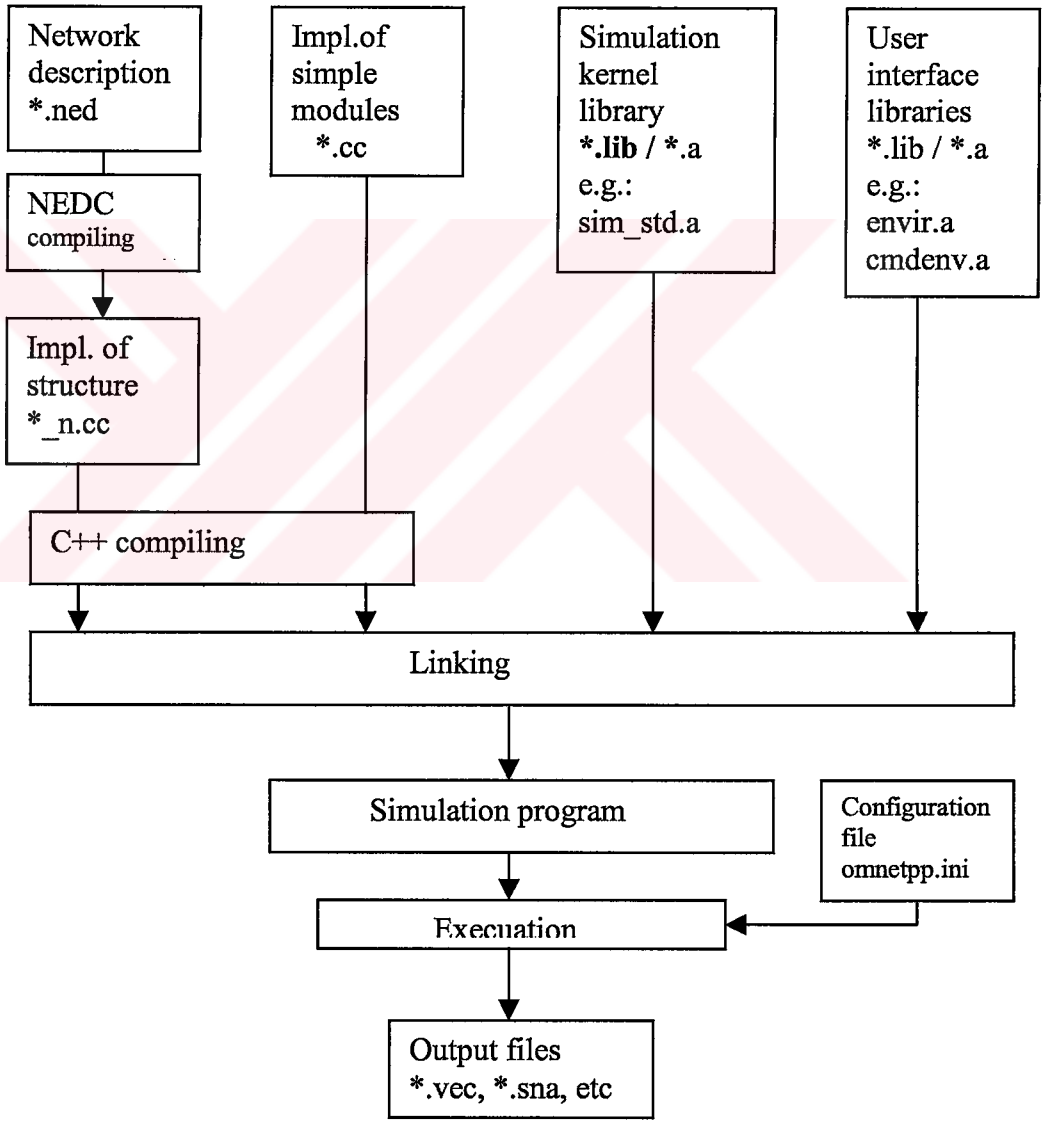


Figure 7.2: Figure gives an overview of the process of building and running simulation program.

CHAPTER 8

PERFORMANCE ANALYSIS

In this dissertation the route optimization extension of mobile IP over IPv4 is implemented in OMNeT++. The implementation also includes Binding Update message of the route optimization messages.

The first experiment modifies and extends the current modules of Mobile IP in OMNeT++. The usual routing scheme in Mobile IP without optimization to the routing with optimization in terms of the end-to-end packet delay is compared.

The results show that Mobile IP with the route optimization extension conducts a better performance than basic Mobile IP with smaller end-to-end packet delay.

The objectives of the experiment are listed as follows:

1. Understand the mobile IP protocol in IPv4.
2. Understand the Route Optimization in mobile IP.
3. Figure out the current mobile IP and mobile IP with route optimization architecture in OMNeT++.
4. Implement the mobile IP when mobile node in home network in OMNeT++.
5. Implement the mobile IP with and without route optimization in OMNeT++ when mobile node in foreign network.
6. Simulate the mobile IP scenarios with and without route optimization.
7. Analyses and evaluate the simulation results of mobile IP with and without route optimization.

8.1 Mobile IP With And Without Route Optimization

The first and mandatory step of the experiment is to figure out the current architecture of mobile IP implementation in OMNeT++.

The architecture includes typical mobile IP scenario consist of home agent (HA), Foreign Agent (FA), Mobile Host (MH) and Correspondent Node (CN). In the OMNeT++ system, Home Agent, Foreign Agent and Correspondent Node are basically the same kind of node and they use the same agent to handle the packets. Since the Home Agent and Foreign Agent play the role to interconnect the wired and wireless nodes, they are implemented as Hybrid nodes of both wired and wireless nodes. In OMNeT++, each node is a simple module and passes input and output gates which are linked to each other as a wired or wireless links.

Structure of Mobile IP with mobile node in its home network in OMNeT++ is shown Figure 8.1 and structure of Mobile IP with and without Route Optimization in OMNeT++ is shown Figure 8.2.

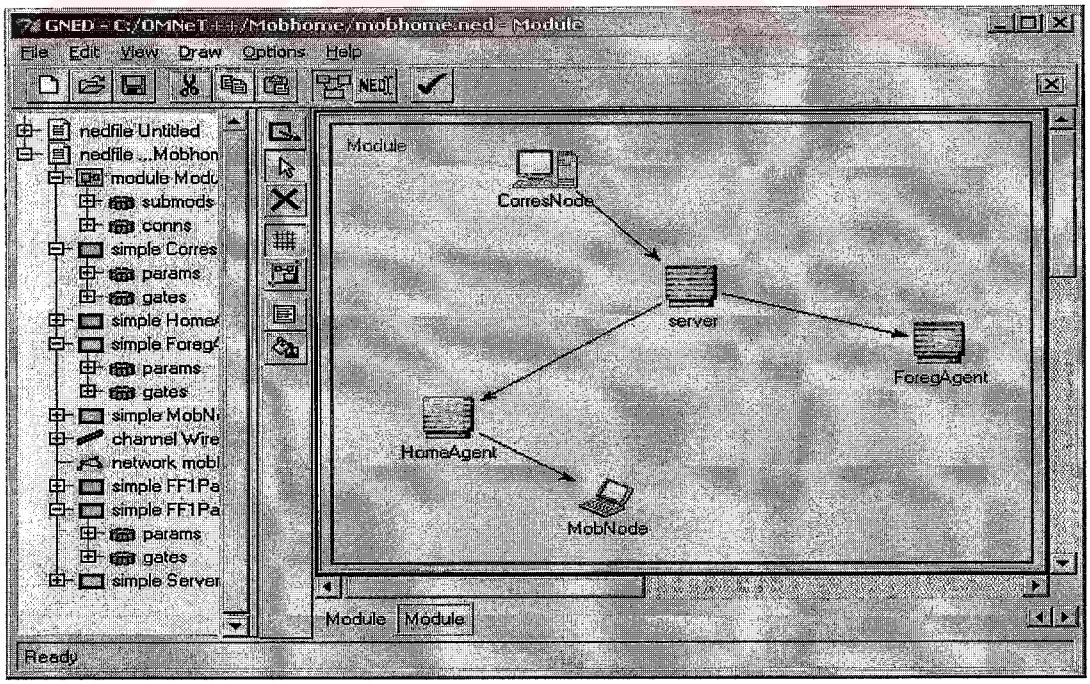


Figure 8.1: Structure of Mobile IP when mobile node in home network.

If the mobile node is within the domain of its own home agent, i.e., in the home network, the mobile node communicates with the correspondent node through home agent in usual routing manner. If the mobile node moves to the domain of a foreign network, the home agent will encapsulate the entire packet with the mobile node's new care of address and directs it to the mobile host. This encapsulation process is called "tunneling". When the Foreign Agent receives the encapsulated IP packet, it decapsulates the IP packet end extracts the original IP packets, and then deliver to the mobile node using the wireless route agent.

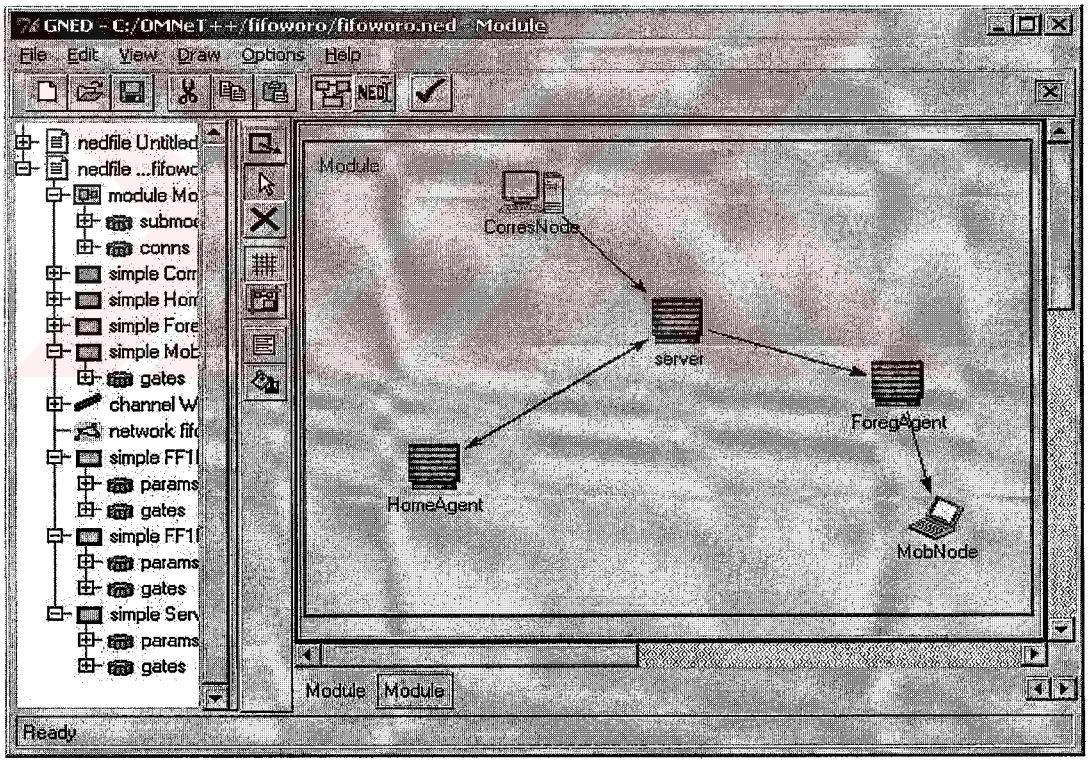


Figure 8.2: Structure of Mobile IP with and without route optimization.

In mobile IP with route optimization, when the mobile node's home agent intercepts a datagram from correspondent node and tunnels it to the mobile node, the home agent decides that the correspondent node has no binding cache entry for the

destination mobile host. The home agent then will send a Binding Update (a message indicating a mobile node's current mobility binding, and particularly its care of address) message to the correspondent node. Upon receiving a Binding Update message from the home agent of the mobile node, Correspondent node will tunnel the datagram to the mobile node's care of address directly. Structure of Mobile IP with Route Optimization in OMNeT++ is shown Figure 8.2.

8.2 Implement The Mobile IP With And Without Route Optimization

This experiment discusses the implementation details of route optimization and without route optimization of mobile IP using both text descriptions and flowcharts. The experiment first discusses the main challenges encountered, then the design of implementation of binding update message. Finally, the implementation details in the source code level in OMNeT++.

The following Figure 8.3 is a vivid overview of the basic concept of the mobile IP without route optimization, and Figure 8.4 is a vivid overview of the basic concept of the mobile IP with route optimization.

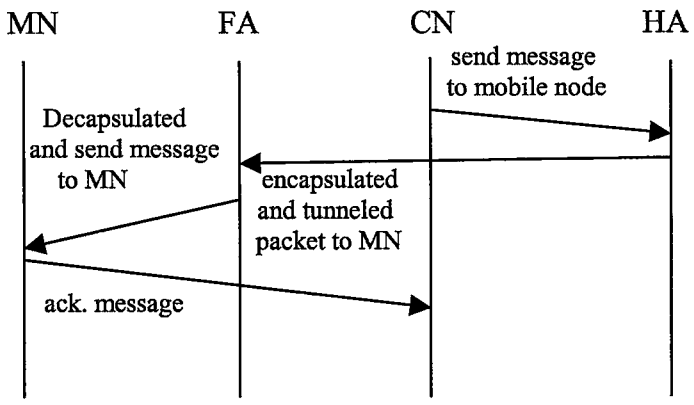


Figure 8 3: Overview of mobile IP without route optimization.

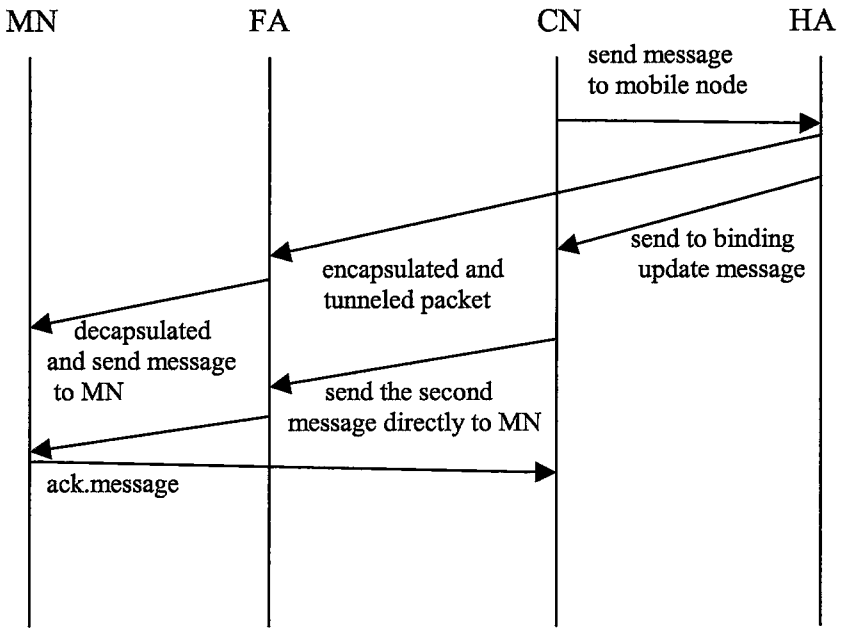


Figure 8.4 : Overview of mobile IP with route optimization.

These states are implement-using OMNeT++. Overview analyzes the source code of the current mobile IP implementation in OMNeT++. The implementation was completely done in C++ using OMNeT ++ simulation library and its NEDC language. The NEDC language defines the topology of the network to be tested. The C++ classes define the module behaviors. Due to the nature of the NEDC language, there is no necessity for a complicated concept. The NEDC definition has been split in to two definition files, the first one holding the basic module and connections types. The C++ files hold the corresponding classes. As customary, the implementation and the headers are separately stored in the corresponding cc- and h-files. These corresponding classes are mobile node, home agent, correspondent node and server. Each class has virtual functions, activity () and finish (). OMNeT++ starts these two functions automatically for each module. The first activity () is started after the creation of each module and the second finish (), before its

destruction. OMNeT++ destructs each module automatically when its activity () function returns.

CHAPTER 9

DESIGN OF THE SIMULATION FOR PERFORMANCE ANALYSIS

This simulation verifies the correctness the modification of mobile IP with and without route optimization and when mobile node in its home network. The goal of the simulation is performance analysis of mobile IP comparing end-to-end delay for three different scenarios. The experiment assumed that each scenario has only one mobile node, one home agent, one foreign agent and one wired node (server). Architecture of the simulation shown in Figure 8.2. In simulation it is assumed that the correspondent node, the home agent and the foreign agent are connected by wired links, and the mobile node attached to the foreign agent through wireless link. The wired and wireless networks are simulated using 10Mbps duplex links and 1.2 Mbps duplex links respectively [9], [12], [13], [14], [15]. The scenario assumed no loss of signaling or data packets are simulated. Data is voice in simulation [16]. Pulse code modulation is used for packetization process with 8 Kbps transmission speeds. In simulation it is assumed that shortest packet arrival time for pulse code modulation (PCM) format is 20ms [9], and message type is a UDP packet. In the UDP case, the source sends one 200-byte packet every 20 millisecond. In the simulation these parameters are used. The delay value which is calculated according to the other valid parameters for Voice transmission through the Packet network, using 1 switch, 2

routers and 10 Mbps Line speed, is taken from the simulation program written by Barbaros Preveze *, in his Msc thesis work.

In the first simulation the situation of the mobile node in home network is considered. This state is shown in Figure 8.1 as the correspondent node sends packets to mobile node; the received packet by mobile node is sent to the mobile node's home address directly. In this state end-to-end packet delay depends on time between the correspondent node-home agent and home agent-mobile node. In this scenario there are 800 numbers of packets, which are sent to the mobile node. The end-to-end delay (T_{ee}) is defined in equation 9.1 [17].

$$T_{ee,h} = (T_{cn,pack}) + (T_{cn,seri}) + (T_{cn,quin}) + (T_{cn->ha}) + (T_{ha,proc}) + (T_{ha->mn}) + (T_{jit}) + (T_{mn,dec}) \quad (9.1)$$

" $T_{cn,pac}$ " is the packetization delay ,the work assumed PCM packet format is 20ms for 200 byte packet. "$T_{cn,seri}$ " is serialization delay. The speed of line for calculation of the serialization delay in wired network among each nodes is taken as 10 Mbps. "$T_{cn,quin}$" is queuing delay that is taken from calculation of voice transmission delay. "$T_{cn->ha}$" is the time for a packet to travel from correspondent node to home agent, the length of each wired cable between the nodes 400m. Delay is calculated from the cable length [in meter]. In glass, light travels at approximately 200m/microseconds [18]. 2us accuracy is used in calculations. The "$T_{ha,proc}$" is the processing time (intercepting) at in the home agent. It includes switching delay and serialization delay. The speed of line for calculation of the serialization delay in wireless parts of the network is 1.2 Mbps. And the $T_{ha->mn}$ is the time for a packet to travel from home agent and mobile node. The distance between the mobile node and the home agent is changing random between the 100m and 500m. "T_{jit}" is jitter delay that is taken from

* Barbaros Preveze, VoIP End-to-End Simulation, Msc Thesis (in progress) Çankaya University, May 2003.

calculation of voice transmission delay. "$T_{mn,dec}$" is decoding delay that is taken from calculation of voice transmission delay. The simulation result is shown in Figure 9.1.

In the second scenario the case in which mobile node is in foreign network is considered and the performance analysis is done without route optimization. This architecture is shown in Figure 8.2. When corresponding node sends the packets to the mobile node each packet is sent to the mobile node though the home agent. In this case end-to-end packet delay depends on time between the correspondent node and the home agent, home agent-foreign agent and foreign agent-mobile node. The end-to-end delay is defined in equation 9.2.

$$T_{ee,wout} = (T_{cn,pack}) + (T_{cn,seri}) + (T_{cn,quin}) + (T_{cn->ha}) + (T_{ha,proce}) + (T_{ha->fa}) + (T_{fa,proc}) + (T_{fa->mn}) + (T_{jit}) + (T_{mn,dec}) \quad (9.2)$$

"$T_{ha,proce}$" includes encapsulated time, switching delay and serialization delay. The encapsulated time is 270us [12]. "$T_{ha->fa}$" is the time for a packet to travel from correspondent node to home agent, the length of each wired cable between the nodes 400m. Delay is calculated from the cable length [in meter]. In glass, light travels at approximately 200m/microseconds. 2us accuracy is used in calculations. "$T_{fa,proc}$" time includes decapsulated time, switching delay and serialization delay. Decapsulation delay is 160us [12]. "$T_{fa->mn}$" is the time for a packet to travel from home agent and mobile node. The distance between the mobile node and the home agent is changing random between the 100m and 500m. The simulation result is shown in Figure 9.2.

In the third scenario mobile IP with route optimization is considered. In this scenario the binding update messages are used for route optimization. In this simulation, the first packet send to the mobile node same as protocol of mobile IP without route optimization. Other packet end-to-end delays depend on time between correspondent

node-foreign agent and foreign agent-mobile node the end-to-end delay is defined in equation 9.3.

$$T_{ee} = (T_{cn,pack}) + (T_{cn,seri}) + (T_{cn,quin}) + (T_{cn->fa}) + (T_{fa,proc}) + (T_{fa->mn}) + (T_{jit}) + (T_{mn,dec}) \quad (9.3)$$

The simulation result is shown in Figure 9.3.

In mobile IP with route optimization, end-to-end packet delay ($T_{ee,wout,n}$) is calculated according to equation (9.4) for n packets. And also in mobile IP with route optimization, end-to-end packet delay ($T_{ee,wrout,n}$) is calculated according to equation (9.5) for n packets.

$$T_{ee,wout,n} = n * (T_{ee,wout}) \quad (9.4)$$

$$T_{ee,wrout,n} = T_{ee,wout} + (n-1) * (T_{ee}) \quad (9.5)$$

If end-to-end packet delay in mobile IP without route optimization is compared with the end-to-end packet delay in mobile IP with route optimization there are some extra delay in mobile IP without rout optimization. This delay ($T_{ext,delay,n}$) is calculated according to equation (9.6) for n packets. And also the delay ($T_{ext,delay}$) is calculated according to equation (9.7) for one packet.

$$T_{ext,delay,n} = (T_{ee,wout,n}) - (T_{ee,wrout,n})$$

$$T_{ext,delay,n} = n * (T_{ee,wout}) - (T_{ee,wout} + (n-1) * (T_{ee}))$$

$$T_{ext,delay,n} = (n-1) * (T_{ee,wout}) - (n-1) * (T_{ee})$$

$$T_{ext,delay,n} = (n-1) * ((T_{ee,wout}) - (T_{ee}))$$

$$T_{ext,delay,n} = (n-1) * ((T_{cn->ha}) + (T_{ha,proce}) + (T_{ha->fa}) - (T_{cn->fa}))$$

and $(T_{cn->ha}) = (T_{cn->fa})$

$$T_{ext,delay,n} = (n-1) * ((T_{ha,proce}) + (T_{ha->fa})) \quad (9.6)$$

$$T_{ext,delay} = ((n-1) / n) * ((T_{ha,proce}) + (T_{ha->fa})) \quad (9.7)$$

9.1 Simulation Result For Performance Analysis In Mobile IP With Omnet++

The simulation result of the first scenario is shown in Figure 9.1. Horizontal is the number of packet axis, vertical is the end-to-end packet delay, and the curve is packet end-to-end delay of when mobile node is in home network. Also in Table 9.1 average end-to-end packet delay, standard deviation, and total end-to-end simulation delay for 200 byte packet sizes and different number of messages while mobile node in home network is shown.

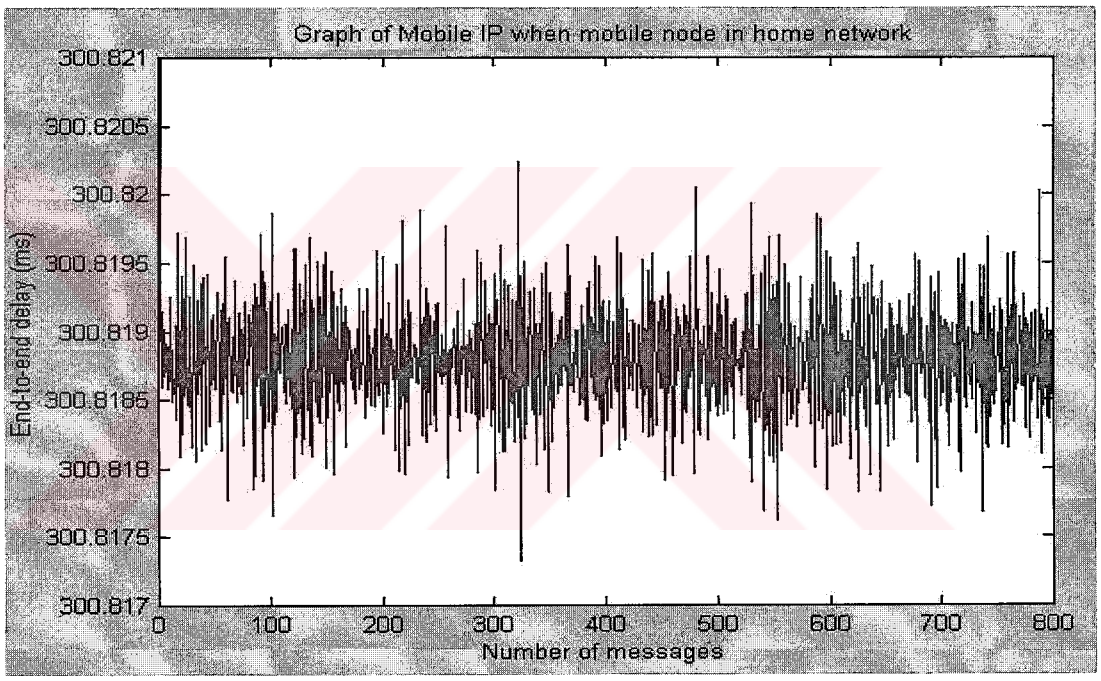


Figure 9.1: Simulation result of mobile node in home network.

From the above figure, it is seen that the minimum end-to-end delay of simulation.

Table 9.1: Simulation results of Mobile IPv4 when mobile node in home network for 200-byte packet.

Number of messages	800	2000	8000
Average end-to-end delay (sn)	0.300819	0.300812	0.300819
Standard deviation (sn)	$5.7126*10^{-6}$	$3.62826*10^{-6}$	$1.851118*10^{-6}$
Total end-to-end delay (sn)	240.65524	601.63796	2406.5506

The simulation result of second scenario is as Figure 9.2, horizontal is the number of message, vertical is end-to-end packet delay, and curve is shown end-to-end delay for mobile IPv4 without rote optimization. Table 9.2 is shown of simulation results; average end-to-end delay, standard deviation and total end-to-end delay for mobile IPv4 without route optimization.

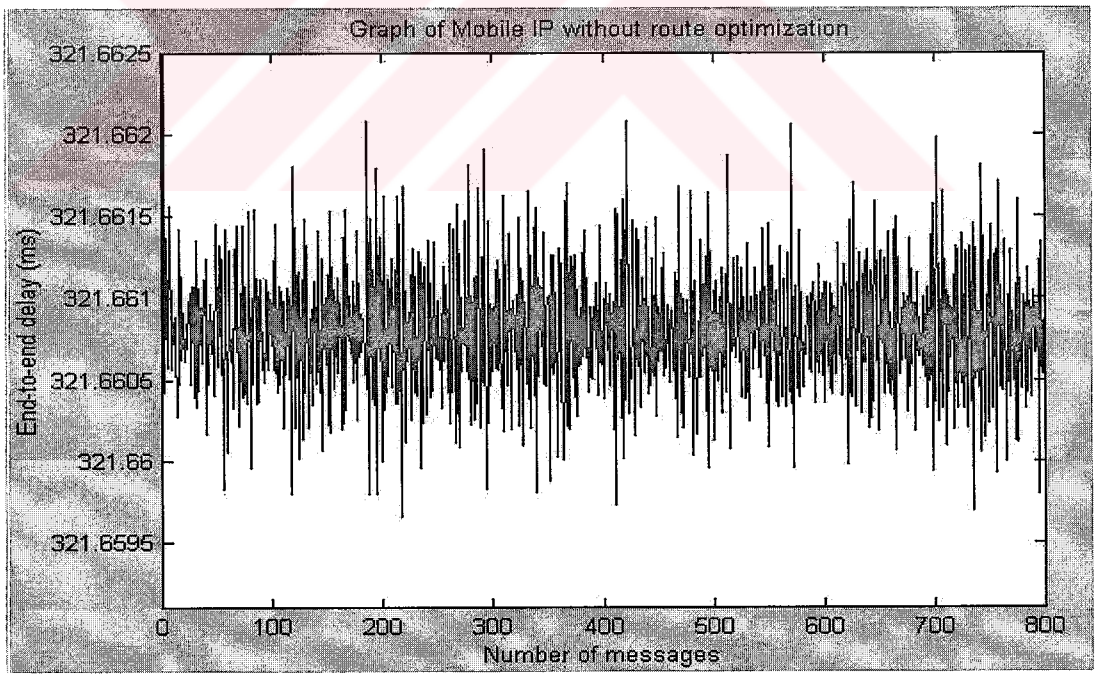


Figure 9.2:Simulation result of mobile IPv4 without route optimization.

Table 9.2: Simulation results of mobile IPv4 without route optimization for 200-byte packet

Number of messages	800	2000	8000
Average end-to-end delay (sn)	0.321687	0.321671	0.321663
Standard deviation (sn)	0.0007491	0.000473772	0.000236886
Total end-to-end delay (sn)	257.34984	643.357	2573.3074

From the above figures, it is clear that minimum end-to-end delay of packets when mobile node in home network is quite smaller than that of packets sent using mobile IPv4 without route optimization, as the mobile node moves to foreign network.

The simulation result of the third scenario is as Figure 9.3. Horizontal is the number of messages, vertical is the end-to-end packet delay, and the curve is the end-to-end packet delay when route optimization is used. Also, in Table 9.3 simulation results of average end-to-end delay, standard deviation, and total end-to-end-delay for mobile IPv4 with route optimization is shown.

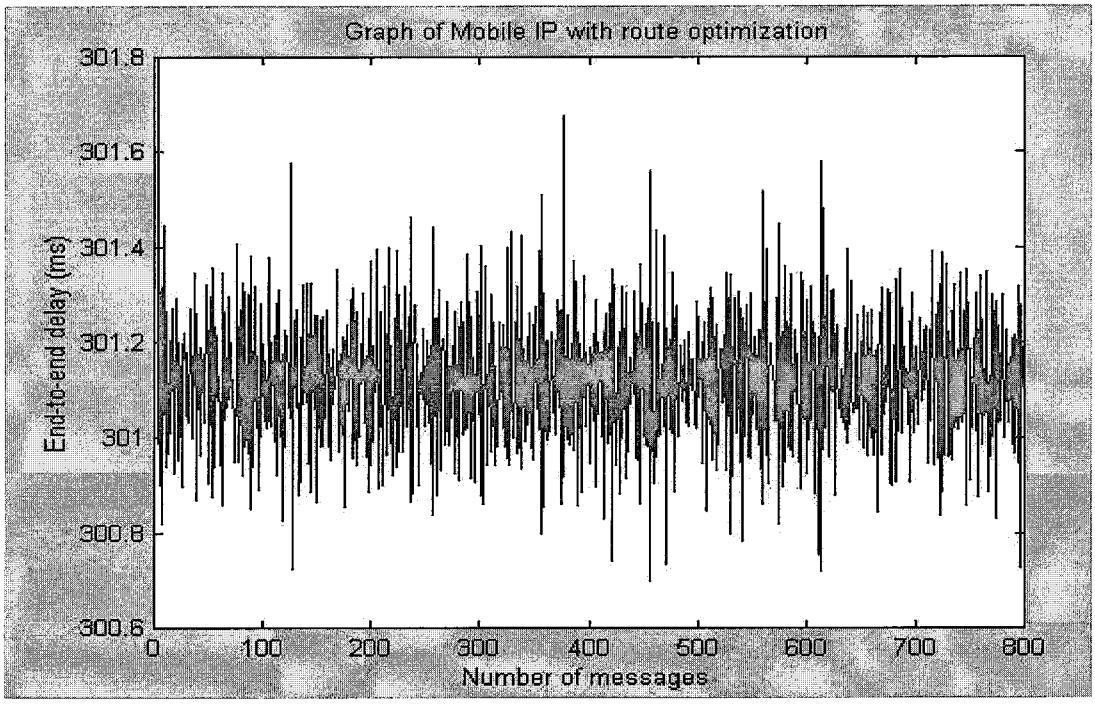


Figure 9.3: Simulation result of mobile IPv4 with route optimization.

Table 9.3: Simulation results of mobile IPv4 with route optimization for 200-byte packet.

Number of messages	800	2000	8000
Average end-to-end delay (sn)	0.301227	0.301163	0.301131
Standard deviation (sn)	0.00224199	0.00148572	0.000753375
Total end-to-end delay (sn)	240.98284	602.3268	2409.0464

From the above figures, minimum end-to-end delay of packets with route optimization is approximately equal that of the packets when mobile node in home network and quite smaller than that of the packets sent without route optimization. Comparing the average delay curves, they justify thesis goal that the end-to-end packet delay decrease drastically when mobile IPv4 with Route Optimization is enabled.

The thesis has proved by comparing the end-to-end packet delay curves and minimum delay, The Route Optimization successfully eliminates the effect of "Triangle routing" and improves the network efficiency by lowering down the end-to-end packet delay.

CHAPTER 10

CONCLUSION & FUTURE WORK

After the literature survey of the mobile IPv4 protocol and Route Optimization in mobile IPv4, thesis modifies and extends the mobile IPv4 in OMNeT++ to enable the Route Optimization. Simulations have been done and justified by thesis extension. All initial project goals have been fulfilled. Although this implementation is only part of the whole Route Optimization scheme, it is effective and sufficient to demonstrate the significant of Route Optimization in mobile IPv4. Through the simulations, the results are already self-explained.

The future work could be done all four-route optimization messages in OMNeT++ and compare them with other route optimization proposes, such as reverse routing. [19]. And also the buffering effect in route optimization is another search topic that is under investigation.

REFERENCES

[1] Charles E.Perkins, Mobile IP Design Principles And Practices, Addision Wesley Longman, 2nd Printing, January 1998, Page 27-169.

[2] http://www.mobilenetwork.dk/maindoc/node49.html

[3] http://www.zvon.org/tmRFC/RFC3344/Output/chapter2.html#sub2.

[4] Andrew T. Campbell," E6951 Wireless and Mobile Networking Mobile IP", University of Columbia, **http://www.comet.columbia.edu\~campbell**, 2000.

[5] David A. Maltz, David B. Johnson, The CMU Monarch Project IETF Mobile Ipv4 Implementation User's Guide,Computer Science Department Carnegie Mellon Universty 500 Forbes Avenue Pittsburg, PA 15213-3891, www.monarch.cs.rice.edu/ftp/monarch/mobileip/users-guide.ps, June 27,1997.

[6] C.Perkins, Request For Comments 2004, Minimal Encapsulation Within IP, standart track, http://www.faqs.org/rfcs/rfc2004.html, Oct.1996.

[7] Eric Ha, Ryan Retting, Rachel Chang. Mobile IP, Winter X564, http://www.csc.calpoly.edu/~husmith/CSC564-Winter02/Mobile_IP_paper.pdf, February 2002.

[8] C. Perkins and D. Johnson, Internet Draft-Rote Optimization in mobile IP, http://www.ietf.org/internet-drafts/draft-ietf-mobileip-optim-09.txt.March work in progress, November 1997.

[9] Babak Ayani, Smooth handoff in Mobile IP, University of California in Berkeley, Department of Microelectronics and Information Technology at KTH, master thesis, May 2002, http://www.imit.kth.se/courses/html/exjobb/Projects/Karlsson,

[10] Debalina Grosh, ACM Crossroads student magazine, Mobile IP, www.acm.org/crossroads/espanol/xrds72/-8, Winter 2000.

[11] Andras Varga, OMNeT++ Discrete Event Simulation System, Version 2.1,Technical University of Budapest Faculty of Electrical Engineering and Informatics Department of Telecommunications, User Manual, May 24 2001.

[12] Charles E. Perkins, Kuang-Yeh Wang, Optimized Smooth Handoffs in Mobile IP, Sun Microsystems, Inc.901 San Antonia Rd.Palo Alto, CA 94303, U.S.A., http://citeseer.nj.nec.com/perkins99optımızed.html, 1999.

[13] Andreas Festag, Performance Evaluation of Mobile IP w/ and w/o Hierarchical Foreign Agents: Goals, Metrics, Parameters and Tested Setup, TU Berlin,TKN, http://www.tkn.ee.tu-berlin.de/research/SeQoMo/seqomo_docu/D-MM-3.pdf, 19.02.2001.

[14] Cheng Lin Tan, Stephen Pink, Kin Mun Lye, A Fast Handoff Scheme for Wireless Networks, Computer Science &Electrical Engineering Lulea University of Technology, Centre for Wireless Communication National University of Singapore, http://citeseer.nj.nec.com/310996.html, 1999.

[15] J. Conover, Wireless LANs work their magic, Network Computing, http:/www.networkcomputing.com/1113/1113f2.html, July 10, 2000.

[16] Cheng Lin Tan, Stephen Pink, Kin Mun Lye, A Fast Handoff Scheme for Wireless Networks, Seattle, Washington, http://citeseer.nj.nec.com/310996, 20 August 1999.

[17] Jon-Olov Vatn, Improving Mobile IP handover performance, Laboratory of Telecomunication Systems Department of Teleinformatics Royal Institute of Technology Electum 204, 164 40 Kista,Sweden. http://www.citeseer.nj.nec.com/cachedpage/482475/1, 1999.

[18] http://www.wildpackets.com/compendium/EN/EN-Propa.html.

[19] P.Zhou and O. Yang, Reverse Routing: An Alternative to Mobile IP and ROMIP Protocols, Proceedings of 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Volume 1, pp.150-155. http://www.sfu.ca/~lcheu/study/885Presentation.pdf, 4 August 1996.

[20] Ericsson,"MobileIP", http://www.symbol.com/products/wireless/wireless_alliances_ericson.html, 3 July 1999,

[21] C.Perkins, Request For Comment 3344: IP Mobility Support for IPv4, http://rfc.sunsite.dk/rfc/rfc3344.html, August 2002.

[22] C.Y.Chen, Mobility With The Internet and IP Networks, University Of Birmingham, Msc project Dissertation, http://web.bham.ac.uk/chency/work/Mobile_IP.pdf, 23 August 1999.

[23] A.Hess,G.Schafer,"Performance Evaluation of AAA/Mobile IPAuthentication", Telecommunication Networks Group, Technische University at Berlin, Germany, http://www-tkn.ee.tu-berlin.de/publications/papers/pgts2002.pdf, September 2002.

[24] Peng Sun, Sam Y. Sung, Enhancement of Binding Update for Mobile IP, Deparment of Computer Science National University of Singapore, http://www.comp.nus.edu.sg/~ssung/publications1017.pdf, 2002.

[25] J.Solomon, Motorola, Request For Comment 2005, Applicability Statement For IP Mobility Support, http://rfc.sunsite.dk/rfc/rfc2005.html, October 1996.

[26] G.Montenegro,Editor Sun Microsystems, Request For Comment 2344, Reverse Tunneling For Mobile IP, http://rfc.sunsite.dk/rfc/rfc2344.html, May 1998,

[27] Rajib Ghosh,George Varghese, Fault-Tolerant Mobile IP, Department of Computer Science Campus Box 1045 Washington University One Brookings Drive St. Louis, MO 63130-4899, http://citeseer.nj.nec.com/correct/513163, April 29, 1998.

[28] Maoyu Wang, Implementation and Performance Evaluation of Route Optimization in Mobile IP, School of computer Science Carleton University, Ottawa, Ontario, Master Thesis, http://www.sce.carleton.ca/wmc/chameleon/mc/MaoyuThesis.pdf, Nov 1,2001,

[29] Dan Forsberg, Communication availability with Mobile IP in Wireless LANs, Helsinki University of Technology, Master Thesis, http://www.cs.hut.fi/Research/Dynamics/publications.html, March 2000.

[30] C.Perkins, Request For Comment 2003:IP Encapsulation Within IP, http://rfc.sunsite.dk/rfc/rfc2003.html, October,1996,

[31] S. Hanks NetSmith, Ltd, T. Li, D. Farinacci, P. Traina, cisco systems, RFC 1701, Generic Record Encapsulation, http://www.faqs.org/rfcs/rfc/701.html, October 1994.

[32] ODTÜ-BIDB: Network Association, http:/www.bidb.odtu.edu.tr/index.php?go=ng&sub=802_11_b, 2003

APPENDICE

SOURCE CODE OF THE APPLICATION

//MOBILE IPv4 with Route Optimization

//Correspondent node cpp.

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
#include"genpak.h"
#include "gen1.h"

int k;

long double simtime;

long  seed;

  class CorresNode: public cSimpleModule
  {

public:
    Module_Class_Members(CorresNode,cSimpleModule,16384*6)

virtual void activity();
virtual void finish();

 };

Define_Module (CorresNode);

void CorresNode::activity()
```

```
  {

int num_messages = par("num_messages");
cPar& ia_time = par("ia_time");
cPar& msg_length = par("msg_length");

 data* serap=new data();
  serap->A.gt='h';

 for (int i=0; i<num_messages; i++)

   {

      char msgname[32];
      sprintf( msgname, "job-%d", i);

      ev << "Generating " << msgname << endl;

      cMessage *msg=new cMessage(msgname);

       msg->setContextPointer((void *)serap);

       msg->setLength((long) msg_length );

       msg->setTimestamp(simtime);

             if (serap->A.gt=='h')

             {

      send( msg,"out" );

      cMessage *msgr = receive();

      serap=(data*)msgr->contextPointer();

      wait((double) ia_time );

             }
             else

             {

         wait((double) ia_time );
        serap->A.gt='f';
         msg->setContextPointer((void *)serap);

                    send( msg,"out");
           }
```

```
        }
    }

void CorresNode::finish()
{
   ev << "*** Module: " << fullPath() << "***" << endl;
   ev << "Stack allocated:      " << stackSize() << " bytes";
   ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
endl;
   ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
}


//Server cpp
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "genpak.h"


class Serverwro: public cSimpleModule
{
   Module_Class_Members(Serverwro,cSimpleModule,16384*5)

   virtual void activity();
   virtual void finish();

};

Define_Module( Serverwro );

void Serverwro::activity()
{
int coun, deccount;
            coun=0;
            deccount=0;
            int k;
   double bits_per_sec= par("bits_per_sec");
   cPar& ia_time = par("ia_time");

   data* ser1=new data();

    for( ; ; )

   {
```

```
        cMessage *msg1 = receive();

        ser1=(data*)msg1->contextPointer();

        k= msg1->length();

                if(ser1->A.gt=='h' && k==1600)

                {

                 send(msg1,"outh");

                 wait((double) ia_time );

                }

                else if ( ser1->A.gt=='f' && k >> 1600 )

                {

               send( msg1,"outf");

                wait((double) ia_time );

                        }

                 else   if ( ser1->A.gt=='s' && k==64)

                {

          wait((double) ia_time );
                ser1->A.gt='c';
          msg1->setContextPointer((void *)ser1);
          send( msg1,"outc");

                 }
        }
}

 void Serverwro::finish()

{

  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}
```

//Home agent cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "genpak.h"

class HomeAgent: public cSimpleModule
{
  Module_Class_Members(HomeAgent,cSimpleModule,16384*4)

  virtual void activity();
  virtual void finish();

};

 Define_Module( HomeAgent );

void HomeAgent::activity()
{
int coun, deccount;
            coun=0;
            deccount=0;
            double bits_per_sec= par("bits_per_sec");

             cPar& ia_time = par("ia_time");
     data* ser1=new data();

   for(;;)

  {
    cMessage *msg1 = receive();

    ser1=(data*)msg1->contextPointer();
     long k= msg1->length();

                if(ser1->A.gt=='h')
                {

                 ser1->A.gt='f';

                msg1->setContextPointer((void *)ser1);
                msg1->setLength(k+64);

                wait(2.7);   //encapsulation time
                send(msg1,"out");

                wait((double) ia_time );
```

```
                char msgname[64];
                 sprintf( msgname, "binding_update");
                 ev << "Generating " << msgname << endl;
                cMessage *msg=new cMessage(msgname);
                 ser1->A.gt='s';
                 msg->setContextPointer((void *)ser1);
                 msg->setLength(64);
                 wait((double) ia_time );
                  send( msg,"out" )              }
      else
                {
                send( msg1,"outf");
                wait( 0.1);
}
          wait((double) ia_time );
                 }
 }
  void HomeAgent ::finish()
{
  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}
```

//Foreign Agent CPP

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "genpak.h"

class HomeAgent: public cSimpleModule
{
   Module_Class_Members(HomeAgent,cSimpleModule,16384*4)
```

```
   virtual void activity();
   virtual void finish();

};

 Define_Module( HomeAgent );

 void HomeAgent::activity()
{
int coun, deccount;
             coun=0;
             deccount=0;
double bits_per_sec= par("bits_per_sec");

      cPar& ia_time = par("ia_time");
      data* ser1=new data();

for(;;)

   {
     cMessage *msg1 = receive();
                  ser1=(data*)msg1->contextPointer();
                  long k= msg1->length();

                  if(ser1->A.gt=='h')

{

            ser1->A.gt='f';

            msg1->setContextPointer((void *)ser1);
            msg1->setLength(k+64);

            wait(2.7);   //encapsulation time
            send(msg1,"out");

            wait((double) ia_time );

           char msgname[64];

            sprintf( msgname, "binding_update");

            ev << "Generating " << msgname << endl;


            cMessage *msg=new cMessage(msgname);

      ser1->A.gt='s';
```

```
        msg->setContextPointer((void *)ser1);
        msg->setLength(64);

        wait((double) ia_time );

        send( msg,"out" );
                    }

        else
                    {
                    send( msg1,"outf");

                    wait( 0.1);
                    }
                        wait((double) ia_time );
            }
}

 void HomeAgent ::finish()

{

  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}
```

//Mobile node cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
#include"genpak.h"
#include "sink1.h"
 int s;
long double l;
long double delay[802];
long double t[802];
  FILE* fp;
 FILE* fs;

  Define_Module( MobileNode );

  void MobileNode::activity()
  {
```

```
 qstats.setName("queuing time stats");
 cOutVector qtime("queueing time vector");

long double t_delay=0;
long double m=0;
long double delay1=0;
delay[1]=0.00000000000;

t[0]=0;
 s=1;

 if( (fp  = fopen( "c:\\Omnet++\\fifowro\\mdata.txt", "wb+" )) == NULL )
  printf( "The file 'data' was not opened\n" );

 fseek(fp,0,0);
 if( (fs  = fopen( "c:\\Omnet++\\fifowro\\sdata.txt", "wb+" )) == NULL )
  printf( "The file 'data' was not opened\n" );

  fseek(fs,0,0);

for (;;)

  {

        cMessage *msgr = receive();

     msgr->setTimestamp();

            long double l=msgr->timestamp();

                  t[s]=l;
     fprintf(fs,"%f",l);
     fprintf(fs,"%c",'\n');

                  delay[s]=t[s]-t[s-1];

      ev<<"simtime="<<simtime<<endl;
      ev<<"t[i]="<<t[s]<<endl;
      ev<<"delay[i]="<<delay[s]<<endl;
      fprintf(fp,"%f",delay[s]);
          fprintf(fp,"%c",'\n');
      ev<<"i="<<s<<endl;

ev << "Received " << msgr->name() << ", queueing time: " << l << "sec" <<
    endl;
     qtime.record( delay[s]/10000 );
     qstats.collect( delay[s]/10000 );
        s=s+1;
```

```
                    ev<<" about to close file 1 ";
                }

  fclose(fp);
  fclose(fs);

  }
void MobileNode::finish()

{
   ev << "*** Module: " << fullPath() << "***" << endl;
   ev << "Total jobs processed: " << qstats.samples() << endl;
   ev << "Avg queueing time:    " << qstats.mean() << endl;
   ev << "Max queueing time:    " << qstats.max() << endl;
   ev << "Standard deviation:   " << qstats.stddev() << endl;
   ev << endl;
   ev << "Stack allocated:      " << stackSize() << " bytes";
   ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
  endl;
   ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
}
///OMNeT initial file,

[General]
network = fifowro

random-seed = 1
sim-time-limit = 100000000s
cpu-time-limit= 1800000s
ini-warnings = no

[Cmdenv]
runs-to-execute = 1,2
#express-mode = yes
module-messages = yes
event-banners = yes

[Tkenv]
default-run=1
use-mainwindow = yes
print-banners = yes
slowexec-delay = 300ms

[Parameters]
fifowro.gen.num_messages = 5000

[Run 1]
fifowro.gen.ia_time = exponential( 1 )
fifowro.gen.msg_length = intuniform( 5, 10)
```

```
fifowro.fifo.bits_per_sec = 10
output-vector-file = fifowro-r1.vec

[Run 2]
fifowronet.gen.ia_time = exponential( 0.7 )
fifowronet.gen.msg_length = intuniform( 5, 10)
fifowronet.fifo.bits_per_sec = 10
output-vector-file = fifowro-r2.vec
```

//MOBILE IP WITHOUT ROUTE OPTIMIZATION
//Correspondent node cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
#include"genpak.h"
#include "gen1.h"

int k;

long double simtime;

long  seed;

  class CorresNode: public cSimpleModule
  {

   public:
    Module_Class_Members(CorresNode,cSimpleModule,163842*4)

   virtual void activity();
   virtual void finish();

 };

  Define_Module (CorresNode);

  void CorresNode::activity()
  {

   int num_messages = par("num_messages");
   cPar& ia_time = par("ia_time");
   cPar& msg_length = par("msg_length");
```

```
            data* serap=new data();

    serap->A.gt='h';

    for (int i=0; i<( (long) msg_length /1600)*num_messages; i++)

    {
      char msgname[32];
      sprintf( msgname, "job-%d", i);

      ev << "Generating " << msgname << endl;

      cMessage *msg = new cMessage( msgname );

       msg->setLength(1600);

      wait(600 );

      msg->setTimestamp(simtime);

              msg->setContextPointer((void *)serap);

              send( msg,"out" );

              wait((double) ia_time );
            } }

 void CorresNode::finish()
 {
   ev << "*** Module: " << fullPath() << "***" << endl;
   ev << "Stack allocated:      " << stackSize() << " bytes";
   ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
   endl;
   ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
 }
```

//Server cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "genpak.h"

class Server: public cSimpleModule
{
  Module_Class_Members(Server,cSimpleModule,16384*4)
```

```
   virtual void activity();
   virtual void finish();

};

 Define_Module( Server );

void Server::activity()
{
 int couna=0;
 int deccounta=0;

int coun, deccount;
            coun=0;
            deccount=0;
            int k;
double bits_per_sec= par("bits_per_sec");

cPar& ia_time = par("ia_time");
 data* ser1=new data();

   for(;;)

   {

      cMessage *msg1 = receive();

            if (msg1->hasBitError()==true)    //(crc2==crc1)

                  deccounta=deccounta+1;

            else

        couna=couna+1;

             ev<<"false mesage's number is"<<deccounta<<endl;
             ev<<"true mesage's number is"<<couna;

                 wait((double) ia_time );

                  ser1=(data*)msg1->contextPointer();

                  k= msg1->length();

                  if(ser1->A.gt=='h' && k==1600)

                  {
                           send(msg1,"outh");
```

```
                }

               else if (k >> 1600)

                {

                send( msg1,"outf");
}
                    }
 }

  void Server ::finish()

{
  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;
}
```

//Foreign Agent cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include"genpak.h"
#include "fifo2.h"


void FF1Abstract_Fifo::activity()
{

   msgServiced = NULL;
   endServiceMsg = new cMessage("end-service");
   queue.setName("queue");

   data* ser2=new data();

    for(;;)

   {

      cMessage *msg = receive();

      long k=msg->length();
```

```
        wait(1.6); //encapsulation time

        msg->setLength(1600);

        ev<<msg->length()<<endl;


if (msg==endServiceMsg)

        {
          endService( msgServiced );
          if (queue.empty())
    {
             msgServiced = NULL;
          }


else

          {
             msgServiced = (cMessage *) queue.pop();
             simtime_t serviceTime = startService( msgServiced );
             scheduleAt( simTime()+serviceTime, endServiceMsg );

          }

        }

        else if (!msgServiced)

        {
          arrival( msg );
          msgServiced = msg;
          simtime_t serviceTime = startService( msgServiced );
          scheduleAt( simTime()+serviceTime, endServiceMsg );

        }

        else

        {
          arrival( msg );
          queue.insert( msg );
        }
      }
}
```

```
void FF1Abstract_Fifo::finish()

{
  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}

Define_Module( FF1Packet_Fifo );

simtime_t FF1Packet_Fifo::startService(cMessage *msg)
{
   ev << "Starting service of " << msg->name() << endl;
   return par("service_time");
 }

void FF1Packet_Fifo::endService(cMessage *msg)
{
   ev << "Completed service of " << msg->name() << endl;
   send( msg, "out" );
}

//---------------------------------------------

Define_Module( ForegAgent);

simtime_t ForegAgent::startService(cMessage *msg)
{
   ev << "Starting service of " << msg->name() << endl;
   return msg->length() / (double)par("bits_per_sec");
}

void ForegAgent::endService(cMessage *msg)
{
   ev << "Completed service of " << msg->name() << endl;
   send( msg, "out" );
}
```

// Mobile Node

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
```

```
#include"genpak.h"

#include "sink1.h"

int s;

 int coun=0;
 int deccount=0;

long double l;

long double delay[8002];

long double t[8002];

  FILE* fp;

  FILE* fs;

Define_Module( MobNode );

void MobNode::activity()
{

    qstats.setName("queuing time stats");
   cOutVector qtime("queueing time vector");

  long double t_delay=0;
  long double m=0;
  long double delay1=0;
  delay[0]=0.00000000000;


   t[0]=0;
            s=1;

  if( (fp  = fopen( "c:\\Omnet++\\fifoworo\\pdata.txt", "wb+" )) == NULL )
     printf( "The file 'data' was not opened\n" );

 fseek(fp,0,0);

   if( (fs  = fopen( "c:\\Omnet++\\fifoworo\\sdata.txt", "wb+" )) == NULL )
     printf( "The file 'data' was not opened\n" );

 fseek(fs,0,0);

   for(;;)
   {
```

```
        cMessage *msg = receive();

         msg->setTimestamp();

            if (msg->hasBitError()==true)
                                deccount=deccount+1;

            else

        coun=coun+1;
             ev<<"false mesage's number is"<<deccount<<endl;
                  ev<<"true mesage's number is"<<coun<<endl;

      long double d=msg->timestamp();


                  t[s]=d;
       fprintf(fs,"%f",d);
       fprintf(fs,"%c",'\n');

       delay[s]=t[s]-t[s-1];
                  ev<<"d="<<d<<endl;
                  ev<<"delays="<<delay[s];
       fprintf(fp,"%f",delay[s]);
       fprintf(fp,"%c",'\n');


       ev << "Received " << msg->name() << ", queueing time: " << d << "sec" <<
       endl;
       qtime.record( delay[s]/10000 );
       qstats.collect( delay[s]/10000 );
       delete msg;

        s=s+1;
}

 fclose(fp);
 fclose(fs);
 }

void MobNode::finish()

{
   ev << "*** Module: " << fullPath() << "***" << endl;

   ev << "Total jobs processed: " << qstats.samples() << endl;

   ev << "Avg queueing time:    " << qstats.mean() << endl;
```

```
    ev << "Max queueing time:    " << qstats.max() << endl;

    ev << "Standard deviation:   " << qstats.stddev() << endl;

    ev << endl;
    ev << "Stack allocated:      " << stackSize() << " bytes";
    ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
    endl;
    ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
}
#ifndef __FIFO_H

#define __FIFO_H

#include <omnetpp.h>

// FF1AbstractFifo : abstract base class for single-server queues

class FF1AbstractFifo : public cSimpleModule

{

 public:

    Module_Class_Members(FF1AbstractFifo,cSimpleModule,16384);

    cMessage *msgServiced;

    cMessage *endServiceMsg;

    cQueue queue;

    virtual void activity();

    virtual void finish();

    // hook functions to (re)define behaviour
    virtual void arrival(cMessage *msg) {}
    virtual simtime_t startService(cMessage *msg) = 0;
    virtual void endService(cMessage *msg) = 0;
};

// FF1PacketFifo : single-server queue with given service time
class FF1PacketFifo : public FF1AbstractFifo
{
 public:
    Module_Class_Members(FF1PacketFifo,FF1AbstractFifo,16384);

    virtual simtime_t startService(cMessage *msg);
```

```
    virtual void endService(cMessage *msg);
};

// FF1BitFifo : single-server queue with service time based on message length
class HomeAgent : public FF1AbstractFifo
{
 public:
   Module_Class_Members(HomeAgent,FF1AbstractFifo,16384);

   virtual simtime_t startService(cMessage *msg);

   virtual void endService(cMessage *msg);
};

//#endif

class ForegAgent : public FF1AbstractFifo
{
 public:
   Module_Class_Members(ForegAgent,FF1AbstractFifo,16384);

   virtual simtime_t startService(cMessage *msg);
   virtual void endService(cMessage *msg);
};

#endif

#ifndef __FIFO_H

#define __FIFO_H

#include <omnetpp.h>

// FF1AbstractFifo : abstract base class for single-server queues
class FF1Abstract_Fifo : public cSimpleModule
{
 public:
   Module_Class_Members(FF1Abstract_Fifo,cSimpleModule,16384);

   cMessage *msgServiced;
   cMessage *endServiceMsg;
   cQueue queue;

   virtual void activity();
   virtual void finish();

   // hook functions to (re)define behaviour
   virtual void arrival(cMessage *msg) {}
   virtual simtime_t startService(cMessage *msg) = 0;
```

```
    virtual void endService(cMessage *msg) = 0;
};

// FF1PacketFifo : single-server queue with given service time

class FF1Packet_Fifo : public FF1Abstract_Fifo
{
 public:
   Module_Class_Members(FF1Packet_Fifo,FF1Abstract_Fifo,16384);

   virtual simtime_t startService(cMessage *msg);
   virtual void endService(cMessage *msg);
};

class ForegAgent : public FF1Abstract_Fifo
{
 public:
   Module_Class_Members(ForegAgent,FF1Abstract_Fifo,16384);

   virtual simtime_t startService(cMessage *msg);
   virtual void endService(cMessage *msg);
};

#endif

#ifndef __FIFO_H

#define __FIFO_H

#include <omnetpp.h>

// FF1AbstractFifo : abstract base class for single-server queues

class FF1Abstract_Fifo : public cSimpleModule

{

 public:

   Module_Class_Members(FF1Abstract_Fifo,cSimpleModule,16384);

   cMessage *msgServiced;

   cMessage *endServiceMsg;

   cQueue queue;

   virtual void activity();
```

```
    virtual void finish();

    // hook functions to (re)define behaviour

    virtual void arrival(cMessage *msg) {}

    virtual simtime_t startService(cMessage *msg) = 0;

    virtual void endService(cMessage *msg) = 0;

};

// FF1PacketFifo : single-server queue with given service time
class FF1Packet_Fifo : public FF1Abstract_Fifo
{
 public:
   Module_Class_Members(FF1Packet_Fifo,FF1Abstract_Fifo,16384);

   virtual simtime_t startService(cMessage *msg);
   virtual void endService(cMessage *msg);
};

class ForegAgent : public FF1Abstract_Fifo
{
 public:
   Module_Class_Members(ForegAgent,FF1Abstract_Fifo,16384);

   virtual simtime_t startService(cMessage *msg);
   virtual void endService(cMessage *msg);
};

#endif

struct mplip
{
 char gt;
 int ip;
};

class data
{
            public:
   struct mplip A;
   long ds;
            char cr;
            int dz;
    data()
    {
              ds=12;
```

```
                dz=44;
                cr='d';};
#ifndef global_h
#define global_h

extern int crc1;
int    Crc16 (char   *ptr,int count);

extern long double d1;
extern long double d2;
extern long double totaldelay[20];
extern int p_size;
extern int k;
extern long double simtime;
extern long double d3;
extern long double t[8002];
extern long double delay[8002];
extern  fp;
#endif

#ifndef __SINK_H
#define __SINK_H

#include <omnetpp.h>

class MobNode : public cSimpleModule
{
  Module_Class_Members(MobNode,cSimpleModule,16384)

  cStdDev qstats; // needs to be accessed from finish() too

  virtual void activity();
  virtual void finish();
};

#endif
```

//MOBILE NODE IN HOMNETWORK
//Correspondent Host

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
#include"genpak.h"
#include "gen1.h"
```

```
int k;

long double simtime;

long  seed;

  class CorresNode: public cSimpleModule
  {

   public:
    Module_Class_Members(CorresNode,cSimpleModule,163842*5)

   virtual void activity();
   virtual void finish();
 };

  Define_Module (CorresNode);

  void CorresNode::activity()
  {

   int num_messages = par("num_messages");
   cPar& ia_time = par("ia_time");
   cPar& msg_length = par("msg_length");

              data* serap=new data();

    serap->A.gt='h';

      for (int i=0; i<( (long) msg_length /1600)*num_messages; i++)

    {

      wait(200);

       char msgname[32];
       sprintf( msgname, "job-%d", i);

       ev << "Generating " << msgname << endl;

       cMessage *msg = new cMessage( msgname );

        msg->setLength(1600);

        msg->setTimestamp(simtime);


              msg->setContextPointer((void *)serap);
```

```
                send( msg,"out" );

               wait((double) ia_time );

              }
  }

 void CorresNode::finish()
{
   ev << "*** Module: " << fullPath() << "***" << endl;
   ev << "Stack allocated:      " << stackSize() << " bytes";
   ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
endl;
   ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
}

//Server cpp
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "genpak.h"

class Server: public cSimpleModule
{
   Module_Class_Members(Server,cSimpleModule,16384*4)

   virtual void activity();
   virtual void finish();

};

 Define_Module( Server );

void Server::activity()
{
 int couna=0;
 int deccounta=0;
 int coun, deccount;
             coun=0;
             deccount=0;
             int k;
 double bits_per_sec= par("bits_per_sec");

 cPar& ia_time = par("ia_time");
 data* ser1=new data();
```

```
   for(;;)

   {

      cMessage *msg1 = receive();

      wait((double) ia_time );

            if (msg1->hasBitError()==true)

                  deccounta=deccounta+1;

               else

       couna=couna+1;
            ev<<"false mesage's number is"<<deccounta<<endl;
                ev<<"true mesage's number is"<<couna;

                        send(msg1,"outh");
                }
}

 void Server ::finish()

{


  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}
```

//Home Agent cpp

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include"genpak.h"
#include "fifo1.h"

 void FF1AbstractFifo::activity()
{

   msgServiced = NULL;
```

```
endServiceMsg = new cMessage("end-service");
queue.setName("queue");

 cPar& ia_time = par("ia_time");
 data* ser=new data();
        for(;;)
{

  cMessage *msg1 = receive();

   long k= msg1->length();

            ev<<msg1->length()<<endl;

          if (msg1==endServiceMsg)
  {
    endService( msgServiced );
    if (queue.empty())
{
      msgServiced = NULL;
    }

    else

    {

      msgServiced = (cMessage *) queue.pop();
      simtime_t serviceTime = startService( msgServiced );
      scheduleAt( simTime()+serviceTime, endServiceMsg );

    }

  }
  else if (!msgServiced)
  {
    arrival( msg1 );
    msgServiced = msg1;
    simtime_t serviceTime = startService( msgServiced );
                  ev<<"servicetime"<<serviceTime<<endl;
    scheduleAt( simTime()+serviceTime, endServiceMsg );

  }
  else
  {
    arrival( msg1 );
    queue.insert( msg1 );
  }

 }
```

```
}


void FF1AbstractFifo::finish()

{
  ev << "*** Module: " << fullPath() << "***" << endl;
  ev << endl;

}

Define_Module( FF1PacketFifo );

simtime_t FF1PacketFifo::startService(cMessage *msg1)
{
   ev << "Starting service of " << msg1->name() << endl;
   return par("service_time");
}

void FF1PacketFifo::endService(cMessage *msg1)
{
   ev << "Completed service of " << msg1->name() << endl;
   send( msg1, "out" );
}

Define_Module( HomeAgent);

simtime_t HomeAgent::startService(cMessage *msg1)
{
   ev << "Starting service of " << msg1->name() << endl;
   return msg1->length() / (double)par("bits_per_sec");
}

void HomeAgent::endService(cMessage *msg1)
{
   ev << "Completed service of " << msg1->name() << endl;
   send( msg1, "out" );
}
```

// Mobile Node

```
#include <omnetpp.h>
#include"global.h"
#include<string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include<coutvect.h>
```

```
#include"genpak.h"

#include "sink1.h"

int s;

 int coun=0;
 int deccount=0;

long double l;

long double delay[8002];

long double t[8002];

  FILE* fp;

  FILE* fs;

Define_Module( MobNode );

void MobNode::activity()
{

    qstats.setName("queuing time stats");
   cOutVector qtime("queueing time vector");

  long double t_delay=0;
  long double m=0;
  long double delay1=0;
  delay[0]=0.00000000000;


   t[0]=0;
            s=1;

  if( (fp  = fopen( "c:\\Omnet++\\mobhome\\pdata.txt", "wb+" )) == NULL )
     printf( "The file 'data' was not opened\n" );

 fseek(fp,0,0);

   if( (fs  = fopen( "c:\\Omnet++\\mobhome\\sdata.txt", "wb+" )) == NULL )
     printf( "The file 'data' was not opened\n" );

 fseek(fs,0,0);

         for(;;)

   {
```

```
    cMessage *msg = receive();

        msg->setTimestamp();

            if (msg->hasBitError()==true)    //(crc2==crc1)
                                    deccount=deccount+1;
            else

          coun=coun+1;
                ev<<"false mesage's number is"<<deccount<<endl;
                     ev<<"true mesage's number is"<<coun<<endl;

       long double d=msg->timestamp();

                     t[s]=d;
        fprintf(fs,"%f",d);
        fprintf(fs,"%c",'\n');

        delay[s]=t[s]-t[s-1];
                     ev<<"d="<<d<<endl;
                     ev<<"delays="<<delay[s];
        fprintf(fp,"%f",delay[s]);
        fprintf(fp,"%c",'\n');


        ev << "Received " << msg->name() << ", queueing time: " << d << "sec" <<
        endl;
        qtime.record( delay[s]/10000 );
        qstats.collect( delay[s]/10000 );
        delete msg;
        s=s+1;

      }

    fclose(fp);
    fclose(fs);

}

void MobNode::finish()

{
    ev << "*** Module: " << fullPath() << "***" << endl;

    ev << "Total jobs processed: " << qstats.samples() << endl;

    ev << "Avg queueing time:    " << qstats.mean() << endl;
```

```
    ev << "Max queueing time:    " << qstats.max() << endl;

    ev << "Standard deviation:   " << qstats.stddev() << endl;

    ev << endl;
    ev << "Stack allocated:      " << stackSize() << " bytes";
    ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
    endl;
    ev << "Stack actually used: " << stackUsage() << " bytes" << endl;
}


[General]
network = mobhome

random-seed = 1
sim-time-limit = 1000000000s
cpu-time-limit= 18000000s
ini-warnings = no

[Cmdenv]
runs-to-execute = 1,2
#express-mode = yes
module-messages = yes
event-banners = yes

[Tkenv]
default-run=1
use-mainwindow = yes
print-banners = yes
slowexec-delay = 300ms

[Parameters]
mobhome.gen.num_messages = 5000

[Run 1]
mobhome.gen.ia_time = exponential( 1 )
mobhome.gen.msg_length = intuniform( 5, 10)
mobhome.fifo.bits_per_sec = 10
output-vector-file = fifo1-r1.vec

[Run 2]
fifonet1.gen.ia_time = exponential( 0.7 )
fifonet1.gen.msg_length = intuniform( 5, 10)
fifonet1.fifo.bits_per_sec = 10
output-vector-file = fifo1-r2.vec

// Ned files:

module Module //When mobilen node in home network.
```

```
  submodules:
    CorresNode: CorresNode;
      display: "i=pc;p=135,41;b=40,28";
    HomeAgent: HomeAgent;
      display: "i=router;p=78,221;b=24,34";
    ForegAgent: ForegAgent;
      display: "i=router;p=364,168;b=40,28";
    MobNode: MobNode;
      display: "i=laptop;p=170,278;b=40,28";
    server: Server;
      display: "i=router;p=220,127;b=40,28";
  connections:
    server.outh --> delay 2us error 0.0999 --> HomeAgent.in display
"m=a,0,54,92,35";
    CorresNode.out --> delay 2us error 0.999999 datarate 1000000 --> server.in
display "m=a,93,46,60,-3"; //
    server.outf --> delay 2us error 0.1 datarate 1000000 --> ForegAgent.in display
"m=a,100,82,40,-6"; //
    HomeAgent.out --> Wireless --> MobNode.in display "m=a,92,59,45,0";
  display: "p=10,10;b=396,304";
endmodule
//CorresNode
//
//generator message (jobs,packetsetc)
simple CorresNode //
  parameters:
    num_messages : numeric,
    ia_time : numeric,
    msg_length : numeric;
  gates:
    out: out;
endsimple
simple HomeAgent //
  parameters:
    bits_per_sec : numeric,
    ia_time : numeric;
  gates:
    in: in;
    out: out;
endsimple
simple ForegAgent //
  parameters:
    bits_per_sec : numeric;
```

```
    gates:
        in: in;
endsimple
simple MobNode
    gates:
        in: in;
endsimple
channel Wireless //
    delay normal(0.01,0.003);
    error normal(0.003,0.001);
    datarate 1000000;
endchannel

network mobhome : Module
endnetwork
// FF1PacketFifo
//
// single server queue, same service rate for each packet
simple FF1PacketFifo //
    parameters:
        service_time : numeric;
    gates:
        in: in;
        out: out;
endsimple
//FF1Packet_Fifo
simple FF1Packet_Fifo //
    parameters:
        service_time : numeric;
    gates:
        in: in;
        out: out;
endsimple
simple Server //
    parameters:
        bits_per_sec : numeric,
        ia_time : numeric;
    gates:
        in: in;
        out: outf;
        out: outh;
endsimple

module Module// When mobile node in Foreign Network.
    submodules:
        CorresNode: CorresNode;
            display: "i=pc;p=135,41;b=40,28";
        HomeAgent: HomeAgent;
            display: "i=router;p=78,221;b=24,34";
```

```
        ForegAgent: ForegAgent;
            display: "i=router;p=348,168;b=32,32";
        MobNode: MobNode;
            display: "i=laptop;p=370,254;b=32,29";
        server: Server;
            display: "i=router;p=220,127;b=40,28";
    connections:
        ForegAgent.out --> Wireless --> MobNode.in display "m=s,40,107,65,0"; //
        server.outh --> delay 1us error 0.0999 --> HomeAgent.in display
"m=a,0,54,92,35";
        server.in <-- delay 1us error normal(0.99999 ,0.01) datarate 1000000 <--
HomeAgent.out; //
        CorresNode.out --> delay 1us error 0.999999 datarate 1000000 --> server.in1
display "m=a,93,46,60,-3"; //
        server.outf --> delay 1us error 0.1 datarate 1000000 --> ForegAgent.in display
"m=a,100,82,40,-6"; //
    display: "p=2,10;b=412,296";
endmodule
//CorresNode
//
//generator message (jobs,packetsetc)
simple CorresNode //
    parameters:
        num_messages : numeric,
        ia_time : numeric,
        msg_length : numeric;
    gates:
        out: out;
endsimple
simple HomeAgent //
    parameters:
        bits_per_sec : numeric,
        ia_time : numeric;
    gates:
        in: in;
        out: out;
endsimple
simple ForegAgent //
    parameters:
        bits_per_sec : numeric;
    gates:
        in: in;
        out: out;
endsimple
simple MobNode
    gates:
        in: in;
endsimple
channel Wireless //
```

```
    delay normal(0.01, 0.003);
    error normal(0.001,0.003);
    datarate 1000000;
endchannel

network fifoworo : Module
endnetwork
// FF1PacketFifo
//
// single server queue, same service rate for each packet
simple FF1PacketFifo //
    parameters:
        service_time : numeric;
    gates:
        in: in;
        out: out;
endsimple
//FF1Packet_Fifo
simple FF1Packet_Fifo //
    parameters:
        service_time : numeric;
    gates:
        in: in;
        out: out;
endsimple
simple Server
    parameters:
        bits_per_sec : numeric,
        ia_time : numeric;
    gates:
        in: in;
        out: outf;
        out: outh;
        in: in1;
endsimple
```